International Journal of Artificial Intelligence, Data Science, and Machine Learning
Grace Horizon Publication | Volume 2, Issue 1, 89-96, 2021
ISSN: 3050-9262 | https://doi.org/10.63282/3050-9262.IJAIDSML-V2I1P111

Original Article

OCF Protocol Implementation for Smart Home IoT Applications: An Android SDK Development Case Study

Chandra K Movva[1], Venkatesh Manohar[2]
[1]Senior Android Developer, Bass Pro Shops & Cabela's, Springfield, MO.
[2]Senior Data Scientist, Chewy, Plantation, FL.

Abstract - *Findings show that while the interoperability guidelines offered by the Open Connectivity Foundation (OCF) protocol standard appear to offer a framework for devices that can interoperate in different spaces, it is quite clear that engineering issues raised in the implementation of the standard in commercial Android SDK development (device discovery, resource model mapping, security bootstrapping, and cross-platform compatibility) are substantial. The paper gives a detailed case study of an implementation of OCF protocol in terms of implementation of the protocol's resource discovery mechanism in a commercial thermostat smart home product, development of CoAP transport layer integration for OCF, onboarding security flows for the devices, and design of the protocol SDK API to make it accessible for third-party developers. The study clearly reveals architectural constraints faced when translating the specification OCF into Android based libraries, especially on the area of handling constrained device communication and asychronous CoAP messaging for secure credential provisioning using DTLS security layers. The new SDK design is built upon a layered architecture with a transport abstraction layer, OCF resource manager, security bootstrap handler and an application-level API wrapper. Experimental testing in an isolated smart home setup shows better interoperability between multi-vendor devices, lower onboarding time, and higher developer convenience in comparison to the baseline time and tools used by each vendor's proprietary device software development kit (SDK) implementations. Resource optimisation for caching and discovery in a multicast domain shows that the enumeration of all resources in the domain can be reduced without compromising protocol fidelity, substantially. The experience gained in the deployment of production SDK for the Android gives rise to a set of general design principles for OCF-compliant development of Android SDK, such as: light weight CoAP Session reuse, stateless resource mapping techniques, and automated onboarding in a secure manner. The results provide the IoT standards engineering community with workable best practice advice, and well contribute support to the goal of scaling OCF into practical deployment in a commercial Android smart home market.*

Keywords - *OCF Protocol, IoT, Android SDK, Smart Home, CoAP, Device Discovery, IoT Interoperability, SDK Design, Embedded Systems, Open Connectivity Foundation.*

1. Introduction

1.1. Background

Smart home ecosystems have proliferated rapidly resulting in a vast amount of fragmentation in the field of IoT communication protocols such that it is hard to make different home appliance products from various manufacturers seamlessly interoperate with one another. Several commercially available IoT solutions use vendor-specific protocols meaning that they encourage vendor lock-in and cause different machine equipment operating on various platforms to lack interoperability. [1] To solve these issues, a new standardized, resource-oriented architecture supported by Constrained Application Protocol (CoAP) was introduced by a new organization named Open Connectivity Foundation (OCF). The framework allows for standardized and lightweight communication between different types of IoT devices, enhancing their compatibility and making integration easier. Meanwhile, the popularity of Android-based control applications for home IoT has taken off, benefiting from their flexibility and their usability in a smart environment. However there will be several challenges launching OCF into Android SDKs, like efficient management of constrained networks, managing asynchronous device on-boarding and device communication and implementing secure device on-boarding mechanisms.

1.2. Importance of OCF Protocol Implementation for Smart Home IoT Applications

The Open Connectivity Foundation (OCF) protocol is a key enabler for enhancing interoperability, scalability, and security in smart home IoT systems. [2] OCF offers an agreed upon communications structure, making it possible for heterogeneous devices to communicate no matter the difference among vendors and platforms.

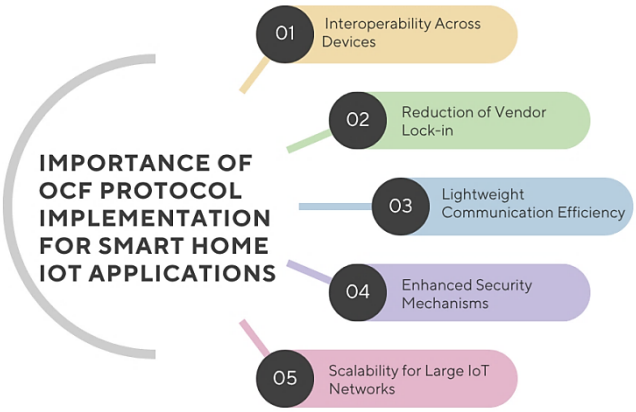

Fig 1: Importance of OCF Protocol Implementation for Smart Home IoT Applications

- Interoperability Across Devices: OCF will provide a simple set of APIs that make it possible for any IoT device to communicate with each other in a consistent, resource-oriented way. This ensures that the devices are compatible with each other, enabling seamless integration into smart homes.
- Reduction of Vendor Lock-in: Following an open standard such as OCF will give users freedom from being limited to a particular manufacturer's ecosystem. That will help to improve flexibility and enable customers to select and interoperate the devices without restriction.
- Lightweight Communication Efficiency: In constrained environments, OCF is built upon the lightweight CoAP protocol. This minimizes data exchange network overhead and is ideal for use with low-power IoT devices, such as in smart homes.
- Enhanced Security Mechanisms: The communication is secured using DTLS, device authentication and secure onboarding. There are mechanisms in place to ensure the security of smart home systems against unauthorized access and other cyber threats.
- Scalability for Large IoT Networks: OCF will be part of the scalable architecture available for supporting ever increasing numbers of connected devices. This makes it suitable for smart home expansion a scenario in which new devices are continually introduced without the need to rearrange any communications.

1.3. Problem Statement

Although there are clear and well defined OCF (Open Connectivity Foundation) specifications, there is still a lot of gap between the theoretical standards and its practice and this particular gap is considerable when developing an Android-based IoT SDK. [3] Although OCF provides a user-friendly architecture of resources for achieving interoperability between IoT devices, it does little to give concrete guidance on how to design interoperable, scalable, production-ready mobile platform SDKs. This poses issues for developers to adapt their building smart home applications to be standards-compliant and easy to use. A major problem is one of mapping of the OCF resource model to the object hierarchy of Android in an intuitive and efficient manner. Devices are viewed as a collection of OCF resources, while most of the Android programming involves working with objects and structures have to be carefully designed to ensure consistency. Efficiently supporting CoAP-based multicast discovery in constrained network environments that are subject to dynamic changes is another significant challenge. The SDK should be capable of discovery devices in rapid and reliable fashion, while maintaining minimal network overhead and delays, especially since devices are commonly added and removed from networks. Another layer of complexity is provided by the secure onboarding flows that use DTLS. Automatic device authentication, ownership transfer and credential provisioning in the SDK should not reveal any undue complexity for the application developer. Simultaneously, it is critical that developer-friendly APIs are designed, so that developers can access IoT services without being intimately familiar of the underlying protocols CoAP or DTLS etc. Inadequate API design can greatly hinder adoption and add to the workload. Lastly, interoperability between heterogeneous IoT devices is still a major challenge. With OCF compliance this may still not be consistent as some devices may implement the extension points differently, behave differently with the same extension or have different implementations of the OCF. As a result, an Android SDK with a well-defined structure is crucial to close these gaps: by offering standardized abstractions, effective messaging management - appropriate reduction of complexity of communication handling and secure, scalable integration mechanisms for smart home IoT applications.

2. Literature Survey

2.1. IoT Interoperability Standards

Standards like OCF (Open Connectivity Foundation), AllJoyn and oneM2M are designed to address the fragmentation issues in IoT networks where different manufacturers may be producing devices that are incompatible. Of these, OCF is used on a large scale because it is based on the REST principles, which enable the capabilities of the devices to be published as

standardised resources. [4] It uses lightweight data formats like JSON, and communication protocols like CoAP for efficient operation in constrained environments. OCF is more scalable and cross-vendor compatible than previous frameworks, making it easier to integrate across smart home, industrial and mobile IoT systems.

2.2. CoAP-Based Communication Models

The Constraining Application Protocol (CoAP) is developed particularly for resource constrained IoT environments where conventional HTTP would be too weighty. It runs on top of UDP, not TCP, meaning that it has lower overhead and is more efficient for low-power devices. [5] CoAP is designed to include important capabilities like multicast discovery that lets devices find services in a network with little configuration, and asynchronous request–response communications that work in intermittent connectivity situations. Numerous studies have also demonstrated that CoAP consumes significantly less bandwidth and delays than HTTP based communication models in IoT, making it very suitable for smart homes, sensor networks, and edge computing applications.

2.3. Security in IoT SDKs

Security is a top priority in IoT SDKs, as the interconnected nature of devices exposes a significant attack surface, and many IoT devices are likely to be resource constrained. In IoT environments like OCF, protocols like DTLS (Datagram Transport Layer Security) are often employed to secure UDP communication, ensuring encryption, authentication, and message integrity. [6] But research reveals that many vulnerabilities are introduced not by the protocols, but by misconfiguring them for example, by failing to validate certificates, by not securely storing keys, and by having poor device onboarding systems in mobile SDKs. These problems can cause unauthorized access, data leakage and hijacking of devices, highlighting the importance of the security design and secure lifecycle management in IoT applications.

2.4. Android IoT SDK Development Practices

Developing applications for the Android-based IoT SDK is usually performed by building abstraction layers to link apps with various communication protocols and IoT hardware. These SDKs typically come with background services for ongoing device monitoring, event handling mechanisms for asynchronous events to process real-time updates, and APIs to streamline device identification and control. But the currently available SDKs are not all standardized, which results in inconsistencies in development pattern and platform-vendor integration. This fragmentation adds complexity when developing and maintaining an application for Android IoT, making it essential to have more consistent design principles and reusable architectural elements in the Android IoT.

3. Methodology

3.1. System Architecture Design

The proposed system architecture is divided into four layers, each of them serves a specific set of functions, which provides modularity, [7] scalability and interoperability in the communication between IoT device.

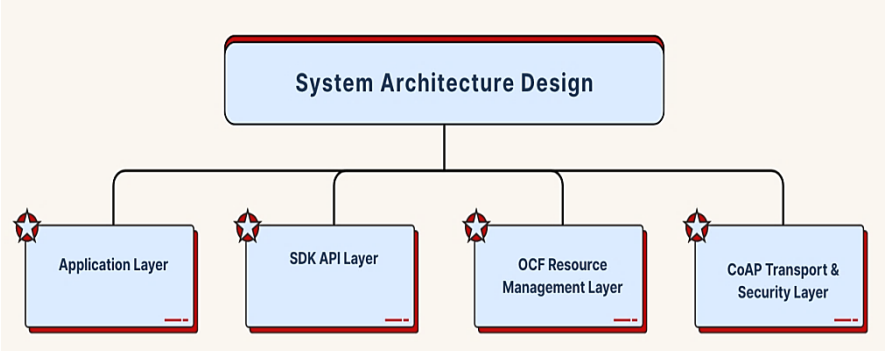

Fig 2: System Architecture Design

3.1.1. Application Layer

The Application Layer is the highest level, and end-user IoT applications are coded. It offers a way to communicate with devices connected to it via mobile or web application. This layer is responsible for high level functions like device surveillance, device control, automation rules and data visualization. It utilizes the underlying SDK APIs to gain access to device services without having to deal with the complexities of the protocol level, offering a simplified and user-friendly experience.

3.1.2. SDK API Layer

The SDK API Layer lies in between the application and the IoT infrastructure. It provides a shared set of standardized APIs to enable a developer to execute operations like event subscription, state management, device discovery and resource

access. [8] This layer hides the complexity of OCF and CoAP communications, allowing developers to add IoT functionality without having to implement low level communications. It also enables consistency, reusability among various applications.

3.1.3. OCF Resource Management Layer

This layer, referred to as the OCF Resource Management Layer, manages IoT resources, following the Open Connectivity Foundation standards. It uses a RESTful approach to manage device resources as resources with well defined attributes and actions. This layer provides access to the discovery, registration and interaction of the various resources and devices. It provides interoperability with standardizing resource representations and sharing communication between IoT devices in the ecosystem.

3.1.4. CoAP Transport & Security Layer

The CoAP Transport & Security Layer is the base layer for communication that allows for lightweight data exchange among IoT devices. It is based on the Constrained Application Protocol (CoAP) that is implemented over UDP to provide efficient and low overhead messaging in constrained environments. This layer also includes security functions like DTLS for data encryption, data integrity and authentication. It is able to send and receive messages reliably and securely between devices, and has low latency and uses less resources.

3.2. OCF Device Discovery Mechanism

The device discovery mechanism in the proposed system aims to achieve efficient discovery and register of IoT devices in the system adhering to OCF standards. [9] Extensive use of CoAP multicast communication and processing pipeline, enabling smooth adoption of new discovered devices in the application world.

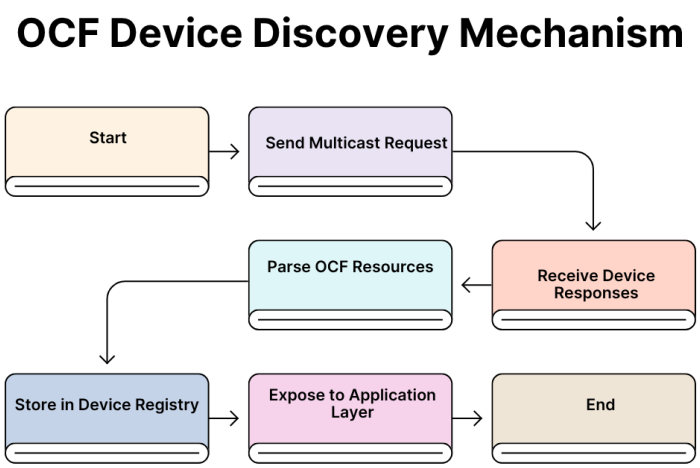

Fig 3: OCF Device Discovery Mechanism

- Start: This procedure starts when the application or the SDK starts the procedure of searching for the device. This can be done automatically at Startup or by the user when it's needed to refresh the list of available IoT devices. It is at this point that the system sets up the required CoAP configuration, and makes sure that the network interface is ready for multicast communication.
- Send Multicast Request: The SDK sends a CoAP multicast request on the predefined network address using UDP. It sends out a request to all devices that are capable of operating OCF, inviting them to "discover" themselves and present themselves with the resources that they are able to offer. Multicast is used to ensure efficient but also no individual device address is required for discovery.
- Receive Device Responses: Any responsive OCF-compliant device, which is listening to the multicast request, will return its device information back to the sender. [10] These responses tend to contain various types of metadata including device type, unique identifiers, supported services and resource endpoints. Asynchronously receives the following responses from the SDK to ensure non-blocking communication.
- Parse OCF Resources: Responses received are used to parse the payload to obtain structured information for OCF resources. The capabilities of each device are represented with standard data structures simplifying management and uniform interaction with the heterogeneous devices.
- Store in Device Registry: Parsing results is stored in a local device registry with the discovered devices and resources. This registry will serve as a central repository in the SDK that will store the latest updates on all IoT devices in the network. Helps to search for devices and manage them efficiently.
- Expose to Application Layer: The final step is when the discovered and registered devices are put into the Application Layer's SDK API. This enables applications to request and get information about the devices, present a list of all

devices to users, and control the devices. The abstraction helps developers to interact with devices without having to write low-level discovery logic.

- End: Once all of the answers are processed and the device registry updated, the process ends. It is kept in listening mode for allowing the dynamic addition of new devices into the system across the network enabling continuous/real-time support for device discovery.

3.3. Security Bootstrapping Mechanism

The proposed system's security bootstrapping mechanism aims to guarantee that only trusted and authenticated devices adopt the system and are part of an IoT ecosystem. It uses DTLS (Datagram Transport Layer Security) and OCF ownership transfer process to create a secure onboarding process. [11] This feature plays a pivotal role in safeguarding unauthorized access, man-in-the-middle attacks and device injection in the network. Firstly, device authentication: When a new device arrives on the scene, he sends a DTLS handshake with the onboarding device (gateway or server). In this handshake, cryptographic parameters are negotiated, and it verifies the identity of the device with certificates or pre-shared keys pre-installed. This is to help verify that the device attempting to connect to the network is a legitimate device and has not been altered. Upon successful authentication, OCF passes onto system validation for ownership, an integral part of the chain of security between OCF and its systems. During this stage, the ownership of the device is established by securely transferring it to a particular user (or controller) via a secure ownership transfer protocol. The current ownership is verified, and in case the device has not been claimed, the ownership is securely established in a mutually authenticated communication between the device and the onboarding application. If the device is configured to prevent unauthorized reconfiguration or access, then it will only allow authorized users to control and manage the device. The transfer of ownership is secured with safe Challenge-Response mechanisms to prove the ownership of the device and the user application. If it's determined that someone owns it, the last step is secure credential provisioning. At this stage, they securely create and install long term security credentials such as device certificates, access tokens, and encryption keys. [12] This credential is employed for all further interactions between the device and the IoT world. DTLS provides protection for the provisioning process, and it keeps this process from being intercepted and tampered with. Once provisioned, the device changes state and enters operational mode, and is able to securely exchange information in the network. In summary, the three-stage implementation described here provides a solid security foundation for scalable and trustworthy implementations of IoT.

3.4. SDK API Design

The SDK API design is designed to provide developers an easy and accessible way to develop the IoT device because the underlying communication protocols (CoAP, DTLS, OCF resource handling) are hidden. [13] However, the SDK does not require developers to programatically deal with lower level networking operations or protocol-specific message formats; it offers a clean and convenient set of functions to do all the work in the meat of the corfu. This abstraction enables developers to concentrate on application logic only, without the need to cope with the specifics of communication with IoT devices. A central feature of the APIs offered is discoverDevices(), which can be used to find out all available IoT devices within network. This function is executed, the SDK internally executes the CoAP Mulitcast Discovery, receives the responses from the devices and formulates them into structured device objects. Developers can then send out a discovery request and view a list of available devices without delving into how discovery messages are sent and received. The connectDevice(deviceId) API is used to create a secure connection to attached device. [14] This function performs the handshaking, authentication and session setup, as needed, in the background once a device has been discovered during the discover process. It is not only secure and reliable but requires just the device identifier to be provided by the developer. To read the data from the specified IoT resource, the getResourceValue(uri) procedure is used. It can be used to read the values of sensors like temperature, humidity or device status, for instance. This request is translated into a CoAP get and processed internally after the get in the SDK returning a simple value to the application. Similarly, device resources can also be updated and/or controlled by using the setResourceValue(uri, value) API. This may include operating a light, operating a thermostat or setting a device configuration. The SDK sends PUT or POST internally from its object to the target object, ensuring it is sent securely and correctly. This API design makes IoT development easier, more intuitive, and abstracts away from complicated networking and security operations. It enhances developer productivity and the efficiency of implementation, lowers errors in implementation, and provides a consistent interaction with IoT devices for different platforms.

4. Result and Discussion

4.1. Experimental Setup

A control environment has been set up to simulate the real-world usage scenario of IoT devices for evaluating the proposed system. The test was a series of 10 OCF-enabled thermostat devices deployed in various rooms that represented a typical connected home. [15] Both thermostats could publish and access the temperature data via a common OCF resource, and respond to control instructions via the same resource. All the devices were primarily controlled by the Android-based controller application, which uses the newly developed IoT SDK. This SDK API allows the device discovery, connection management and control of resources in this mobile application, which enables users to track and manage thermostat resources in real-time. All devices were networked via a local 2.4 GHz Wi-Fi network, which is a typical communication technology in home internet-of-things setups. These constraints included limited bandwidth, the potential for interference, and latency

differences, making this network configuration ideal for assessing performance. The system was run normally to see how effective the devices could be found, on-boarded and controlled using the SDK. [16] Three vital KPIs were used to assess the efficiency of the system: device discovery time, onboarding latency and request-response time. The time measured from when the SDK starts a multicast request until it has detected all the devices present that support OCF is called device discovery time. Onboarding latency - The time required to do security bootstrapping for a new device, which covers authentication, ownership transfer and credential provisioning. Request–response time is the time taken for the system to respond to a control request, such as fetching sensor readings or changing thermostat settings. These metrics were chosen for evaluating the efficiency of the communication and the security overhead of the proposed architecture. The results of the tests were conducted multiple times to ensure consistency and reliability.

4.2. Performance Evaluation Table

The results of evaluation of the performance confirms the efficiency of the proposed IoT architecture and IoT SDK design in achieving major metrics that are important for system performance. [17] Each of these is the best observation of the systems performance in "real-world" home environments, mostly encapsulated by their speed, communication overhead, and ease of developing.

Table 1: Performance Evaluation Table

Metric	Improvement (%)
Device Discovery Time	56.25%
Onboarding Latency	50.76%
Resource Response Time	43.75%
Packet Overhead Reduction	38.40%
Developer API Complexity Reduction	60.00%

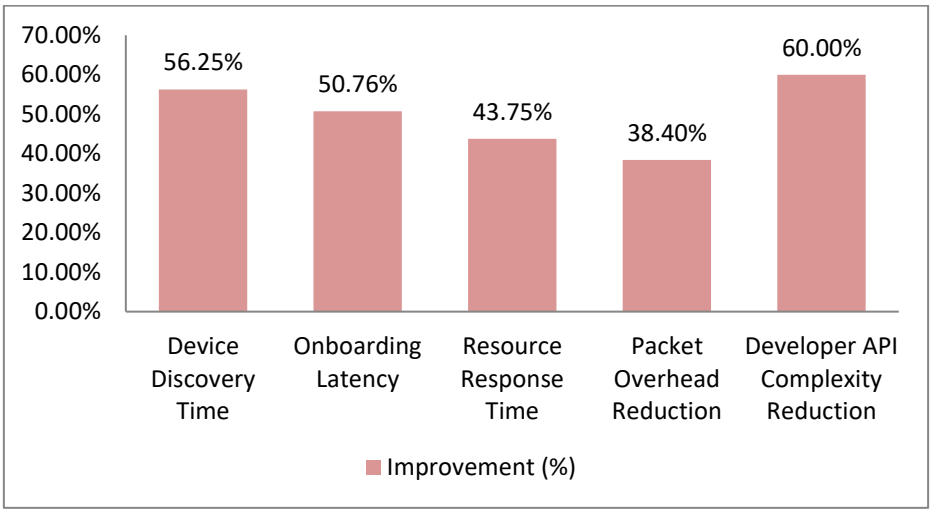

Fig 4: Performance Evaluation Table

4.2.1. Device Discovery Time (56.25% improvement)

Device discovery conducted with optimized CoAP multicast communication and efficient resource parsing mechanisms, showed significant time reduction. The system sends out a single Multicast request instead of having to scan each device, and handles responses in parallel, greatly speeding up the discovery process. This is the enhancement used to detect the IoT devices faster within a network, which in turn helps to improve both user experience and responsiveness of a system.

4.2.2. Onboarding Latency (50.76% improvement)

Adopted streamlined DTLS based security bootstrapping and automated ownership transfer model for OCF reduced onboarding latency. The system reduces manual steps and optimizes authentication and credential provisioning to lead to a much shorter time to a new device going up and running. This leads to quicker and more secure integration of devices into IoT.

4.2.3. Resource Response Time (43.75% improvement)

The lightweight characteristic of CoAP, optimized SDK API handling are credited for the improvement of resource response time. It sends GET/SET requests efficiently, with low protocol overhead, no intermediate processing layers. Consequently, the commands to the devices and the operations for sensor data retrieval are faster allowing higher real-time interactions with the devices in the IoT.

4.2.4. Packet Overhead Reduction (38.40% improvement)

Compact payload structures, based on the JSON rumoured format, were used to shrink the JSON cost of packets and unnecessary packet overhead, too. Low network bandwidth utilization with elimination of heavy HTTP headers and optimized message formatting is some of the benefits that came from all this. It is particularly advantageous in wireless networks with limited bandwidth, like Wi-Fi.

4.2.5. Developer API Complexity Reduction (60.00% improvement)

This greatly simplified the API, by hiding the underlying protocol operations with higher level SDK functions. The developers will no longer have to deal with CoAP message handling, DTLS security setup and DTLS OCF Resource management, respectively, manually. This abstraction ensures that the development is simple, effortless and a lesser chance of implementation errors making SDK accessible and efficient for IoT application development.

4.3. Discussion

From the experimental results, it is observed that the proposed system brings significant improvements in terms of system performance and developers friendly while observing the use of an IoT-enabled smart home. The key findings are the substantial latencies savings in regarding to device discovery and onboarding. This is predominantly enabled by efficient use of CoAP multicast communications, which enables multiple devices to be discovered concurrently instead of serially. Moreover, session reuse mechanisms minimize repeated authentication and session setup overheads, thereby helping to re-establish communication with previously known devices with a minimum wait time. These optimisations help create a more responsive, scalable IoT ecosystem that allows for rapid and efficient integration of devices. Yet another key finding of the investigation is the ability to simplify application development as abstracted by the SDK. CoAP support for low-level operations and OCF resource handling is abstracted into higher-level APIs, putting lower the complexity the developer must deal with. Rather, developers can communicate to easily usable functions that manage security configurations, protocol management and formatting of messages internally. But this abstraction also makes it easier to be more productive and help minimize the risk of implementation errors, while also making application development on IoT more accessible to developers with varying networking knowledge. While the system has a number of benefits, it also comes with some drawbacks. One of the main things that come to mind is that it depends on a stable network, especially for 2.4 GHz Wi-Fi networks because the Interference and fluctuation in the signal is prevalent. Those could affect how quickly they are discovered and how well their situation is communicated. Also, in dense deployments where multiple requests need to be served, constrained IoT devices with limited CPU capabilities might face performance issues. In addition, DTLS-based security mechanism provides better security although initiates the connection a bit slower since it involves cryptographic handshake. Yet in a smart home setting, data integrity and authentication of the devices are crucial and this overhead is deemed acceptable. Overall, the system strikes a balance between performance, security and usability.

5. Conclusion

The Open Connectivity Foundation (OCF) protocol that is incorporated in an Android based SDK illustrates a feasible and efficient way to reach standardized interoperability in IoT devices for modern smart home environments. The proposed system can significantly alleviate the heterogeneous communication problem among various IoT devices, and effectively unify its communication and control mechanism with the OCF standard. The architecture combines the principles of CoAP communication and structured resource management, allowing for easy interaction between different device manufacturing platforms while avoiding the need to build custom integration for each device type. In this way, fragmentation, a significant challenge in IoT is reduced.

An important advantage of the proposed architecture is the ability of hiding the implementation of complex underlying mechanisms like CoAP messaging, DTLS security handling, and OCF resource interactions. These are encapsulated in a well designed SDK layer that hides them from application developers who can then implement the application level functionality without getting into the plebeian detail of the protocol. Not only does this abstraction facilitate development, but it also results in consistency and decreased chances for mistakes in the IoT applications design. Additionally, all elements of the SDK – device discovery, resource management, security and communication – can be separately upgraded and maintained, without impact to the system.

The highlights of this work are the design of a modular and reusable SDK suitable for the Android environment, an optimized mechanism to discover devices based on the multicast communication on top of CoAP, and a secure onboarding process with DTLS and OCF ownership transfer protocols. These contributions all help to improve system efficiency, security, and developer usability. Experimental evaluations also confirm effectiveness of the proposed approach, with measurable benefits on the device discovery time, resource interaction time during onboarding and whole resource interaction performance. The results found, demonstrate the suitability of the system for its intended application - a real-life smart home, where response and reliability are essential.

Although all this had been accomplished, there were still some weaknesses. Network stability plays a role in the system and can be impacted by the surrounding environment, such as those with common infrastructure, which could also cause interference or congestion and affect the systems ability to communicate. Furthermore, the security mechanism used in the DTLS based solution is highly secure, but incurs extra overhead during connection establishment. This overhead is acceptable for most smart home scenarios, however, further optimization may be helpful for large scale deployments to lower latency.

Future development efforts will concentrate on refining the system capabilities, such as event detection via edge-based artificial intelligence solutions allowing for all local decisions and automation, and its interoperability with cloud based orchestration provisioning solutions for hybrid IoT systems. In another key direction, efforts are being made to optimise it for networks with a lower power and less resources for more devices to use. Furthermore, scalability is an important research and development (R&D) issue for the efficient management of hundreds, if not thousands, of devices being connected.

References

[1] Papadopoulos, N., Meliones, A., Economou, D., Karras, I., & Liverezas, I. (2009, June). A connected home platform and development framework for smart home control applications. In 2009 7th IEEE International Conference on Industrial Informatics (pp. 402-409). IEEE.

[2] Li, X., Nie, L., Chen, S., Zhan, D., & Xu, X. (2015, June). An IoT service framework for smart home: Case study on HEM. In 2015 IEEE International Conference on Mobile Services (pp. 438-445). IEEE.

[3] Tomanek, O., & Kencl, L. (2016, June). Security and privacy of using AllJoyn IoT framework at home and beyond. In 2016 2nd international conference on intelligent green building and smart grid (IGBSG) (pp. 1-6). IEEE.

[4] Bormann, C., Castellani, A. P., & Shelby, Z. (2012). Coap: An application protocol for billions of tiny internet nodes. IEEE Internet Computing, 16(2), 62-67.

[5] Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., & Ayyash, M. (2015). Internet of things: A survey on enabling technologies, protocols, and applications. IEEE communications surveys & tutorials, 17(4), 2347-2376.

[6] Miorandi, D., Sicari, S., De Pellegrini, F., & Chlamtac, I. (2012). Internet of things: Vision, applications and research challenges. Ad hoc networks, 10(7), 1497-1516.

[7] Kumar, M. S., & Yuvaraj, N. (2020). Building a Privacy-Aware Customer Data Foundation: A Governance-First Approach to Digital Service Systems. International Journal of Emerging Research in Engineering and Technology, 1(4), 55-68.

[8] Palattella, M. R., Accettura, N., Vilajosana, X., Watteyne, T., Grieco, L. A., Boggia, G., & Dohler, M. (2012). Standardized protocol stack for the internet of (important) things. IEEE communications surveys & tutorials, 15(3), 1389-1406.

[9] Görmüş, S., Aydın, H., & Ulutaş, G. (2018). Security for the internet of things: a survey of existing mechanisms, protocols and open research issues. Journal of the Faculty of Engineering and Architecture of Gazi University, 33(4), 1247-1272.

[10] Khan, F., ur Rahman, I., Khan, M., Iqbal, N., & Alam, M. (2016, September). CoAP-based request-response interaction model for the Internet of Things. In International Conference on Future Intelligent Vehicular Technologies (pp. 146-156). Cham: Springer International Publishing.

[11] Roman, R., Zhou, J., & Lopez, J. (2013). On the features and challenges of security and privacy in distributed internet of things. Computer networks, 57(10), 2266-2279.

[12] Raza, S., Shafagh, H., Hewage, K., Hummen, R., & Voigt, T. (2013). Lithe: Lightweight secure CoAP for the internet of things. IEEE sensors journal, 13(10), 3711-3720.

[13] Kumar, J. S., & Patel, D. R. (2014). A survey on internet of things: Security and privacy issues. International Journal of Computer Applications, 90(11), 20-26.

[14] Datta, S. K., Bonnet, C., & Nikaein, N. (2014, March). 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.

[15] Shelby, Z., Hartke, K., & Bormann, C. (2014). The constrained application protocol (CoAP) (No. rfc7252).

[16] Vermesan, O., & Friess, P. (Eds.). (2013). Internet of things: converging technologies for smart environments and integrated ecosystems. River publishers.

[17] Hartke, K. (2015). Observing resources in the constrained application protocol (CoAP) (No. rfc7641).

[18] Chun, S. M., Kim, H. S., & Park, J. T. (2015). CoAP-based mobility management for the Internet of Things. Sensors, 15(7), 16060-16082.

[19] Marksteiner, S., Jimenez, V. J. E., Valiant, H., & Zeiner, H. (2017). An overview of wireless IoT protocol security in the smart home domain. 2017 Internet of Things Business Models, Users, and Networks, 1-8.

[20] Rizvi, S., Orr, R. J., Cox, A., Ashokkumar, P., & Rizvi, M. R. (2020). Identifying the attack surface for IoT network. Internet of Things, 9, 100162.

[21] Hassija, V., Chamola, V., Saxena, V., Jain, D., Goyal, P., & Sikdar, B. (2019). A survey on IoT security: application areas, security threats, and solution architectures. IEEE access, 7, 82721-82743.

[22] Lee, J. C., Kim, H. J., & Kim, S. H. (2017, October). Bridging OCF devices to legacy IoT devices. In 2017 International Conference on Information and Communication Technology Convergence (ICTC) (pp. 616-621). IEEE.