iGateLink: A Gateway Library for Linking IoT, Edge, Fog, and Cloud Computing Environments

Riccardo Mancini, Shreshth Tuli, Tommaso Cucinotta, and Rajkumar Buyya

Abstract In recent years, the Internet of Things (IoT) has been growing in popularity, along with the increasingly important role played by IoT gateways, mediating the interactions among a plethora of heterogeneous IoT devices and cloud services. In this paper, we present *iGateLink*, an open-source Android library easing the development of Android applications acting as a gateway between IoT devices and edge/fog/cloud computing environments. Thanks to its pluggable design, modules providing connectivity with a number of devices acting as data sources or fog/cloud frameworks can be easily reused for different applications. Using *iGateLink* in two case studies replicating previous works in the healthcare and image processing domains, the library proved to be effective in adapting to different scenarios and speeding up development of gateway applications, as compared to the use of conventional methods.

Keywords Internet of Things · Gateway applications · Edge computing · Fog computing · Cloud computing

1 Introduction

Recently, the Internet of Things (IoT) has gained significant popularity among both industry and academia, constituting a fundamental technology for creating novel computing environments like smart cities and smart healthcare applications, which pose higher requirements on the capabilities of modern computing infrastructures [1, 2]. Cloud computing allowed offloading complex and heavyweight computations,

R. Mancini · S. Tuli (✉) · R. Buyya
Cloud Computing and Distributed Systems (CLOUDS) Lab, School of Computing and Information System, The University of Melbourne, Melbourne, Australia
e-mail: shreshthtuli@gmail.com

R. Mancini · T. Cucinotta
Scuola Superiore Sant'Anna, Pisa, Italy

S. Tuli
Department of Computer Science and Engineering, Indian Institute of Technology, Delhi, India

D. Mishra et al. (eds.), *Intelligent and Cloud Computing*,
Smart Innovation, Systems and Technologies 194,
https://doi.org/10.1007/978-981-15-5971-6_2

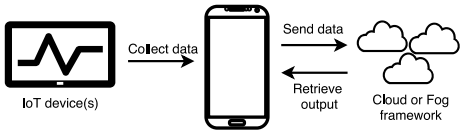

Fig. 1 Example scenario in which one or more IoT devices communicate to the cloud/fog through a gateway device

including big-data processing pipelines, to remote data centers [1]. However, the exponential growth of connected IoT devices and the forecast in the produced data volumes for the upcoming years [3] pushed industry and academia to look into optimized solutions, where virtual machines are dropped in favor of more lightweight containers [4] and, more importantly, *decentralized solutions* are employed, where computing happens at the *edge* of the network, giving rise to the fog computing paradigm. This leads to reduced latency, deployment costs, and improved robustness [5], so in recent years many fog/cloud frameworks have been proposed leveraging computing resources both at the edge of the network and in cloud data centers [6, 7].

In nowadays IoT and fog computing frameworks, a crucial component is the *gateway* device which enables communication of users, sensors, and actuators with edge devices and cloud resources [8]. Gateway devices could be small embedded computers, smart routers, or even smartphones. In IoT, the number of sensors and actuators has increased tremendously in the last few years [9, 10], including advanced gateway devices that emerged in recent fog environments [11]. Even though fog computing frameworks greatly simplify engineering gateway functionality, they do not focus on making generic gateway interfaces for seamless integration with diverse applications, user needs, and computation models.

Contributions. This paper presents the design and implementation of *iGateLink*, an open-source modular fog–cloud gateway library easing the development of applications running on Android-based IoT gateway devices. It provides common core functionalities, so as to let developers focus on the application-specific code, for example, implementing communications with specific sensors or protocols to exchange data with specific fog or cloud systems. *iGateLink* is specific to the mentioned IoT-fog use case but generic enough in order to allow simple extensions to be used in different IoT-fog integrated environments as shown in Fig. 1. It is also easy to use through a simple API and supports integration of different frameworks within the same application, including the possibility to run the required execution locally.

iGateLink has been applied to two use cases, one dealing with an oximeter-based healthcare application for patients with heart diseases, and the other one for low response time object detection in camera images. The presented framework proved to be effective for reducing development complexity and time, when comparing with existing practices in coding IoT gateway applications.

2 Related Work

Due to the vast heterogeneity in the Internet of Things, the importance of the IoT gateway in enabling the IoT has been acknowledged by several works [12–14]. Some authors propose IoT gateways to act only as routers, e.g., encapsulating raw data coming from *Bluetooth* devices in *IPv6* packets to be sent to the cloud [15]. However, in order to enable more complex scenarios, offload computations, and/or save network bandwidth, IoT gateways need to become "smart" by pre-processing incoming data from the sensors [16]. In this context, the use of smartphones as IoT gateways has been proposed [17, 18]; however, those works do not take into consideration the most recent fog and edge computing paradigms.

Regarding efforts to integrate the IoT with fog and edge computing, Aazam et al. [8] proposed a smart gateway-based communication that utilizes data trimming and pre-processing, along with fog computing in order to help lessen the burden on the cloud. Furthermore, Gia et al. [19] developed a Smart IoT gateway for a smart healthcare use case. Finally, Tuli et al. [6] developed *FogBus* an integration framework for IoT and fog/cloud. However, their work does not provide a generic application able to integrate IoT and fog/cloud frameworks, building their application from the ground up, tailored to their specific use case.

Finally, it is worthwhile to note that none of the previously mentioned works focuses on the design and implementation of the software that is required to run in the IoT gateway, especially in the case of an Android device, in order to make it generic and adaptable to many different scenarios, as this paper does.

3 System Model and Architecture

The principles discussed in Sect. 1 have been addressed by using a modular design, with a generic core on which different modules can be loaded.

A variation of the publish/subscribe paradigm [20] has been used, where the components collecting data from sensors and the ones sending it to the fog/cloud are the publishers and are called *Providers*, while the subscribers are the auxiliary components that manage the execution of the publishers or UI components that show incoming data to the user. The publish/subscribe paradigm is realized through an intermediate component that stores the incoming data from the *Providers*, called *Store*, and notifies the subscribers, called *Triggers*, using the *observer* design pattern. A *Store* can also be thought of as the *topic* to which the *Provider publishes* data, while the *Triggers* are its *subscribers*.

In the considered scenario, a *Provider* may be started either as a result of a user interaction (e.g., a button click) or as a consequence of external event (e.g., incoming Bluetooth data). It can also be started with some input data or with no data at all. The proposed model does not assume any of the aforementioned cases, employing a generic design, able to adapt many different scenarios. While the input of a *Provider*

can vary, the result is always data that needs to be stored in a *Store*. Whenever a *Provider* stores new data to a *Store*, all *Triggers* associated with the *Store* are executed. The most common use of a *Trigger* is to start another *Provider* but it could also be used, for example, to update the UI.

These design choices enabled (1) *modularity*, since *Providers* and *Triggers* can be independently and easily plugged and unplugged; (2) *flexibility*, since this model enables even more complicated use cases than the one mentioned above; and (3) *ease of use*, thanks to code reusability.

Furthermore, several *Providers* providing the same data (i.e., publishing to the same *Store*) can be active simultaneously, for example, *Providers* for different fog/cloud frameworks and/or local execution. In this case, it is useful to define a new component, the *Chooser*, whose function is to select a specific *Provider* among a list of equivalent ones in order to produce data. By doing so, it is possible to, for example, use another *Provider* when one is busy or to fallback to another *Provider* if one fails.

3.1 Implementation Details

Based on the model described above, we have implemented the *iGateLink* library for Android devices. The library is open-source and available at https://github.com/Cloudslab/iGateLink. From a high-level point of view (Fig. 2), the library is composed of a platform-independent core written in Java, an Android-specific module which extends the core to be efficiently used in Android and several extension modules that provide complimentary functionalities.

Core. The core of the library is composed of the following classes: *ExecutionManager*, *Data*, *Store*, *Provider*, *Chooser*, and *Trigger*. Refer to Fig. 3 for a conceptual overview of their interactions. The *ExecutionManager* class coordinates the other components and provides an API for managing them. The *Data* class is not properly a component but is the base class that every data inside this library must extend. It is characterized by an `id` and a `request_id`: the former must be unique between data of the same *Store*; the latter is useful for tracking data belonging to the same request. The *Store* class stores *Data* elements and provides two operations: `retrieve` for

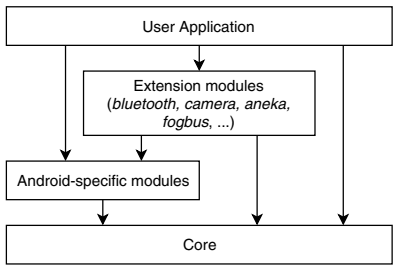

Fig. 2 High-level overview of the library software components

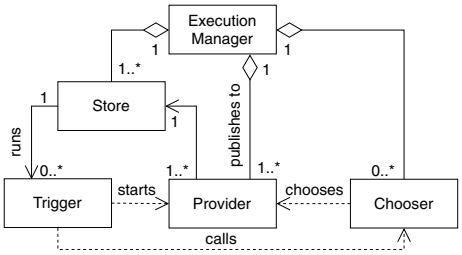

Fig. 3 Simplified UML class diagram of the core components

retrieving previously stored *Data* and `store` for storing new data. Every *Store* is uniquely identified by a `key`. There can be multiple *Stores* for the same *Data* type. Furthermore, a *Store* can have one or more *Triggers* associated with it that are called whenever new data is stored. For example, a common use for a *Trigger* is starting a *Provider* with the recently stored *Data* from another *Provider*. The *Provider* class takes some *Data* in input and produces some other *Data* in output. Every *Provider* is uniquely identified by a `key`. There can be multiple *Providers* for the same *Store* but a *Provider* can only use one *Store* at a time. A *Provider* can be started by calling its `execute` method either through *ExecutionManager*'s `runProvider` or `produceData`. In the latter case, the user specifies only which *Data* it wants to be produced (by means of a *Store* key) and a suitable *Provider* will be executed. In case there are two or more *Providers* for the same *Store*, a *Chooser* will be used.

Android-specific modules. When developing such an application on Android, a common problem is that using *worker threads* for time-consuming tasks that need to execute in the background still lets the Android runtime kill the app as needed, if not properly managed. This is addressed by providing the *AsyncProvider* class, which makes use of the *AsyncTask* class.[1] Furthermore, the *ExecutionManager* is hosted within an Android *Service*, which, if configured correctly as a foreground service, prevents it from being killed by the Android runtime.

4 Case Studies

In order to test the library and demonstrate its applicability to real applications, two case studies have been developed. They both reproduce existing works, namely, FogBus [6] and EdgeLens [7]. The first application connects to a *Bluetooth* oximeter to collect and analyze its data. The second application takes a photo and sends it to the fog or cloud for object detection.

In the original works, both applications have been developed using *MIT App Inventor*, which eases and speeds up the development but provides only a very simple API that cannot be adapted to more complicated use cases. Furthermore, every

[1] It provides a simple API for scheduling a new task and executing a custom function on the main thread before and after the execution.

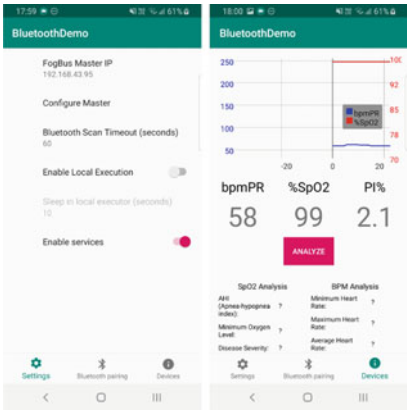

Fig. 4 Oximeter demo app: on the left, configuration screen; on the right, live data and analysis results screen

application needs to be built from the ground up even though many components may be in common, especially when using the same input method or framework. By developing the applications using *iGateLink*, both modularity and fast development can be achieved.

Oximeter-based healthcare application The *bluetoothdemo* application can be used to detect hypopnea in a patient by collecting data from an oximeter. The application consists of four screens: (1) the configuration screen (left picture in Fig. 4); (2) the Bluetooth device pairing screen; (3) the Bluetooth device selection screen through which the user chooses the device whose data to show; and (4) the data and analysis result screen (right picture in Fig. 4).

Data is collected in real time from the oximeter using *Bluetooth LE*. In order to do that, the *Bluetooth* module has been developed which provides an implementation of the *Provider* which registers to the *GATT* characteristics in order to receive push notifications from the *Bluetooth* device. Raw data received from the oximeter is then converted to a *Data* object and stored.

The user can then upload the data to FogBus [6] by tapping the "Analyze" button in order to get the analysis results. Data is sent using an HTTP request to the *FogBus* master node which forwards the request to a worker node (or the cloud) and then returns the result to the application. This simplifies the development of the application, since only one HTTP request is required.

Object detection application The *camerademo* application can be used to take a photo and run an object detection algorithm on it, namely, Yolo [21]. The user interface is very simple, with just a screen showing the camera preview and a button (Fig. 5). When the button is clicked, the photo is taken and sent to the fog/cloud for object detection. When the execution is terminated, the resulting image is downloaded and shown to the user.

In order to integrate Android camera APIs with *iGateLink*, the *camera* module has been developed, which provides an implementation of a *Provider*, namely, *CameraProvider*, that takes a photo when its *execute* method is called. When the photo is

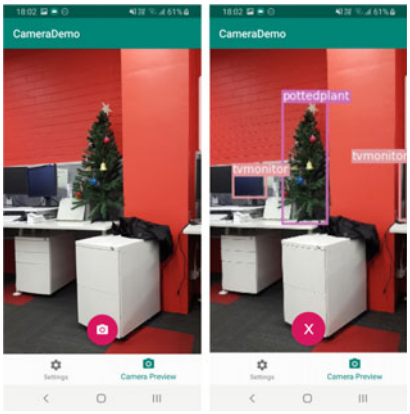

Fig. 5 Object detection demo app: on the left, preview screen; on the right, result screen

stored, two providers are executed: *BitmapProvider* that converts the photo from an array of bytes to a *Bitmap* object so that it can be plug in an *ImageView* and shown to the user, and one of *EdgeLensProvider* or *AnekaProvider* that executes the object detection on either EdgeLens [7] or Aneka [22]. The result of the object detection is, again, an array of bytes so it goes through another *BitmapProvider* before being shown to the user.

The *EdgeLensProvider* is responsible for uploading the image to the *EdgeLens* framework [7] and downloading the result once the execution is completed. *EdgeLens* has a similar architecture to *FogBus* but, differently from it, communication between client and worker is direct, instead of being proxied by the master. The usual *EdgeLens* workflow is (1) query the master to get the designated worker, (2) upload the image to the worker, (3) start execution, and (4) download the result when the execution is terminated.

The *AnekaProvider* is responsible for execution in Aneka [22] through its REST APIs for task submission. The image is first uploaded to an FTP server (which could be hosted by the master node itself), then the object detection task is submitted to *Aneka*, whose master node chooses a worker node to submit the request to. The worker downloads the image from the FTP server, runs the object detection on it, and uploads the resulting image back to the FTP server. In the meanwhile, the client repeatedly polls the master node in a loop, waiting for the submitted task to complete. When it does, the client finally downloads the result which is displayed to the user.

5 Conclusions and Future Work

In this paper, we presented a new Android library, *iGateLink*, which enables developers to easily write new Android applications linking IoT, edge, fog, and cloud computing environments, as it comes with a set of highly reusable modules for common IoT scenarios.

We demonstrated, by means of two use cases, that the library can adapt to different applications and is easy to use, increasing the engineering simplicity of application deployments and making such systems robust and easy to maintain.

As part of future work, more modules could be added to the library to cover the most common use cases, for example, the current version of the library does not include any module providing support for Bluetooth devices (currently only Bluetooth low energy is supported), built-in sensors (for example, accelerometer, gyroscope, magnetometer, luminosity), touch events, and audio recording. Furthermore, more fog/cloud frameworks could be integrated within the library, making it like plug-and-play software for end users.

Finally, while this paper focuses on the advantages, the library brings in terms of development time, a performance evaluation, and a comparison with existing systems could be carried out in the future.

References

1. Yi, S., Li, C., Li, Q.: A survey of fog computing: concepts, applications and issues. In: Proceedings of the 2015 Workshop on Mobile Big Data, pp. 37–42. ACM (2015)
2. Gill, S.S., Tuli, S., Xu, M., Singh, I., Singh, K.V., Lindsay, D., Tuli, S., Smirnova, D., Singh, M., Jain, U., et al.: Transformative effects of IoT, blockchain and artificial intelligence on cloud computing: evolution, vision, trends and open challenges. Internet Things 100118 (2019)
3. Ericsson Mobility Report (2019). https://www.ericsson.com/en/mobility-report
4. Cucinotta, T., Abeni, L., Marinoni, M., Balsini, A., Vitucci, C.: Reducing temporal interference in private clouds through real-time containers. In: 2019 IEEE International Conference on Edge Computing (EDGE), pp. 124–131 (2019)
5. Bonomi, F., Milito, R., Zhu, J., Addepalli, S.: Fog computing and its role in the internet of things. In: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, pp. 13–16. ACM (2012)
6. Tuli, S., Mahmud, R., Tuli, S., Buyya, R.: FogBus: A blockchain-based lightweight framework for edge and fog computing. J. Syst. Softw. **154**, 22–36 (2019)
7. Tuli, S., Basumatary, N., Buyya, R.: EdgeLens: Deep learning based object detection in integrated IoT , fog and cloud computing environments. In: 4th IEEE International Conference on Information Systems and Computer Networks (2019)
8. Aazam, M., Huh, E.N.: Fog computing and smart gateway based communication for cloud of things. In: 2014 International Conference on Future Internet of Things and Cloud, pp. 464–470. IEEE (2014)
9. Singh, D., Tripathi, G., Jara, A.J.: A survey of internet-of-things: Future vision, architecture, challenges and services. In: 2014 IEEE World Forum on Internet of Things (WF-IoT), pp. 287–292. IEEE (2014)
10. Lee, I., Lee, K.: The internet of things (iot): Applications, investments, and challenges for enterprises. Bus. Horizons **58**(4), 431–440 (2015)
11. Whiteaker, J., Schneider, F., Teixeira, R., Diot, C., Soule, A., Picconi, F., May, M.: Expanding home services with advanced gateways. ACM SIGCOMM Comput. Commun. Rev. **42**(5), 37–43 (2012)
12. Chen, H., Jia, X., Li, H.: A brief introduction to iot gateway. In: IET International Conference on Communication Technology and Application, pp. 610–613 (2011)
13. Datta, S.K., Bonnet, C., Nikaein, N.: An iot gateway centric architecture to provide novel m2m services. In: 2014 IEEE World Forum on Internet of Things (WF-IoT), pp. 514–519. IEEE (2014)

14. Kang, B., Kim, D., Choo, H.: Internet of everything: a large-scale autonomic iot gateway. IEEE Trans. Multi-Scale Comput. Syst. **3**(3), 206–214 (2017)
15. Zachariah, T., Klugman, N., Campbell, B., Adkins, J., Jackson, N., Dutta, P.: The internet of things has a gateway problem. In: Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications, pp. 27–32. ACM (2015)
16. Saxena, N., Roy, A., Sahu, B.J., Kim, H.: Efficient iot gateway over 5g wireless: a new design with prototype and implementation results. IEEE Commun. Mag. **55**(2), 97–105 (2017)
17. Kamilaris, A., Pitsillides, A.: Mobile phone computing and the internet of things: a survey. IEEE Internet Things J. **3**(6), 885–898 (2016)
18. Aloi, G., Caliciuri, G., Fortino, G., Gravina, R., Pace, P., Russo, W., Savaglio, C.: A mobile multi-technology gateway to enable iot interoperability. In: 2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI), pp. 259–264. IEEE (2016)
19. Gia, T.N., Jiang, M., Rahmani, A.M., Westerlund, T., Liljeberg, P., Tenhunen, H.: Fog computing in healthcare internet of things: a case study on ecg feature extraction. In: CIT/IUCC/DASC/PICom 2015, pp. 356–363. IEEE (2015)
20. Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.M.: The many faces of publish/subscribe. ACM Comput. Surv. (CSUR) **35**(2), 114–131 (2003)
21. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: unified, real-time object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 779–788 (2016)
22. Vecchiola, C., Chu, X., Buyya, R.: Aneka: a software platform for .net-based cloud computing. High Speed Large Scale Sci Comput **18**, 267–295 (2009)