A Multi-dimensional Study on the State of the Practice of REST APIs Usage in Android Apps

Manel Abdellatif[1,2], Rafik Tighilt[1], Abdelkarim Belkhir[1], Naouel Moha[1], Yann-Gaël Guéhéneuc[3], and Éric Beaudry[1]

[1] Université du Québec à Montréal, Montréal, Québec, Canada
[2] Polytechnique Montréal, Montréal, Québec, Canada.
[3] Concordia University, Montréal, Québec, Canada

Abstract. REST APIs are gaining a tremendous attraction in industry and a growing usage in mobile platforms. They are well suited for providing content to apps running on small devices, like smartphones and tablets. Several research works studied REST APIs development practices for mobile apps. However, little is known about how Android apps use/consume these APIs in practice. Consequently, we propose a multi-dimensional study on the state of the practice of REST APIs usage in Android apps. We follow three directions: analysing of Android apps, mining Stack Overflow posts on REST APIs usage in Android apps, and surveying Android developers about their usage of REST APIs in their mobile apps. We (1) build a catalog of Android REST mobile clients practices, (2) propose an automatic approach to detect these practices, (3) analyze 1,595 Android apps downloaded from the Google Play store, (4) mine 12,478 Stack Overflow posts to study REST APIs usage in Android apps, and (5) conduct an online survey with 118 Android developers to understand their usage of these practices. We report that only two good practices are widely considered by Android developers when implementing their mobile apps. These practices are network connectivity awareness and JSON vs. XML response parsing. We also report Android developers' recommendations for the use of third-party HTTP libraries and their role in implementing the recommended practices.

Keywords: REST API · Android · Practices · third-party HTTP libraries

1 Introduction

Smartphone ownership has spread rapidly around the globe in the past decades. It is estimated that, in 2019, the total number of smartphone users stood at more than 3.8 billion[4]. Also, in the same year, Google Play store boosted close to 2.4 millions of mobile applications (apps) while the Apple App Store has over 2 millions[5]. Most of these mobile apps access data, business rules, and business

[4] https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

[5] https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/

processes of vital information-systems remotely for architectural, efficiency, and security reasons.

REST APIs (Representational State Transfer Application Programming Interfaces) have been a mainstream form of services for some years now. They are well suited for providing content to mobile devices, like smartphones and tablets, because they offer a lightweight and flexible access to remote services for end users and mobile apps developers.

Several research works studied best practices in the implementation of REST APIs [1,2,3,4]. However, little is known on how Android apps use/consume these APIs in practice. HTTP requests to REST APIs were first implemented for mobile apps with "low-level" APIs provided by programming languages (sockets) or class libraries (HttpURLConnection). However, these requests can now be implemented using dedicated, third-party HTTP libraries, such as Google Volley, OkHttp, or Retrofit.

Consequently, we ask in this paper how dedicated HTTP libraries are used in practice and if they help adopt the recommended practices of REST APIs usage in Android apps. We investigate "good" and "poor" practices of REST APIs usage in Android apps. We also study the prevalence of these practices and how they are used. We perform a multi-dimensional study of the state of the practice of REST APIs usage in Android apps. We answer the following research questions:

- **RQ1:** What is the state of the practice in the use of REST APIs by Android apps? We want to observe the use of practices by developers and their prevalence.
- **RQ2:** What is the state of the implementation of HTTP libraries in Android apps? We want to observe the implementations used by developers for REST APIs, which will inform the choice of HTTP client libraries.
- **RQ3:** Do third-party HTTP libraries help developers adopt the recommended practices of REST APIs usage in Android apps? We want to study if third-party HTTP libraries promote the use of the recommended practices of REST APIs usage in Android apps and study as well the recommended libraries for each practice.

By answering these research questions, we want to recommend to developers the libraries and practices to adopt based on their prevalence in implementations and their benefits for developers. Such recommendations are important for REST APIs providers because (1) mobile apps run on mobile devices that have many constraints in terms of memory, battery, computational power, and competition for resources, (2) they would help them offer more features that ease the Android developers' work, and (3) they would help them improve the usability and performance of the third-party HTTP libraries [5,6].

This paper presents an extension to our previous observational study [6]. We additionally conduct an online survey to assess the importance of the identified practices from Android developers' point of view. We study in detail why developers consider/ignore the recommended practices of REST APIs usage in

Android apps. We also study the role of third-party HTTP libraries in promoting the implementation of the good practices from the point of view of Android developers.

Therefore, we provide a total of five contributions. We review the literature extensively and compile a catalog of seven practices related to the development of mobile apps that consume REST APIs. These practices pertain to the use of dedicated, third-party HTTP libraries and help to understand how Android apps use/consume REST APIs. Then, we propose a framework, PIRAC, that automatically detects occurrences of all these practices using detection rules. Then, we perform an observational study on 1,595 Android apps out of 9,173 apps downloaded from the Google Play store to report how they use/consume REST APIs. We mine 12,478 Stack Overflow posts to assess the importance of the identified practices from Android developers' point of view. Finally, we conduct an online survey with 118 Android developers to study further the usage of these practices, the reasons of ignoring them as well as the developers' recommendations for using third-party HTTP libraries.

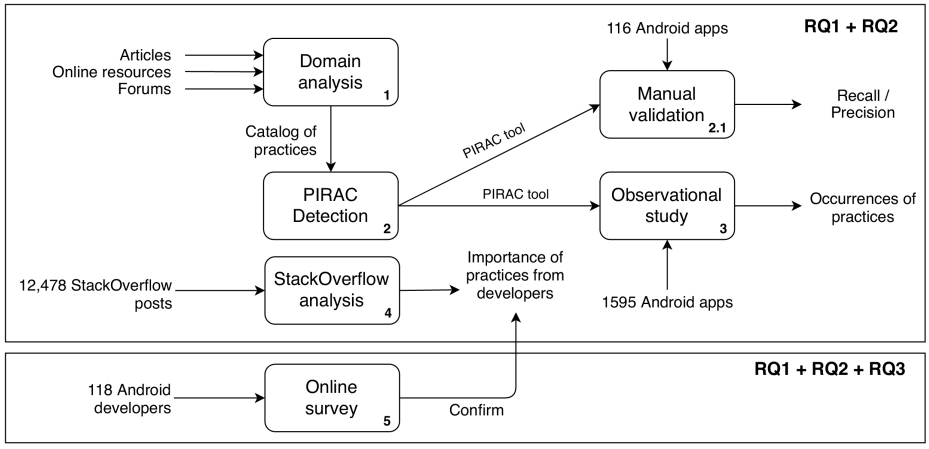

Fig. 1. Methodology of our multi-dimensional study

Figure 1 outlines our methodology to realise these contributions and response to our research questions. It shows that we pursued a three-prong approach by (1) developing a tool, PIRAC, to identify occurrences of REST mobile clients practices, which we analyse manually the precision and recall; (2) performing an observational study on the occurrences of these practices in real Android apps using PIRAC; and, (3) confirming the importance of these practices through a study of Stack Overflow posts related to Android REST clients and a survey of developers' perception of these practices.

We show in this paper that developers tend to ignore some good practices of REST APIs usage in Android clients: caching responses, timeout management,

and error handling. We also report that only two good practices are widely considered by Android developers: network connectivity awareness and JSON vs. XML responses parsing. We show that third-party HTTP libraries promote the use of recommended practices of REST APIs usage in Android apps.

The remainder of this paper highlights the multi-dimensionality of our work as follows. Section 2 describes related works. The next three sections follow the same plan: study design, study results, threats to their validity for, in Section 3, the practices, in Section 4, the developers' questions on Stack Overflow, and, in Section 5, the developers' answers to a survey. Finally, Section 6 answers and discusses our research questions while Section 8 concludes with future work.

2 Related Work

Several research works have been proposed in the literature on bad and good practices in REST APIs. However, few are the works that study how Android apps use/consume REST APIs.

In the context of mobile apps, Rodriguez *et al.* [1] were the first to study the traffic of HTTP requests from mobile clients. They evaluated the conformance of some state-of-the-art design best practices of REST APIs from the perspective of mobile clients. They analyzed these practices on a large dataset of 78 GB of HTTP requests collected from a mobile-Internet traffic-monitoring site. However, the best practices analyzed are common to any kinds of REST APIs and they focused specifically on HTTP requests.

Oumaziz *et al.* [5] conducted an empirical study on 500 popular Android apps and 15 popular services to identify best practices when using/consuming REST APIs for Android mobile clients. They showed that Android clients generally favour invoking REST APIs by using official dedicated service libraries instead of invoking services with a generic HTTP client like HttpURLConnection. They also presented which good practices service libraries should be implemented following an online survey and manual analyses of the apps. In this paper, we go more in details to identify how dedicated service libraries are used by Android clients. We propose a tool to automate the detection of these practices in Android mobile apps. We empirically analyze more than 9,000 Android mobile clients to study the usage of REST APIs in Android mobile apps. We mine Stack Overflow posts and conduct an online survey with 118 Android developers to assess the importance of the identified practices from the point of view of Android developers.

Several works were carried out for the detection of service anti-patterns [1,2,3,4,7,8]. For example, in the context of REST APIs implementation, Palma *et al.* evaluated the design of several REST APIs and proposed different approaches to detect automatically REST (anti)patterns. They proposed SODA-R (Service Oriented Detection for Anti-patterns in REST) [4], a heuristics-based approach to detect (anti)patterns in REST systems. They relied on heuristics and detection rules for eight REST anti-patterns and five patterns. They applied their tool on a set of 12 widely-used REST APIs including BestBuy, DropBox,

and Facebook. Then, Palma *et al.* proposed a syntactic and semantic approach to detect REST linguistic (anti-)patterns, which they define as poor/good practices in the naming, documentation, and choice of identifiers in REST APIs [8]. Finally, Palma *et al.* proposed UniDoSA [7], a unified approach that (1) embeds a unified meta-model for the three main service technologies REST, SCA, and SOAP and (2) detects the presence of anti-patterns in service-based systems.

A set of automatic approaches for the detection of REST (anti)patterns are proposed in these works. However, they specifically evaluated APIs without considering any interaction with clients, in particular mobile clients, as we do here. Other works proposed similar (anti)patterns detection approaches in service applications. They implemented other techniques, such as bi-level optimization problems [9] or ontologies [10].

In contrast, in this paper, we consider and automatically detect practices related to the development of REST mobile clients. We take also into account the interactions among clients and REST APIs, not on the service side but on the client one. We study in details the use of REST APIs libraries for mobile clients and how Android mobile apps use/consume REST APIs.

3 Study of the Practices of REST APIs Usage in Android Apps

In this section we will describe the first dimension of our study that focus on the analysis of the practices of REST APIs usage in Android apps. We will detail the study design, the obtained results as well as its related threats to validity.

3.1 Study Design

We conduct an observational study on hundreds of Android apps from the Google Play store. First, we reviewed the literature and developers' forums to build a catalog of Android REST clients practices. Second, we developed a tool, PIRAC[6], to detect each of the identified practices. Third, we validated the detection precision of PIRAC on 116 Android apps collected from the F-Droid repository. Fourth, we applied PIRAC on 9,173 mobile apps downloaded from Google Play store to study the state of the practices in Android clients.

3.1.1 Step 1: Cataloguing Android REST Mobile Clients Practices

To answer our first research question, we performed a domain analysis of the development practices related to Android REST clients. We studied the definitions and specifications of the practices in the literature as well as in online resources and articles. We executed a search query on Google, Scopus, IEEE Xplore, and Engineering Village to obtain references about Android REST mobile clients practices. The search query was: *Android And (REST OR RESTful OR HTTP)*

[6] http://git.sofa.uqam.ca/mabdellatif/PIRAC/tree/master

And (API OR Service OR Webservice). We excluded references about the design of REST APIs for mobile apps and kept only references that described how Android apps should consume REST APIs. We extracted from these references all practices related to Android REST mobile clients and classified these practices as good, neutral, or bad. This domain analysis allowed us to identify seven practices described in the following.

List of Good and Bad Practices for Android REST Clients

1. ✓ **Use of third-party HTTP client vs. ✗ HttpURLConnection:** This practice concerns the use of third-party libraries to manage REST requests. It is recommended that mobile HTTP queries should be encapsulated in a method proposed by the interface of official third-party libraries, such as OkHttp, Retrofit, Google Volley, etc. A *Non-encapsulated HTTP Query* must be manually built by the developer with all the needed parameters using HttpURLConnection. This process could be long and complicated in some cases and could make the code difficult to maintain.
2. ✓ **Network connectivity aware vs. ✗ Unaware REST service invocation:** This practice pertains to the validation of the network connectivity before sending REST request. It is recommended to check network connectivity (1) to offload heavy REST queries when the device is connected to WiFi, (2) to increase device battery life, (3) to avoid charges related to limited mobile data, and (4) to detect network changes and resume incomplete REST requests.
3. ✓ **Timeouts vs. ✗ Perpetual requests:** This practice is related to setting or not timeouts for REST requests. There are several types of timeouts: connection timeout, read timeout, write timeout, etc. If a mobile client fails to establish a connection to the server within the set connection timeout, it will consider that the request failed. It is recommended to set proper timeouts values to make mobile apps more responsive and user friendly.
4. ✓ **JSON vs. ✗ XML:** This practice pertains to REST responses parsing by mobile clients. It is recommended to parse REST responses with JSON as it is more human-readable than XML. Also, JSON is more CPU-friendly to parse as it is more compact than XML [11,12].
5. ✓ **Caching vs. ✗ Non caching:** Caching is the ability to keep copies of frequently accessed data in several places along the request–response path. Some third-party Android REST libraries offer facilities to manage response caching, such as removing a single cached response, clearing the entire cache, retrieving the date of a cached response, so that developers can accurately decide when an update should be made. It is recommended to cache frequent REST requests to reduce bandwidth usage, network latency, and battery consumption.
6. ✓ **Specification vs. ✗ Non specification of a behaviour for failed requests:** The specification of a behaviour when REST requests fail is highly recommended to increase usability and responsiveness of the mobile client apps. Possible behaviours include to drop the requests until some change

to the network connectivity or to retry the requests in some chosen time-intervals until successfully, etc.

List of Neutral Practices for Android REST Clients

1. **Synchronous vs. Asynchronous requests:** REST APIs requests can be synchronous or asynchronous. For synchronous requests, the code execution will block until the API call returns. For asynchronous requests, calls to remote APIs are made while the execution continues. Android developers should carefully choose whether to invoke REST APIs synchronously or asynchronously based on their needs to increase the responsiveness of their apps.

3.1.2 Step 2: Detection of the Practices

We developed a framework, PIRAC, to detect the seven identified practices. As depicted in Figure 2, our framework takes as input Android APKs, their corresponding meta-data, and a list of HTTP libraries, which we use to filter the code to analyze. Our tool uses the SOOT framework [13] to parse the bytecode of mobile apps and extract all the information needed for our analyses, such as classes and methods. PIRAC creates models for Android mobile apps based on the information extracted by SOOT,and those extracted from the reconstructed manifest file. Then, we apply the detection algorithms for the identified practices to detect their uses.

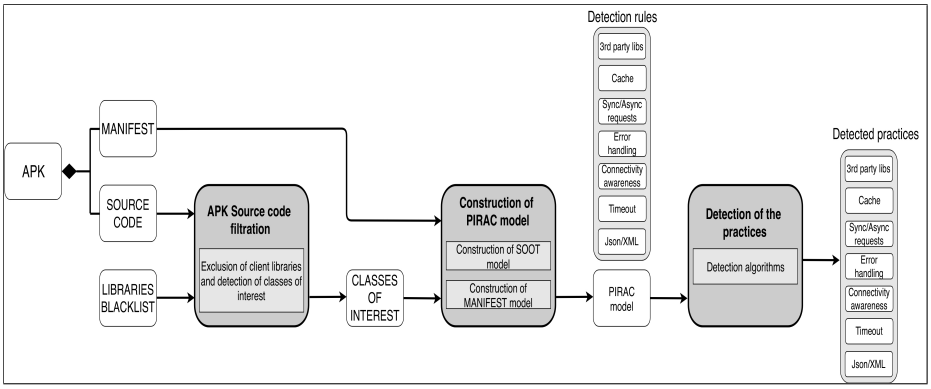

Fig. 2. Detection of REST APIs practices usage in Android apps with PIRAC

An Android APK contains the compiled source code of the app as well as that of third-party and Android libraries. Running our analyses on the entire packaged code would (1) produce misleading results, and (2) affect the execution time of our analyses. Thus, we filter the application code to differentiate the code

of the app currently under analysis from the code belonging to Android SDK and third-party libraries. We rely on a *list* of third-party Android libraries, which contains 1,353 package names of the most used libraries identified by Li *et al.* [14]. This list has not been updated since 2016 so we updated it by adding 1,176 package names of the libraries that we manually collected from Android community Web sites[7].

After filtering the app code, we construct models of the APKs that embed all the required information to apply our detection heuristics. Afterwards, we identify classes of interest that are related to REST APIs services calls. Finally, we analyze these classes and identify the practices based on our detection rules.

In the following, we describe some of the detection rules that we use in our framework.

Android REST Clients Identification. Identifying automatically Android apps that make REST calls can be a very complicated task, especially with the use of static analyses methods. Indeed, we can only rely on some used practices in the apps source code to verify whether a given app is potentially using a REST API. We rely primarily on these rules:

1. *The use of Android INTERNET permission.* Android apps require Internet permission to access the mobile network. This information is explicitly defined in the Android manifest file.
2. *The referencing of an HTTP library.* The communication with REST APIs is primarily based on the HTTP protocol. REST apps must use an HTTP library to communicate through this protocol. We rely on a list of 75 HTTP libraries collected from a Maven repository[8].

When we detect these practices in an app, it is automatically marked as *"Potentially Using a REST API"*. For a better accuracy of the detection of our targeted practices, we ensured that the HTTP library is executing REST calls and that it is not just referenced for other purposes/uselessly (dead code). Indeed, the simple presence of an HTTP library in the APK file does not guarantee that this app uses/consumes REST APIs. Also, some Android apps may reference an HTTP library to use only some of its classes without executing any REST calls.

Use of Third-party HTTP Library vs. HttpURLConnection. When it comes to executing HTTP calls, developers can rely on the native HttpURLConnection API or choose to use an external HTTP library. Our tool detects the usage of HttpURLConnection or an external library by analyzing the instantiation of objects and calls made with the specific Java methods of each of the libraries.

[7] https://android-arsenal.com/

[8] https://mvnrepository.com/open-source/http-clients

Cached vs. Non-cached Responses. Developers can use the caching capabilities offered by the HTTP libraries or develop their own caching strategy. To detect response caching, PIRAC detects the use of relevant methods and classes provided by the libraries, which allow such operations. PIRAC detects also the creation and use of caching folders dedicated to Android apps.

Network-connectivity Aware vs. Unaware REST Requests. The Android SDK provides a class named `ConnectivityManager`, which provides information about the network connectivity of a device (network type, availability, etc.). Developers can use this class to adapt their use of REST API requests. To detect such behaviour, our tool detects the invocations of methods that provide information about network connectivity from the *ConnectivityManager* class.

JSON vs. XML Response Parsing. The responses from REST APIs come in multiple formats, mainly JSON and XML. The use of JSON is recommended due to its size relatively to XML, its readability by developers, and its ease of use. To detect the usage of one of these two data formats, our framework detects the usage of the most common JSON and XML libraries as well as their instantiation in a code executing an HTTP call.

Timeout vs. Perpetual REST Requests. Each one of the studied client libraries provide classes or methods to configure a timeout for their HTTP requests. To detect a timeout configuration, PIRAC detects invocations of these methods or instantiation of classes with a timeout value in the constructor.

Specification vs. Non-specification of a Behaviour Upon Failure. When a request fails, regardless of the reason, developers should implement custom logic and show an adapted message to the end user. To detect this behaviour, in the case of `HttpURLConnection`, we detect the retrieval of HTTP status codes after a request. For external libraries, we detect the usage of specific library error-handling methods (e.g., `onFailure()` or `response.isSuccessful()`).

Synchronous vs. Asynchronous Requests. When using third-party libraries, it is simple to perform asynchronous requests because these libraries offer specific methods with callbacks. To detect synchronous/asynchronous requests from third-party libraries, we implemented a detection approach specific to each kind of libraries. We rely on the detection of third-party REST clients methods dedicated to synchronous/asynchronous REST requests. The detection of asynchronous requests for Java `HttpURLConnection` is more challenging. When using this library, developers must customize and hard-code asynchronous requests, most commonly using the `AsyncTask` class, which provides methods with callbacks. We analyze the bodies of the methods running in the background to build a method-invocation call graph. Finally, we search for a REST call in each of these methods.

3.1.3 Step 3: Validating PIRAC for the Detection of REST Mobile Clients and their Practices

For our validation, we analysed 1,448 Android apps from the F-Droid repository. We applied our tool on a dataset available online for replications[9,10]. We randomly selected a sample of 116 apps from this dataset, which represents a 95% statistically significant stratified sample with $> 10\%$ confidence interval of the 1,448 analysed apps in our dataset [15].

After the selection, one author studied manually the source code of the 116 apps to check that the tool correctly detected Android REST clients as well as our targeted practices. The author checked the presence of Internet permission in the APK manifest file and the usage of some HTTP library in the source code. The same author extracted the URLs of the HTTP requests and manually checked if the calls were made to some REST API. Then, he also manually checked if we did not miss any occurrence of the practices in these apps by studying the HTTP requests in the app source code.

Table 1 summarises the detection precision and recall values of our tool. We reached a detection precision of Android REST clients of 100% and a recall of 41%. Although our recall is low, we are confident that the selected apps are indeed REST clients through our manual analysis. The detection precision for each targeted practice is satisfactory as it varies between 80.44% and 100%. The recall of our tool for the practices is also satisfactory as it varies between 81.2% and 96%. We reached an average detection precision of 93.92% and an average detection recall of 81.26%. These detection results confirm the reliability of our tool to detect our targeted practices.

	Precision	**Recall**
Identification of Android REST clients	100%	41%
Use of third-party HTTP Library vs. HttpURLConnection	100%	81.2%
Cache usage	98%	90.72%
Connectivity aware clients	97.42%	96%
JSON vs. XML	83.15%	85.74%
Timeout setting	92.31%	88.41%
Specification of a behavior at request failure	80.44%	81.98%
Synchronous vs. asynchronous calls	100%	85%
Average	**93.92%**	**81.26%**

Table 1. Overview of the detection precision of our tool

3.2 Study Results

We now describe our observations about our dataset and the identified practices.

[9] https://github.com/rtighilt/PIRAC-Android-Dataset/blob/master/REST-Client-Validation-apps.csv

[10] http://www.ptidej.net/downloads/replications/ause20/

3.2.1 Dataset

To conduct our observational study, we downloaded randomly 9,173 Android apps from Google Play. We used Androzoo[11] to collect APKs that were originally extracted from Google Play. Then, for each app, we relied on Google Play to extract their metadata, including the APK name, its date of release, its size, its categories, etc. As depicted in Figure 3, the apps belong to varied categories, such as games, communication, weather, etc. We applied PIRAC on the dataset to extract all REST clients for a total number of 1,595 Android REST clients[12,13]. These apps are also of different sizes, as shown in Figure 4.

Observation 1: We observe that the three main categories that use the most Android REST clients are *Lifestyle*, *Business & Finance*, and *Video & Media*.

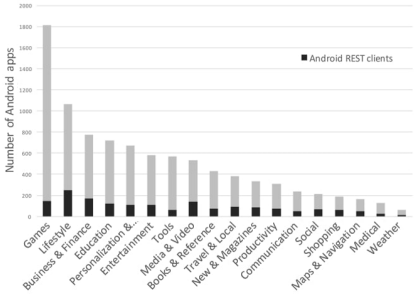

Fig. 3. Distribution of Android apps by category

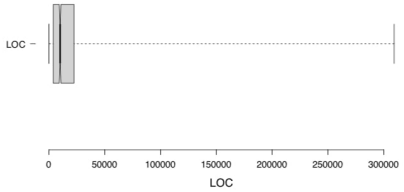

Fig. 4. Android REST clients LOCs

3.2.2 Observations on the Identified Practices

In this section, we present our observations about the distribution of good and bad practices of REST APIs in Android clients.

Use vs. Non-use of Third-party Libraries for HTTP Requests. Based on our previous work [5], the most used Android libraries to execute HTTP requests are OkHttp, Retrofit, Google Volley, and Java HttpURLConnection. We focused our study on these libraries. We relied on the APK metadata to extract the date of release of the apps. We reported in Figure 5 the evolution of the percentage of the apps that use a particular HTTP library by year.

[11] https://androzoo.uni.lu/

[12] https://github.com/rtighilt/PIRAC-Android-Dataset/blob/master/pirac-analysed-dataset.csv

[13] http://www.ptidej.net/downloads/replications/ause20/

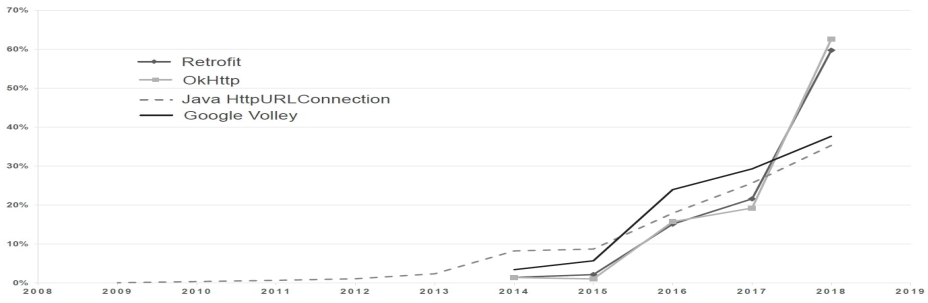

Fig. 5. Evolution of the use of HTTP libraries in time

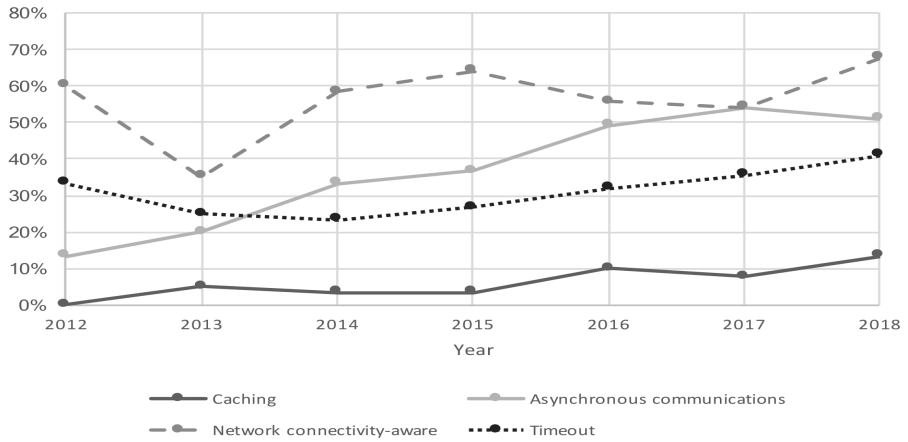

Fig. 6. Evolution of the practices in time

We noticed that Java HttpURLConnection is the oldest library to execute HTTP requests in REST Android clients. With Figure 5, we noticed that HttpURL-Connection is getting less used by developers compared to newer third-party libraries (i.e., OkHttp, Retrofit, and Google Volley).

In 2014, OkHttp, Retrofit, and Google Volley have been released to ease and simplify the management of HTTP requests by Android clients. We observe that between 2014 and 2018, the usage evolution of OkHttp and Retrofit is almost the same because Retrofit uses the OkHttp library for HTTP requests. The usage evolution of these two libraries increased rapidly (60% and 64%, respectively) between 2017 and 2018 compared to Google Volley (37%) and Java HttpURLConnection (35%).

Observation 2: Recently, Android developers have been using OkHttp, Retrofit, and Google Volley to manage REST requests because they offer more interesting features compared to Java HttpURLConnection. For example, implementing asynchronous requests is easier when using third-party libraries because developers do not need to use Android `AsyncTasks` to run network operations in a separate thread.

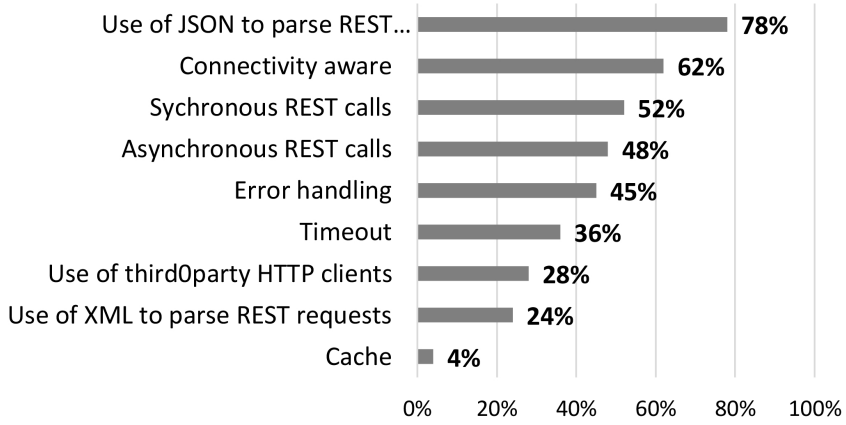

Fig. 7. Distribution of Android REST client practices

Cached vs. Non-cached Responses. Although caching helps developers implement highly capable and scalable REST clients and services by limiting repetitive interactions, REST Android developers widely ignore the caching capability for their REST requests. Based on Figure 7, we observe that only 4% of REST Android clients cache REST responses, which forces the apps to retrieve duplicate responses from servers. This bad practice is known as *Ignoring Caching* anti-pattern [4,16]. Also, based on Figure 6, we observe that the use of response caching has been rapidly increasing during the past five years.

As depicted in Figure 9, we observe that Android apps using third-party libraries tend to consider more frequently the use of caching (85% of Android apps considering caching are using third-party libraries).

Observation 3: Android developers tend to ignore the use of caching to manage REST requests. However, Android apps using third-party libraries tend to consider the usage of caching more.

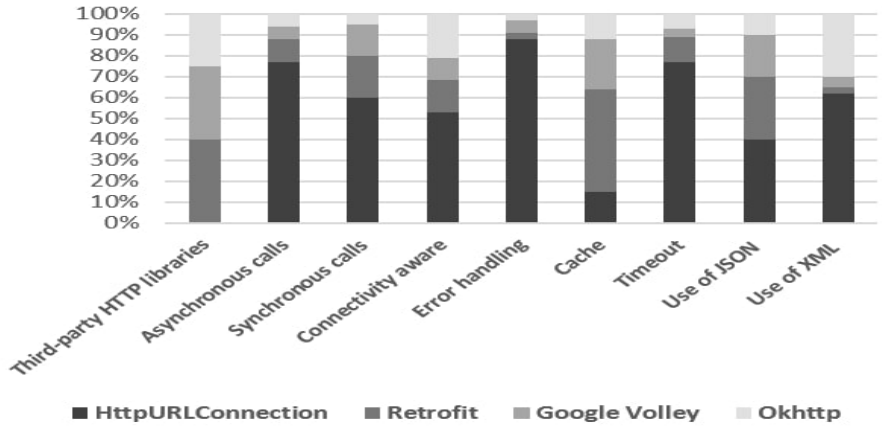

Fig. 8. Distribution of Android HTTP libraries by practices

Network-connectivity Aware vs. Unaware REST requests. Based on Figure 7, we observe that 62% of REST Android clients are aware of/check network connectivity before performing REST requests, which is a good practice with several advantages. Checking network connectivity first lets mobile clients improve battery life by offloading heavy network requests when the device is connected to WiFi, for example.

> **Observation 4:** The check of network connectivity before performing REST requests is a good common practice for Android REST clients.

JSON vs. XML Response Parsing. Using JSON to communicate via REST APIs is highly recommended in comparison to XML due to its ease of use. Based on Figure 7, we observe that this good practice is widely adopted by Android developers with 78% of Android REST clients using JSON to parse REST APIs responses while only 24% use XML.

> **Observation 5:** The good practice of using JSON to communicate REST APIs responses is widely adopted by Android developers.

Timeout vs. Perpetual REST Requests. Based on Figure 7, we observe that only 36% of Android REST clients consider setting timeouts when performing HTTP requests. As depicted in Figure 10, we also observe that 77% of Android REST clients that consider timeouts use HttpURLConnection. This high percentage is due to HttpURLConnection being the only studied HTTP library that does not specify a default value for timeouts. When using such a library, developers must specify a value of timeout.

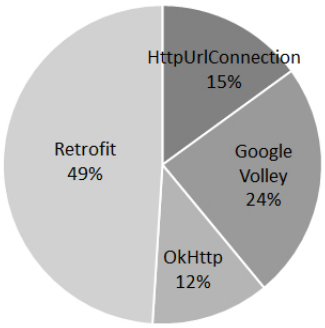

Fig. 9. Cache usage by Android HTTP library

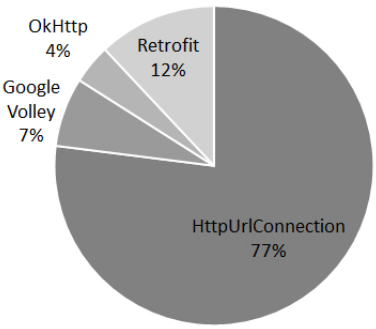

Fig. 10. Timeout usage by client library

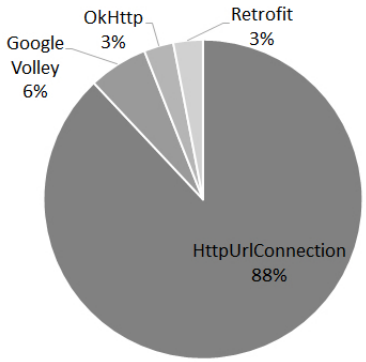

Fig. 11. Error-handling usage by Android HTTP client library

Observation 6: Although it is recommended to set proper timeouts values to make the mobile apps more responsive and user friendly, Android developers tend to ignore this good practice.

Specification vs. Non-specification of a Behaviour Upon Failure. Based on Figures 7 and 11, we observe that only 45% of Android REST clients handle HTTP requests failures when calling REST APIs. We also observe that Android developers do not take full benefit from the error-handling features provided by third-party libraries as 88% of error handling in Android apps are implemented with HttpURLConnection. The wide use of HttpURLConnection may hinder the

adoption of such a good practice as developers must implement themselves how to catch the requests failures and specify what should happen in case of failures.

Observation 7: The specification of a behaviour in case of request failures is widely ignored by Android developers as they poorly use third-party HTTP libraries that facilitate errors handling.

Synchronous vs. Asynchronous REST Requests. Based on Figure 7, we observe that synchronous and asynchronous calls to REST APIs are almost equally used by Android clients. We also observe that asynchronous communications increased by almost 40% in the past five years thanks to the features offered by third-party libraries: Android developers do not have to manage asynchronous calls by themselves anymore. However, Android developers still poorly rely on third-party HTTP libraries to make asynchronous calls as 77% of asynchronous communications are made with HttpURLConnection.

Observation 8: Synchronous and asynchronous calls to REST APIs are almost equally used by Android clients. Also, we observe that Android developers are still relying on third-party HTTP libraries to make asynchronous calls.

3.3 Threats to Validity

In this section, we discuss the threats to validity of our study analysis of Android apps and the measures that we took to limit them.

Internal Validity. Although we did not carry any statistical tests, we assume that the identified practices are representative characteristics of the Android REST clients. There may be other characteristics that describe more accurately these Android REST API clients. We will study more practices to cover possible other characteristics. We also related the practices manually thanks to the information provided in the literature. Yet, other researchers should perform similar analyses to confirm/infirm ours.

To detect REST clients, our tool only checked if the apps have Internet access permission and use some HTTP library. The tool does not distinguish between REST and other HTTP requests as (1) official online documentation of libraries and network queries on Android platform assume that REST is the main reason to perform HTTP requests, (2) HTTP requests towards REST API is the recommended way for an app to interact with some server [5,17]. Our tool could not statically analyse the strings use to encode the URL of REST APIs. To the best of our knowledge, there exists no such tool available. Moreover, to mitigate this limitation, we analyse a sample of the studied Android apps and found a precision and a recall of 100% and 41%, which we deemed good enough for our study.

We did not involve developers in the validation of our tool because this validation is straightforward and simply consists in mining the source code of the APKs. In future work, we will involve the developers of the analysed apps (1) to validate the detection results of our tool and (2) to study possible improvements of the tool to efficiently assist the developers in the use of the recommended practices.

The source code of some of the apps in our dataset is obfuscated. Even a simple obfuscation affects our detection precision because names of packages, classes, and methods change. Also, we do not detect custom caching strategies as we only detect cache-related and asynchronous calls using third-party libraries. Custom caching and synchronisation strategies are challenging to cover because they vary among apps. More research should be done to cover such custom implementations.

External Validity. These threats concern the generalizability of our results. Although we presented the largest study on the practices and implementations of Android REST apps, we cannot generalise our results to all mobile apps. Future work is necessary to analyze more mobile REST clients, from other mobile platforms to confirm and—or infirm our observations on their design quality characteristics.

4 Study of the Practices of REST APIs Usage in Stack Overflow

We detail in this section the second dimension of our study that focuses on mining and analyzing Stack Overflow posts about REST APIs usage in Android apps.

4.1 Study Design

We mined Stack Overflow posts to assess the importance of the identified practices from the developers' point of view. We relied on the tags in Stack Overflow to extract questions related to the use of REST APIs in Android apps. We used several keywords for the tags that are related to the practices (e.g., Android, REST, HTTP, cache, timeout, synchronous, asynchronous, etc.) joined with the name of each HTTP library (HttpURLConnection, Google Volley, Retrofit, OkHttp). We obtained a total number of 12,478 posts related to REST APIs usage practices in Android apps. To verify the correctness of the tagged posts, two of the authors performed a manual validation by comparing the tags of the posts with their content. Because of the large number of posts, we adopted a sampling method to select a validation set [15]. We examined 373 posts to ensure that the studied posts yield a confidence interval of 5% and a confidence level of 95%. The two authors independently evaluated the posts and verified the correctness of their related tags. We then computed the Cohen Kappa metric [18] to evaluate the inter-rater agreement, which was 0.91 for all the evaluated

tagged posts, which indicated almost perfect agreement [19]. The two authors discussed the posts for which they disagreed to obtain a final agreement. Thus, we found that 87% of the mined posts were correctly tagged and consequently we decided to rely on the posts tags to map the 12,478 posts to our targeted practices.

4.2 Study Results

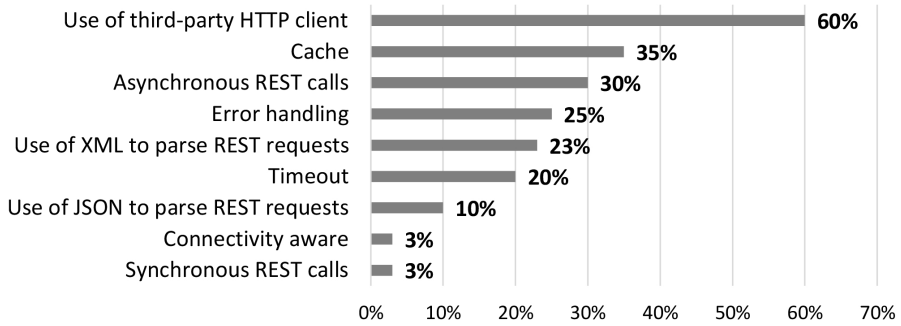

Fig. 12. Distribution of the questions on Stack Overflow about the identified practices

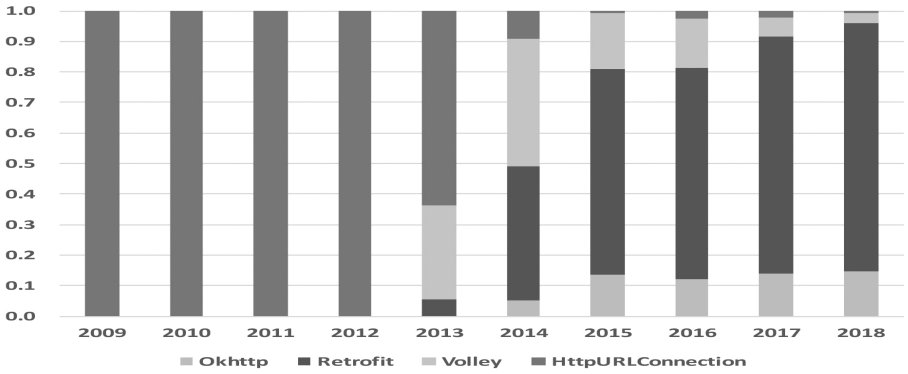

Fig. 13. Evolution of the questions on Stack Overflow about Android HTTP libraries over the years

We provide Figures 12 to 15 to report the results of our analysis of the 12,478 posts Stack Overflow posts.

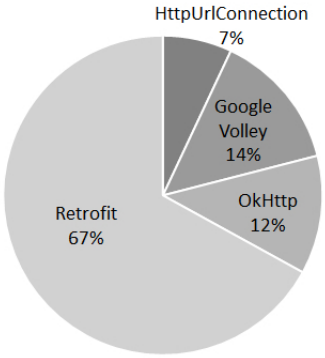

Fig. 14. Distribution of the questions on Stack Overflow about Android HTTP libraries

Figure 12 show the percentages of questions pertaining to the different practices among the mined posts. We conclude from this figure that the most discussed practice on Stack Overflow by Android developers is the use of third-party HTTP libraries. Android developers seem more interested about using third-party HTTP libraries to ease the management of HTTP requests.

Figures 13 and 14 show the distributions of the questions in Stack Overflow per Android HTTP library. Based on Figure 13, we report that, since the apparition of Android third-party HTTP libraries in 2013, the number of questions about HttpURLConnection has been significantly decreasing.

Observation 9: Since the apparition of third-party HTTP libraries, Android developers ask more questions about them in comparison to HttpURLConnection.

Figure 14 shows that there is a high interest in using Android HTTP libraries by Android REST clients developers to implement the identified practices. We found that Android developers ask most questions about REST requests, when using Retrofit (67%). Also, we found that 14% of the questions were about Google Volley, 12% were about OkHttp, and only 7% were about HttpURLConnection. We explain the high interest by Android developers in Retrofit by the poor documentation of this library in comparison to its features and its high performance in comparison to OkHttp, Google Volley, and HttpURLConnection[14].

There is also a concern by Android developers on Stack Overflow about asynchronous (30%) vs. synchronous (2%) HTTP requests to REST APIs because (1) asynchronous communications with REST APIs have more complicated implementation than synchronous ones, and (2) the multiple ways provided by HTTP libraries to handle such kind of requests are confusing to novice developers. Android developers seem more interested in managing such calls with

[14]https://bit.ly/2ypkEl9

third-party HTTP libraries (mainly Retrofit as shown in Figure 15) as they offer more straightforward ways to manage such kind of requests in comparison to HttpURLConnection.

We report that Android developers do not ask many questions about the “connectivity aware" request practice (only 2%), probably because it is very simple to implement. Also, we observe that Android developers do not ask many questions about parsing REST APIs responses with JSON. The more problematic parsing format is XML with 25% of the questions about XML parsing and only 9% are about JSON.

Observation 10: There is a high interest on Stack Overflow in using third-party HTTP libraries for Android REST clients wrt. asynchronous calls, caching responses, error handling, and XML parsing.

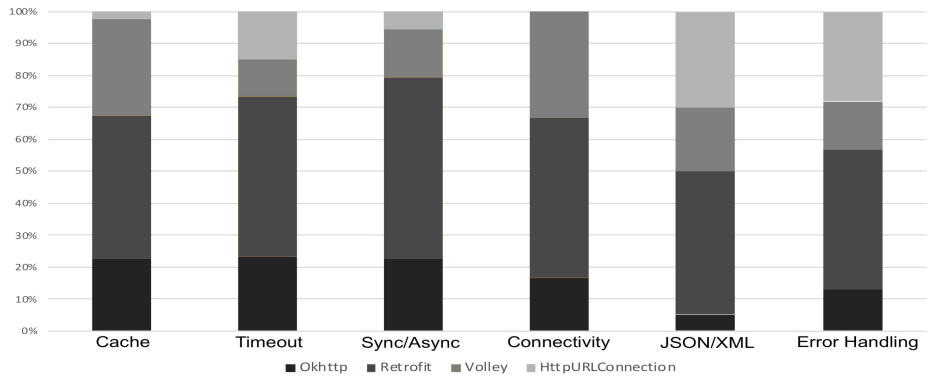

Fig. 15. Distribution of the questions on Stack Overflow about Android HTTP libraries by practice

4.3 Threats to Validity

In this section, we discuss the threats to validity of our observational study and the measures that we took to limit them.

Internal Validity. Although we used several search queries to collect our data, we may have missed some interesting posts about REST APIs usage in Android apps. To mitigate this risk, we used a combination and keywords to gather the most significant posts on Stack Overflow.

We relied partly on developer-provided tags to filter out irrelevant posts. These tags may not reflect accurately the contents of the posts and some relevant posts may not have been tagged properly. We mitigated this threats by studying 12,478 posts in an attempt to apply the law of large numbers to our study.

Also, we do not consider any filtering process for duplicate posts as they are useful for our study: they show the developers' interest in using/adopting a particular REST practice.

External Validity. There is a threat to the generalizability of our results as we only consider posts on Stack Overflow. However, we considered the most popular forum for software development and analysed a large number of posts that are related to REST APIs usage in Android apps. To further generalize our results, we should consider other developers' forums, such as GitHub, Quora, or Reddit.

5 Study of the Practices of REST APIs Usage with Android Developers

We conducted an online survey between September 2019 and November 2019 to collect and understand Android developers' point of view regarding the studied practices of REST APIs usage in Android apps.

5.1 Study Design

Our study consisted of four main phases:

A - Preparation of the Survey. We created a Web-based survey[15] using Google Forms based on our literature review, our prior works [6,5,20], and informal discussions with some Android developers who actively use REST APIs in their mobile apps. We thus could choose the focus of the survey, the individual questions, and the possible answers for each question. Before publishing the survey, we performed a pilot with eight developers, four from academia and four from industry, to validate the relevance of the questions, their wording, the coverage of their answers, etc. They revised the questions and suggested minor changes. The final survey contained six sections: (1) participants' professional and demographic data, (2) recommended best practices of REST APIs usage in Android mobile apps, (3) REST APIs usage in Android Apps, (4) evaluation of third-party HTTP libraries for Android apps, (5) recommended HTTP libraries for each practice, and (6) availability for an interview. All the questions of the survey are available in Appendix A and online[16].

B - Selection of participants. We targeted developers who have an experience in Android development. We relied on (1) Google Play store and GitHub to collect emails of Android developers, (2) information about companies that offer Android mobile development services, (3) Slack channels on Android development, and (4) search queries on LinkedIn profiles, such as "*Android developer OR Android Architect OR Mobile developer*". Once we identified potential participants, we sent them invitations via e-mail, LinkedIn, Facebook, and Twitter.

[15] https://forms.gle/gQhBeFdNQXERf5wc8

[16] http://www.ptidej.net/downloads/replications/ause20/

We chose *not* to solicit more than three professionals from any given company (1) to have a wide representation of Android developers, and (2) to not overburden a single organization with our request.

C - Online Survey. We invited 7,543 professionals to answer our survey and asked them to forward our invitations to other Android developers in their network. The survey was completed by 122 people, out of whom four did not use REST APIs. Consequently, we discarded their responses, leaving us with 118 complete responses.

D - Validation. In the survey, we added several related questions to detect any contradiction among answers. For example, we asked the developers if they use a given practice in their Android apps. Then, we asked them to mention the percentage of their Android apps that use this particular practice. Also, we asked them to mention the reasons for ignoring this particular practice. We assessed the reliability of the answers in the online survey by looking for spurious/facetious answers, as well as any contradictions between the answers, etc.

5.2 Survey Results

We now report the results of our survey. We allowed multiple answers to most questions as well as open answers. Therefore, the sum of the computed percentage may exceed 100% for some questions. Also, we computed the given percentages based on the total numbers of participants who answered the question; these numbers vary from questions to questions. We present in the following subsections the results of each section of the survey as well as the threats to validity of this study.

5.2.1 Participants

We reached a total of 118 participants involved in Android development projects. As shown in Figure 16, 78% were software developers and 9% were software architects. In terms of experience, Figure 17 shows that 45% had more than 5 years of mobile development experience while 49% had between 1 to 4 years. We also summarises the sizes of the Android apps developed by the participant in the box-plot of Figure 4. As shown in Figure 18, 55% of the developed Android apps were medium-sized (816 KLOCs) while 27% of the apps were large-sized (>16 KLOCs).

5.2.2 Prevalence of REST APIs Usage

We asked the participants about the numbers of their apps consuming REST APIs. The majority of the participants (96%) reported that they used REST APIs in their apps but in different proportions as shown in Figure 20: 42% mentioned that all their Android apps were REST clients while 4% mentioned that none used REST APIs.

We also asked the participants what other APIs they use for their Android apps. As shown in Figure 19, we found that developers are using some other

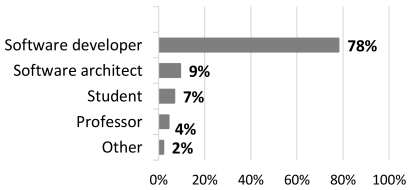

Fig. 16. Professions of the participants

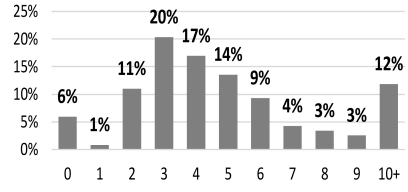

Fig. 17. Number of years related to mobile development experience

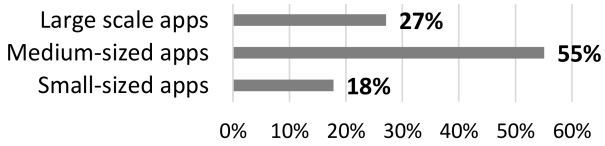

Fig. 18. Size of the developed Android apps

APIs, such as SOAP, RPC, and Web sockets. We also found that there is an increasing interest in using GraphQL (15%), which is an API query language that could be an alternative to REST for developing APIs. However, REST APIs are still widely used by all developers.

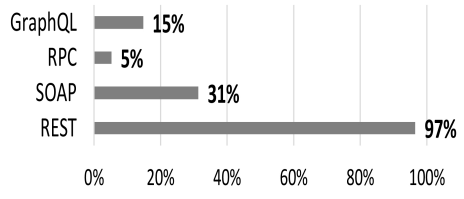

Fig. 19. Distribution of the used API protocols/paradigms

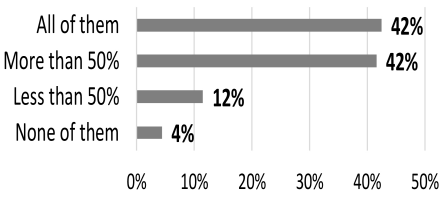

Fig. 20. Distribution of Android REST clients

5.2.3 Developers' Classification of the Practices

We asked the participants to provide their classification of the studied practices of REST APIs usage into "good", "bad", or "neutral" practices. Figure 21 shows the practices that are classified as "good": the use of third-party HTTP libraries, network-connectivity awareness, JSON response parsing, setting timeouts for HTTP requests, and the specification of a behavior upon failure. Regarding caching responses, Android developers considered this practice either "good" or

"neutral" because it depends on its context of usage. Parsing REST responses with XML is considered by most developers as a "bad" or "neutral" practice as they highly recommended parsing responses with JSON. Finally, making synchronous API calls is not considered a "good" practice by Android developers: 72% classified it as "neutral" or "bad".

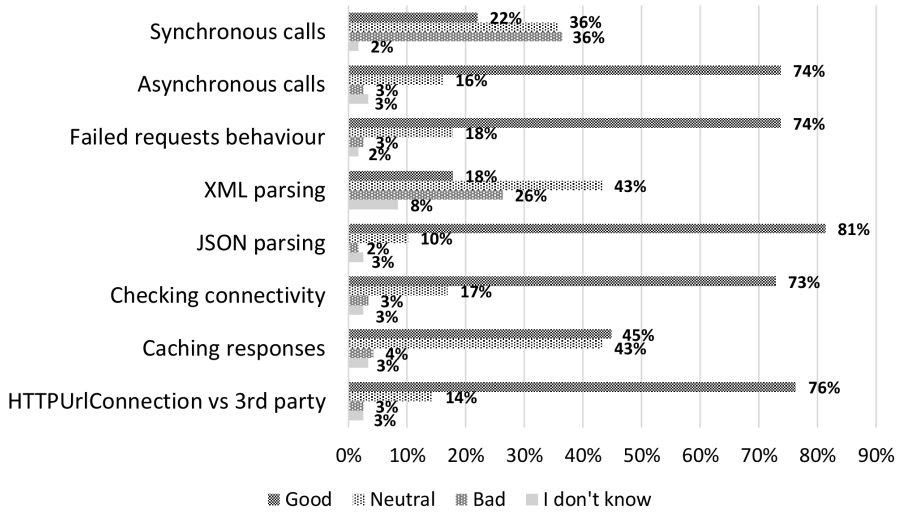

Fig. 21. The classification of the practices by the developers

5.2.4 Usage of the Recommended Practices

We now report the usage of the recommended practices by the Android developers, as summarised in Figure 22. We also report on the numbers of apps that use each practice and the developers' reasons for ignoring some of them.

HttpURLConnection vs. Third-party Libraries for HTTP Requests. We notice that Android developers prefer using third-party HTTP libraries rather than HttpURLConnection. Figure 22 shows that only 33% of the developers mentioned the use of HttpURLConnection vs. 81% who prefer using third-party libraries (especially Retrofit) but in different proportions as shown by Figure 23.

We asked the developers what are the reasons of using HttpURLConnection to manage REST APIs queries rather than third-party HTTP libraries. Figure 24 shows that 31% of the developers prefer to use HttpURLConnection when they must customize specific configurations of REST requests and 27% because of the company internal policies that oblige them to avoid dependencies on third-party HTTP libraries. The participants mentioned other reasons for considering

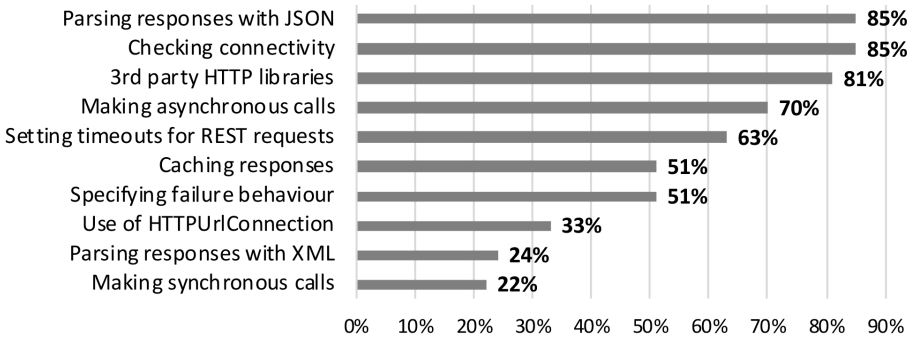

Fig. 22. Distribution of the usage of the practices

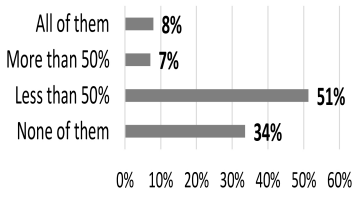

Fig. 23. Distribution of the apps using HttpURLConnection

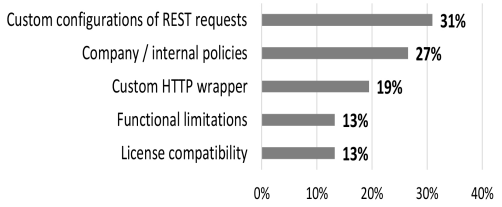

Fig. 24. Reasons of ignoring using HttpURLConnection

HttpURLConnection, such as the use of custom HTTP wrappers (20%), the functional limitations of third-party HTTP libraries (13%), and licensing issues (13%).

Cached vs. Non-cached Responses. As depicted in Figure 22, caching REST API responses is not a widely used practice: only 51% of the developers considered caching in their apps. Also, Figure 25 shows that only 14% of the developers use caching for REST API responses in all their apps.

We asked the developers the reasons for ignoring caching REST APIs responses and report the results in Figure 26. The main reasons were that caching responses was irrelevant in the context of their apps (38%), especially when dealing with dynamic data that change frequently or when the mobile app do not perform many REST APIs requests. Another reason for ignoring caching is security, such as the management of sensitive data, which should not be cached (29%). Also, 19.5% of the developers reported that implementing cache to be complex and, thus ignored this practice. Other reasons include performance issues (20.4%) and deadlines (15.9%).

Network-connectivity Aware vs. Unaware REST requests. Network connectivity checking is a popular practice among developers as illustrated by Fig-

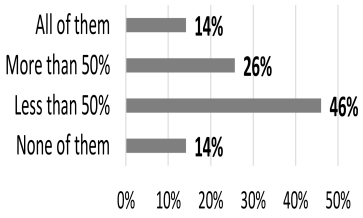

Fig. 25. Distribution of the apps using cache

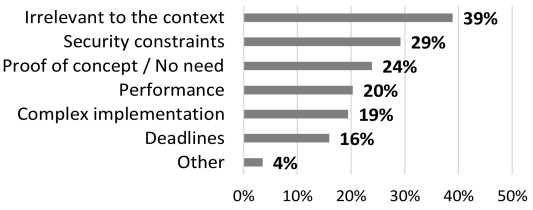

Fig. 26. Reasons of ignoring caching responses

ures 22 and 27. 52% of the developers declared that they check network connectivity in 100% of their apps; 21% implemented this practice for more than 50% of their apps; 16.8% for less than 50% of their apps; and, only 10% never check network connectivity.

We asked the participants when they check the network connectivity of their Android apps to REST APIs. We report the results in Figure 28: most of the developers (72%) check network connectivity before each request while a few of them (9%) only on the initialisation of their apps.

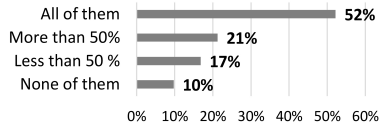

Fig. 27. Distribution of the apps that check network connectivity

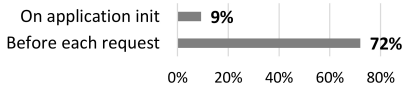

Fig. 28. When do you check network connectivity?

JSON vs. XML Response Parsing. Parsing REST APIs responses with JSON rather than XML is a widely used by Android developers. Figure 22 shows that 86% of the developers mentioned the use of JSON parsers for REST APIs responses while 24% mentioned the use of XML. We also asked which parsers they use, as shown in Figure 29, and found that most (89%) rely on Google Gson. Some use the Jackson parser (31%) and some others native parsers (32%), such as JsonParser, XmlPullParser, etc.

Timeout vs. Perpetual REST Requests. Figure 22 shows that 63% of the developers mentioned the use of timeouts for REST requests in their Android apps. However, Figure 30 shows that only 44% mentioned its use in all their Android apps. We asked the developers the reasons for ignoring this recommended practice and we found that it was mainly due to their use of third-party libraries

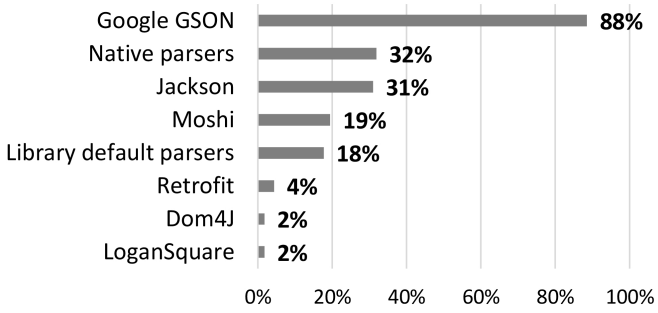

Fig. 29. Distribution of the used parsers for REST APIs

that offer default values for timeouts. Another reason is that timeouts do not matter in the context of their apps (13%) or for performance issues (14%).

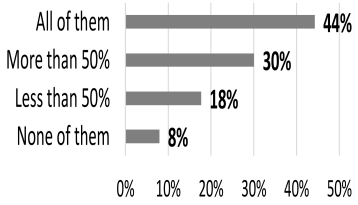

Fig. 30. Distribution of the apps that consider timeouts

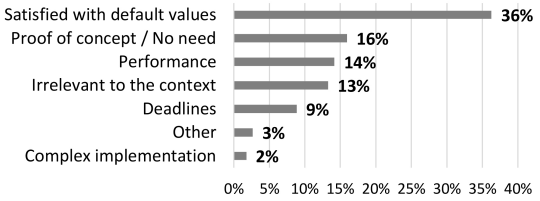

Fig. 31. Reasons for ignoring timeouts

Specification vs. Non-specification of a Behaviour Upon Failure. We found that the specification of a behaviour when REST APIs requests fail is not a widely spread practice among developers, with only 51% mentioning its use in their Android apps, as shows by Figure 22. Only 10% mentioned using this practice in all their apps while 67% in some or none of their apps as shown in Figure 32.

We found that Android developers tend to ignore such practice because they are satisfied with the default values, especially when using third-party libraries (21%). Also, they tend to ignore this practice because of deadlines pushing them to finish other requirements. They also mentioned other reasons for ignoring error handling, such as the context of their apps (15%) and the complexity of the implementation of error handling (11%).

Synchronous vs. Asynchronous REST Requests. Figure 22 shows that developers highly rely on asynchronous calls: 70% use asynchronous calls in their

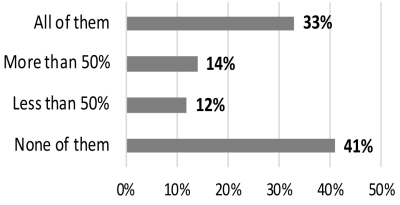

Fig. 32. Distribution of the apps that consider error handling of REST API handling requests

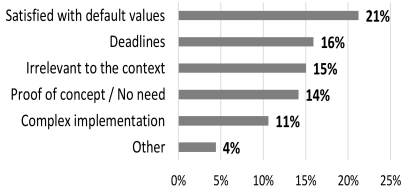

Fig. 33. Reasons for ignoring error handling requests

apps compared to only 22% using synchronous calls. Figure 34 also shows that 37% of the developers never use synchronous calls while only 11% use them for all their apps. In comparison, Figure 35 shows that 8% of the developers never use asynchronous calls while 44% use asynchronous calls for all their apps.

Developers mentioned many reasons for ignoring synchronous calls. Figure 36 shows that performance issues, such as blocking UI thread, is the main reason for using asynchronous calls (46%), followed by the context of their apps (20%). Some mentioned that they did not rely on synchronous calls because their apps were proofs of concept (17%) or because of the complexity of the implementation of such calls (13%).

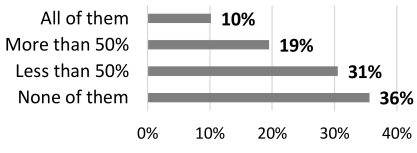

Fig. 34. Distribution of the apps using synchronous REST APIs requests

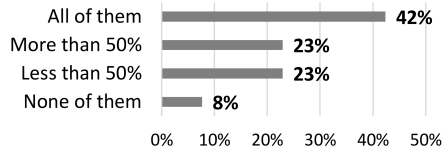

Fig. 35. Distribution of the apps using asynchronous REST APIs requests

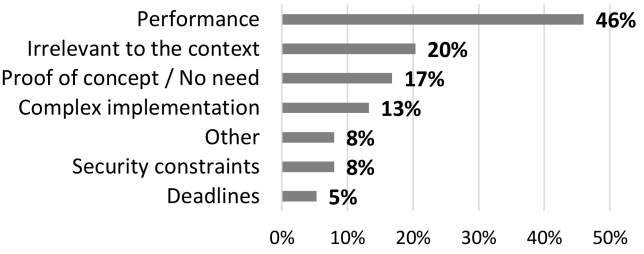

Fig. 36. Reasons of ignoring synchronous REST APIs requests

5.2.5 Evaluation of Third-party HTTP Libraries

We report on the evaluation of third-party HTTP libraries from the Android developers' point of view. We study their prevalence, the elements that the developers consider for choosing HTTP libraries, and their usefulness in the adoption/application of the recommended practices of REST APIs usage.

Prevalence of Third-party HTTP Libraries. We asked the developers the third-party HTTP libraries that they use to manage REST APIs requests in their Android apps. Figure 37 shows that they rely mainly on Retrofit, followed by OkHttp and Google Volley. Also, we found that Android developers rarely use other third-party HTTP libraries as 69% mentioned not using other libraries than Retrofit, OkHttp, and Google Volley. However, a few mentioned LoopJ, RxJava, RxAndroid, and Cronet.

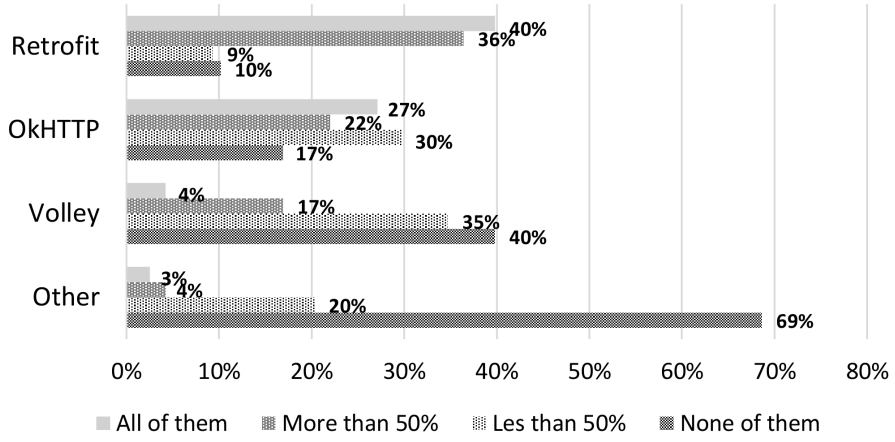

Fig. 37. Distribution of the used third-party HTTP libraries

Choice of Third-party HTTP Libraries. We asked the developers what are the elements that they consider for choosing third-party HTTP libraries. Figure 38 shows that 81% of the developers focus on performance (e.g., reliability when dealing with heavy resources, execution time, etc.). They also consider the ease of use (75%) as well as their flexibility to fulfill their apps requirements (71%). Developers consider quality of the documentation (58%) as well as debugging and tracing tools (43%). Developers consider less how active is the community of the libraries (39%) or battery usage (27%) because third-party HTTP libraries

may not have a major impact on energy consumption even through HTTP calls may impact battery life[17].

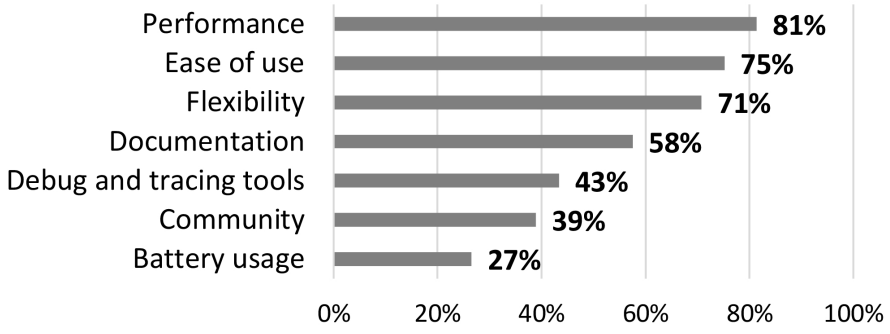

Fig. 38. Elements considered for choosing third-party libraries

Usefulness of Third-party HTTP Libraries for the Recommended Practices. We asked the developers if third-party HTTP libraries help them adopt the recommended practices of REST APIs usage in Android apps. Figure 39 shows that almost all developers (96%) think that such libraries are very helpful to apply the recommended practices because they provide features that are easy to use in comparison to HttpURLConnection.

We also asked their recommendations of the libraries to use for each practice. Figure 39 shows that they highly recommend Retrofit for all the practices, followed by OkHttp and Google Volley. Only a few developers recommend the use of HttpURLConnection for the targeted practices.

5.3 Threats to Validity

Construct Validity. These threats refer to the extent to which the operationalizations of a construct (in our case the survey questions and terminology) do actually measure what the theory claims. To minimize this threat, we used both open and closed questions in the survey and tried to minimize the ambiguities through our pilot study as we mentioned in Section 5.1.

Internal Validity. Acquiescence bias is a kind of response bias because of which survey respondents tend to agree with the questions in a survey or to choose "positive" answers. It is also the tendency to agree with a statement when in doubt. We mitigate this threat by (1) adding several related questions

[17]https://developer.android.com/topic/performance/power/network/action-any-traffic.html

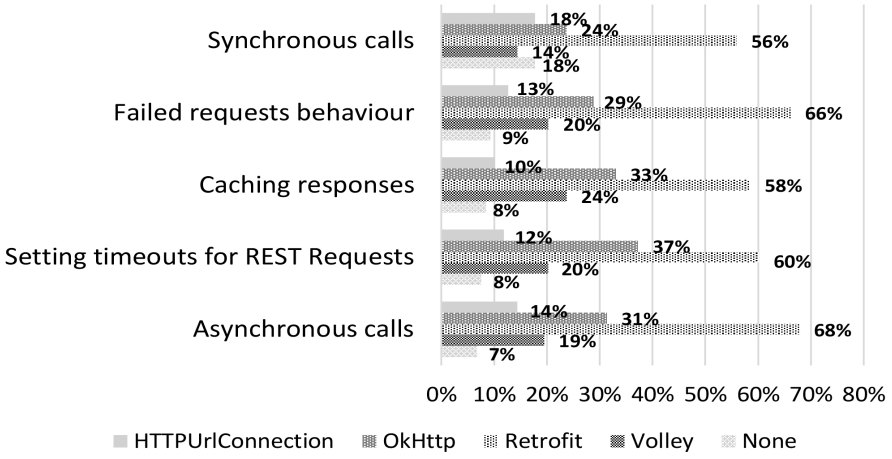

Fig. 39. Recommended HTTP libraries by practice

(e.g., the use of the practices and the reasons for ignoring them), (2) checking and validating the consistency among the responses to questions related to one another, (3) eliminating contradictory responses, and (4) eliminating responses in which participants selected all the possible choices. Finally, we decided not to have incentives for participating in the survey to minimize the response bias.

External Validity. These threats relate to the generalisability of our survey. The participants might not be representative of the general population of Android developers as the response rate (122/7,543) is low but expected. To mitigate this threat, we advertised our survey through various channels (e.g., LinkedIn and Facebook) and targeted Android developers from different mobile development companies. The participants could freely decide whether to participate in the study or not (self-selection). They were informed as well about the topic of the survey, the estimated time to complete it, its purpose, and the guaranteed anonymity of their (non)participation and answers. Also, to the best of our knowledge, it is still the largest number of participants to date in any such study. Finally, while the number of participants cannot ensure the generalisation of our findings, we believe that the answers provide a valuable source of information to understand what developers think about the practices.

6 Answers to Our Research Questions

We now answer our research questions using our previous results.

6.1 RQ1: What is the state of the practice in the use of REST APIs by Android apps?

Based on our analysis of Android apps and our survey, we identified several practices of REST APIs usage in Android clients that we classified into good, neutral, or bad.

Good Practices. We observed that only two practices are widely considered "good" by Android developers when implementing their mobile apps: network connectivity awareness and JSON response parsing. Connectivity-aware REST requests and JSON responses parsing are the most evident/trivial practices for Android developers as reflected by the low number of questions about these practices on Stack Overflow vs. their high usage in Android apps.

Bad Practices. We found that Android developers ignore some good practices of REST APIs usage: the use of third-party HTTP libraries, caching responses, setting timeouts, and error handling. Caching REST responses is the most problematic practice for Android developers as reflected by the high number of questions about caching REST requests vs. the low percentage of Android apps using a cache (as shown in Figures 7 and 12). Specifying a behavior when requests fail is also difficult for Android developers as they ask many questions about it on Stack Overflow.

Neutral. Asynchronous REST requests seem to be difficult to implement for developers as reflected by the high number of questions on this topic vs. the high number of apps relying on asynchronous calls. We found some discrepancies between the results of our observational study of Android REST clients and the results of the survey for synchronous REST requests. Based on our observational study of Android REST clients, we found an equal distribution between synchronous and asynchronous REST APIs calls in Android apps. However, in our survey, we found that Android developers do not prefer to rely on synchronous REST APIs calls. It is even considered as a neutral or a bad practice. Finally, developers highly recommend avoiding synchronous calls to improve apps performance.

Also, we gathered that developers ignore some of the recommended practices with good reasons, because they do not apply in their context, for example caching and timeouts.

6.2 RQ2: What is the state of the implementation of HTTP libraries in Android apps?

Choosing an efficient mobile HTTP library to communicate with a REST API can be difficult for developers because they must handle many aspects, such as making connections, caching, retrying failed requests, parsing responses, handling errors, etc.

Although only 28% of the Android apps use third-party HTTP libraries, since the apparition of such libraries, developers have been using them increasingly. Indeed, we identified that almost 98% of apps released in 2018 are using third-party libraries. Also, based on our survey, we found that Android developers highly use third-party libraries to manage REST APIs requests (81%).

After analyzing the state of the implementation of HTTP libraries in Android apps, the distribution of questions about these libraries in Stack Overflow, and the results of our survey, we found that Retrofit is the most used and discussed third-party HTTP library because of its ease of use, high performance, and its caching features.

Finally, we found that the choice of a third-party HTTP library to use depends basically on the performance of the library, its ease of use as well as its flexibility to fulfill the applications requirements.

6.3 RQ3: Do third-party HTTP libraries help developers adopt the recommended practices of REST APIs usage in Android apps?

Based on our study, we found that since the apparition of third-party libraries, Android apps tend to use more and more good practices when performing REST requests. Additionally, almost all Android developers (98%) claimed in our survey that third-party HTTP libraries help them adopt/apply "good" practices of REST APIs usage in their apps.

We observe that the non-use of third-party HTTP libraries hinders the adoption of good practices, which is the case for timeouts and error handling that are still highly implemented with HttpURLConnection, as shown in Figures 10 and 11). Finally, Android developers recommend the use of third-party libraries to apply the recommended practices of REST APIs usage in Android apps.

Thus, we answer that third-party HTTP libraries help developers in their implementations of good practices of REST APIs usage in Android mobile apps. However, Android developers must also carefully choose the right third-party library that copes with their needs (e.g., performance, flexibility, ease of use, etc.). Based on our results, we recommend using Retrofit.

7 Recommendations

Developing REST clients for mobile apps is a more challenging task for developers than "traditional" software application development because they must work with limited resources, such as the processing power, battery, and connectivity. We aim through this study to motivate/urge the developers to consider good practices of implementing Android REST clients to ensure the quality of the developed apps and, thus, make the best of the limited available resources.

Before performing a REST request, we recommend that developers check the Internet connectivity (1) to offload heavy REST queries when the device is connected to WiFi, (2) to increase device battery life, and (3) to detect network

changes and resume incomplete REST requests if necessary. Then, we recommend that developers choose the "right" third-party HTTP libraries, mainly Retrofit, to manage REST requests on Android apps. Such library speeds up development by reducing coding efforts and improves quality by promoting the implementation of good practices of REST APIs usage.

Although the choice between synchronous and asynchronous requests to REST APIs depends on the context of the apps, developers must carefully choose between the two ways of communication. They must avoid, when possible, synchronous requests to increase the responsiveness of their apps. When implementing a REST request, we also recommend setting carefully timeouts to increase the app responsiveness: developers must adapt default timeout values to make sure they fit with the app responsiveness requirements. We also recommend to parse REST APIs responses with JSON rather than XML because JSON format is faster to parse than XML [12]. We also recommend to cache frequent REST requests to reduce bandwidth usage, network latency, and battery consumption. When a REST request fails, we suggest that developers handle the errors to increase the reliability of the app. Although Android developers rarely update their libraries [21], we highly recommend updating their HTTP libraries to support/implement more good practices of REST APIs usage in their apps.

Finally, we recommend that providers of HTTP libraries focus on performance, flexibility, and ease of use of their libraries because these are the main criteria of the developers to choose an HTTP library for Android apps according to our survey analysis.

8 Conclusion and Future Work

Several research works studied REST APIs development practices for mobile apps. However, little was known on how Android mobile apps use/consume REST APIs. We described in this paper a multi-dimensional study about the state of the practices of REST APIs usage in Android apps, which is an extension to our previous observational study [6] for which we additionally conducted an online survey to assess the importance of the identified practices from Android developers' point of view.

We thus provided five contributions: (1) we reviewed the literature extensively and compiled a catalog of seven practices related to the development of apps using REST APIs. These practices pertained to the use of dedicated, third-party libraries and helped understanding how Android apps use/consume REST APIs, (2) we proposed a framework, PIRAC, to detect automatically occurrences of all the identified practices using detection rules, (3) we conducted an observational study of 1,595 REST mobile apps out of 9,173 Android apps downloaded from the Google Play store to report how they use/consume REST APIs, (4) we mined 12,478 Stack Overflow posts to assess the importance of the identified practices and the use of Android HTTP libraries from the developers' point of view, and (5) we conducted an online survey with 118 Android developers to validate our previous observations.

Based on our study, we found that developers tend to ignore some good practices of REST APIs usage in Android apps. These practices are caching responses, timeout management, and error handling. We found that only two good practices are widely considered by Android developers when implementing their mobile apps: network connectivity awareness and JSON vs. XML responses parsing. We also showed that third-party HTTP libraries promote the use of recommended practices of REST APIs usage in Android apps.

We concluded that Android developers should consider more good practices when implementing mobile REST clients, especially because HTTP requests are the most energy consuming network operation in apps [22,23]. Thus, efficient and high quality REST requests should be a developers' primary focus when considering the use of network connection on mobile devices. Mobile developers should also apply these practices to enhance the quality, responsiveness, and efficiency of their apps. They should consider as well existing HTTP libraries to benefit from the good practices implemented/provided by these libraries, such as asynchronous requests, timeout management, caching responses, and error handling. Finally, we also concluded that service providers must strive to make their libraries as simple as possible. They should focus on their performance, flexibility, and ease of use as these are the main criteria for developers to choose a library.

Our work is beneficial for both practitioners and the research community. It provides the first overview of the state of the practices of REST APIs usage in Android apps. Future work includes selecting and–or defining more practices and analysing their prevalence in Android apps as well as other apps for other mobile operating systems, in particular iOS. Thus, we could further recommend to developers best practices when developing apps that use REST APIs on different OSes. We also want to empirically and quantitatively study the impact of ignoring the recommended practices on the mobile apps in terms of quality and energy consumption. Finally, we want to study source-code transformations that would allow developers to migrate their apps from one library, in particular HttpURLConnection, to other libraries, like Retrofit or Google Volley, to benefit from their features and good practices.

References

1. Carlos Rodríguez, Marcos Báez, Florian Daniel, Fabio Casati, Juan Carlos Trabucco, Luigi Canali, and Gianraffaele Percannella. REST apis: A large-scale analysis of compliance with principles and best practices. In *16th International Conference Web Engineering*, volume 9671 of *Lecture Notes in Computer Science*, pages 21–39. Springer, 2016.
2. Fábio Petrillo, Philippe Merle, Naouel Moha, and Yann-Gaël Guéhéneuc. Are REST apis for cloud computing well-designed? an exploratory study. In *14th International Conference on Service-Oriented Computing*, volume 9936 of *Lecture Notes in Computer Science*, pages 157–170. Springer, 2016.
3. Hayet Brabra, Achraf Mtibaa, Layth Sliman, Walid Gaaloul, Boualem Benatallah, and Faiez Gargouri. Detecting cloud (anti) patterns: Occi perspective. In *In-*

ternational Conference on Service-Oriented Computing, pages 202–218. Springer, 2016.
4. Francis Palma, Johann Dubois, Naouel Moha, and Yann-Gaël Guéhéneuc. Detection of REST patterns and antipatterns: A heuristics-based approach. In *12th International Conference on Service-Oriented Computing*, volume 8831 of *Lecture Notes in Computer Science*, pages 230–244. Springer, 2014.
5. Mohamed A Oumaziz, Abdelkarim Belkhir, Tristan Vacher, Eric Beaudry, Xavier Blanc, Jean-Rémy Falleri, and Naouel Moha. Empirical study on rest apis usage in android mobile applications. In *International Conference on Service-Oriented Computing*, pages 614–622. Springer, 2017.
6. Abdelkarim Belkhir, Manel Abdellatif, Rafik Tighilt, Naouel Moha, Yann-Gaël Guéhéneuc, and Éric Beaudry. An observational study on the state of rest api uses in android mobile applications. In *Proceedings of the 6th International Conference on Mobile Software Engineering and Systems*, pages 66–75. IEEE Press, 2019.
7. Francis Palma, Naouel Moha, and Yann-Gaël Guéhéneuc. Unidosa: The unified specification and detection of service antipatterns. *IEEE Transactions on Software Engineering*, 2018.
8. Francis Palma, Javier Gonzalez-Huerta, Naouel Moha, Yann-Gaël Guéhéneuc, and Guy Tremblay. Are restful apis well-designed? detection of their linguistic (anti)patterns. In *13th International Conference on Service-Oriented Computing*, volume 9435 of *Lecture Notes in Computer Science*, pages 171–187. Springer, 2015.
9. Hanzhang Wang, Marouane Kessentini, and Ali Ouni. Bi-level identification of web service defects. In *14th International Conference on Service-Oriented Computing*, volume 9936 of *Lecture Notes in Computer Science*, pages 352–368. Springer, 2016.
10. Hayet Brabra, Achraf Mtibaa, Layth Sliman, Walid Gaaloul, Boualem Benatallah, and Faïez Gargouri. Detecting cloud (anti)patterns: OCCI perspective. In *14th International Conference on Service-Oriented Computing*, volume 9936 of *Lecture Notes in Computer Science*, pages 202–218. Springer, 2016.
11. Nurzhan Nurseitov, Michael Paulson, Randall Reynolds, and Clemente Izurieta. Comparison of JSON and XML data interchange formats: A case study. In *22nd International Conference on Computer Applications in Industry and Engineering*, pages 157–162, 2009.
12. Carlos Rodrigues, José Afonso, and Paulo Tomé. Mobile application webservice performance analysis: Restful services with json and xml. In *International Conference on ENTERprise Information Systems*, pages 162–169. Springer, 2011.
13. Raja Vallée-Rai, Phong Co, Etienne Gagnon, Laurie Hendren, Patrick Lam, and Vijay Sundaresan. Soot: A java bytecode optimization framework. In *CASCON First Decade High Impact Papers*, pages 214–224. IBM Corp., 2010.
14. Li Li, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. An investigation into the use of common libraries in android apps. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 1, pages 403–414. IEEE, 2016.
15. Ravindra Singh and Naurang Singh Mangat. *Elements of survey sampling*, volume 15. Springer Science & Business Media, 2013.
16. Stefan Tilkov. Rest anti-patterns. *InfoQ Article (July 2008)*, 2008.
17. Mauricio Arroqui, Cristian Mateos, Claudio Machado, and Alejandro Zunino. Restful web services improve the efficiency of data transfer of a whole-farm simulator accessed by android smartphones. *Computers and Electronics in agriculture*, 87:14–18, 2012.
18. Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46, 1960.

19. J Richard Landis and Gary G Koch. The measurement of observer agreement for categorical data. *biometrics*, pages 159–174, 1977.
20. Manel Abdellatif, Geoffrey Hecht, Hafedh Mili, Ghizlane Elboussaidi, Naouel Moha, Anas Shatnawi, Jean Privat, and Yann-Gaël Guéhéneuc. State of the practice in service identification for soa migration in industry. In *International Conference on Service-Oriented Computing*, pages 634–650. Springer, 2018.
21. Pasquale Salza, Fabio Palomba, Dario Di Nucci, Andrea De Lucia, and Filomena Ferrucci. Third-party libraries in mobile apps. *Empirical Software Engineering*, pages 1–37, 2019.
22. Ding Li, Yingjun Lyu, Jiaping Gui, and William GJ Halfond. Automated energy optimization of http requests for mobile applications. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 249–260. IEEE, 2016.
23. Ding Li and William GJ Halfond. Optimizing energy of http requests in android applications. In *Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile*, pages 25–28, 2015.

A Appendix 1: Overview of the Survey Questions

ID	Question	Possible answers
Demographic info		
1.1	What describes you the best ?	Software architect, Director of technology, Software engineer, Software developer, Professor, Student, Other (Open answer)
1.2	How many years of experience in mobile development do you have ?	Scale from 0 to 10+
1.3	How many android applications have you developed ?	Less than 5, 5 to 10, 10 to 25, More than 25
1.4	The applications that you have developed are more often	Small (<8 kLOCs), Medium (8 - 16 kLOCS), Large (>16 kLOCS)
1.5	In all Android apps that you developed select all the API protocols that you used	REST, SOAP, RPC, GraphQL, Other (Open answer)
1.6	Have you ever developed an Android app that uses REST APIs?	Yes, No
Recommended Best Practices		
2.1	Among the Android apps that you developed, how many are consuming REST APIs (REST clients)?	None of them (0%), Some of them (<50%), Most of them (>50%), All of them (100%)
2.2	If possible, can you provide links to some Android applications using REST APIs that you developed?	Open question
2.3	Regarding the Android REST clients that you developed, which of these practices did you consider ?	Use of third party HTTP client libraries, Use of HTTPUrlConnection, Parsing REST APIs responses with JSON, Parsing REST APIs responses with XML, Caching responses, Setting timeouts for REST requests, Specifying a behavior when REST requests fail, Making asynchronous HTTP calls, Making synchronous HTTP calls, Other (Open answer)
2.4.1	How do you classify the Use of 3rd party HTTP client libraries vs HTTPUrlconnection to manage HTTP requests when using REST APIs in Android Apps?	Good, Neutral, Bad, I don't know
2.4.2	How do you classify caching responses when using REST APIs in Android Apps?	Good, Neutral, Bad, I don't know

2.4.3	How do you classify Check for netwotk connectivity before performing HTTP requests when using REST APIs in Android Apps?	Good, Neutral, Bad, I don't know
2.4.4	How do you classify Parsing REST APIs responses with JSON when using REST APIs in Android Apps?	Good, Neutral, Bad, I don't know
2.4.5	How do you classify Parsing REST APIs responses with XML when using REST APIs in Android Apps?	Good, Neutral, Bad, I don't know
2.4.6	How do you classify Specifying a behavior when REST requests fail when using REST APIs in Android Apps?	Good, Neutral, Bad, I don't know
2.4.7	How do you classify Making HTTP call asynchronously when using REST APIs in Android Apps?	Good, Neutral, Bad, I don't know
2.4.8	How do you classify Making HTTP call synchronously when using REST APIs in Android Apps?	Good, Neutral, Bad, I don't know
2.5	What other good practices do you recommend for using REST APIs in Android applications?	Open question
REST APIs usage in Android Apps		
3.1	Among the Android apps that you developed, how many check the network connectivity for REST requests ?	None of them (0%), Some of them (<50%), Most of them (>50%), All of them (100%)
3.2	When do you check the network connectivity for REST requests?	On application initialisation, Before each request
3.3.1	Among the Android apps that you developed, how many use JSON to parse HTTP responses?	None of them (0%), Some of them (<50%), Most of them (>50%), All of them (100%)
3.3.2	Among the Android apps that you developed, how many use XML to parse HTTP responses?	None of them (0%), Some of them (<50%), Most of them (>50%), All of them (100%)
3.4	What libraries do you use to parse HTTP responses ?	Native parsers, Library defaulkt parser, Google Gson, Moshi, LoganSquare, XmlPullParser, Jackson, Dom4J, Other (Open answer)
3.5	Among the Android apps that you developed, how many apps consider caching responses for REST APIs requests?	None of them (0%), Some of them (<50%), Most of them (>50%), All of them (100%)

3.6	Regarding the Android REST clients you developed, what are the reasons for ignoring caching responses for REST APIs requests?	Irrelevant to the context, Complex implementation, Deadlines, Proof of concept / No need, Performace, Security constraints, I don't ignore this practice, Other (Open answer)
3.7.1	Among the Android apps that you developed, how apps perform synchronous calls to REST APIs?	None of them (0%), Some of them (<50%), Most of them (>50%), All of them (100%)
3.7.2	Among the Android apps that you developed, how apps perform asynchronous calls to REST APIs?	None of them (0%), Some of them (<50%), Most of them (>50%), All of them (100%)
3.8	Regarding the Android REST clients you developed, what are the reasons for ignoring asynchronous calls for REST APIs requests?	Irrelevant to the context, Complex implementation, Deadlines, Proof of concept / No need, Performace, Security constraints, I don't ignore this practice, Other (Open answer)
3.9	Regarding the Android REST clients you developed, what are the reasons for ignoring synchronous calls for REST APIs requests?	Irrelevant to the context, Complex implementation, Deadlines, Proof of concept / No need, Performace, Security constraints, I don't ignore this practice, Other (Open answer)
3.10	Among the Android apps that you developed, how many apps consider setting timeouts for REST requests?	None of them (0%), Some of them (<50%), Most of them (>50%), All of them (100%)
3.11	Among the Android apps that you developed, what are the reasons for ignoring setting timeouts for REST requests?	Satisfied with default values, Irrelevant to the context, Complex implementation, Deadlines, Proof of concept / No need, Performace, I don't ignore this practice, Other (Open answer)
3.12	Among the Android apps that you developed, how many apps specify a behaviour when REST requests fail?	None of them (0%), Some of them (<50%), Most of them (>50%), All of them (100%)
3.13	Among the Android apps that you developed, what are the reasons for ignoring the specification of a behaviour when REST requests fail?	Satisfied with default values, Irrelevant to the context, Complex implementation, Deadlines, Proof of concept / No need, Performace, I don't ignore this practice, Other (Open answer)
3.14	Among the Android apps that you developed, how many apps use HttpUrlConnection to manage HTTP queries?	None of them (0%), Some of them (<50%), Most of them (>50%), All of them (100%)
3.15	Among the Android apps that you developed, what are the reasons for using HttpUrlConnection rather than third-party HTTP client libraries (Retrofit, Okhttp, Volley, etc.)?	License compatibility, Custom configuration of REST requests, Company / internal policies, Functionnal limitations, Custom HTTP Wrapper, I don't ignore this practice, Other (Open answer)
3.16	If you metionned "Functionnal limitations" above, please give an example	Open answer

Evaluation of 3rd party HTTP library for Android apps		
4.1.1	Among the Android apps that you developed, how many use Volley to manage HTTP queries?	None of them (0%), Some of them (<50%), Most of them (>50%), All of them (100%)
4.1.2	Among the Android apps that you developed, how many use OkHTTP to manage HTTP queries?	None of them (0%), Some of them (<50%), Most of them (>50%), All of them (100%)
4.1.3	Among the Android apps that you developed, how many use Retrofit to manage HTTP queries?	None of them (0%), Some of them (<50%), Most of them (>50%), All of them (100%)
4.1.4	Among the Android apps that you developed, how many use another third-party client library to manage HTTP queries?	None of them (0%), Some of them (<50%), Most of them (>50%), All of them (100%)
4.2	What other third-party HTTP client libraries do you use for Android apps?	Open answer
4.3	What elements do you consider for choosing the third-party HTTP client library for your Android app?	Flexibility, Performance, Documentation, Ease of use, Community, Battery usage, Debugging and tracing tools, Other (Open answer)
4.4.1	How do you evaluate the documentation of Volley client library?	Good, Average, Poor, I don't know
4.4.2	How do you evaluate the documentation of OkHTTP client library?	Good, Average, Poor, I don't know
4.4.3	How do you evaluate the documentation of Retrofit client library?	Good, Average, Poor, I don't know
4.5.1	In the context of REST API requests, what library(ies) you recommend for caching responses	HTTPUrlConnection, OkHTTP, Retrofit, Volley, None
4.5.2	In the context of REST API requests, what library(ies) you recommend for setting timeouts	HTTPUrlConnection, OkHTTP, Retrofit, Volley, None
4.5.3	In the context of REST API requests, what library(ies) you recommend for specifying behavior at failed requests	HTTPUrlConnection, OkHTTP, Retrofit, Volley, None
4.5.4	In the context of REST API requests, what library(ies) you recommend for making synchronous HTTP calls	HTTPUrlConnection, OkHTTP, Retrofit, Volley, None
4.5.5	In the context of REST API requests, what library(ies) you recommend for making asynchronous HTTP calls	HTTPUrlConnection, OkHTTP, Retrofit, Volley, None
4.6	If you answered none, please explain why	Open answer

4.7	Do you think that using 3rd party HTTP libraries helps you in adopting the recommended practices of REST APIs usage in Android apps?	Yes, No
4.8	If you answered "No" in the question above, please mention why	Open answer

Table 2: Overview of the survey questions