Web API Management Meets the Internet of Things

Paul Fremantle, Jacek Kopecký and Benjamin Aziz

University of Portsmouth, Portsmouth PO1 3HE, UK,
{paul.fremantle, jacek.kopecky, benjamin.aziz}@port.ac.uk

Abstract. In this paper we outline the challenges of Web API management in Internet of Things (IoT) projects. Web API management is a key aspect of service-oriented systems that includes the following elements: metadata publishing, access control and key management, monitoring and monetization of interactions, as well as usage control and throttling. We look at how Web API management principles, including some of the above elements, translate into a world of connected devices (IoT). In particular, we present and evaluate a prototype that addresses the issue of managing authentication with millions of insecure low-power devices communicating with non-HTTP protocols. With this first step, we are only beginning to investigate IoT API management, therefore we also discuss necessary future work.

1 Introduction

Web APIs are capabilities offered across the web that are designed to be accessed by software rather than people. Unlike traditional APIs, Web APIs are inherently public or semi-public in that they are designed to be used over the public Internet and not solely over private networks or VPNs. The public nature of Web APIs poses a number of challenges addressed by the emerging area of *API Management*.

The Internet of Things is the name given to the systems that connect the real world to the Internet. This includes both sensors that measure the world around us (including pollution sensors, weather monitors, car parking space sensors, baby monitors and many others) and actuators that affect the world (including automated lighting systems, internet-connected door locks, and many others). Complex devices such as Connected Cars, Connected Homes, and so forth, combine multiple sensors and actuators. Due to their low power, IoT devices often employ non-HTTP protocols such as MQ Telemetry Transport (MQTT) and Constrained Application Protocol (CoAP).

Inevitably the Internet of Things will need to engage with Web APIs. For now, most IoT devices connect to services that are created by the provider of the hardware, and so are using private APIs. Public APIs are an increasingly important factor. There are a set of companies that are providing common cloud services and corresponding APIs for IoT (such as Xively [5]), and there are emerging API standards for IoT communication (such as HyperCat [17]). Much

of the envisioned strength of the IoT will emerge when data from multiple sources can be aggregated, analysed and acted upon. This will increase the demand for IoT devices to communicate with open Web APIs.

Our work addresses the new problem of adapting the principles and technologies of Web API management to the landscape of the IoT, which poses challenges stemming from the great numbers and low power of IoT devices, compared to typical full-fledged clients for Web APIs. The problems we are addressing can be clearly stated:

- What is the impact of the Internet of Things onto Web APIs and Web API Management
- How do IoT devices identify themselves to Web APIs over IoT protocols?
- How can we add IoT protocol support to existing Web API Management systems?
- What is the impact of adding identity, usage control and analytics to existing IoT protocol interactions?

This paper provides three clear contributions: firstly, the identification of new challenges that emerge from the use of Web APIs from IoT devices, especially those around authentication, usage control and analytics. Secondly, we have implemented a prototype which we believe is the first of its kind to add support for IoT specific protocols to API management systems. Finally we provide early experimental data on the performance of this prototype.

The rest of the paper is laid out as follows. In the next section, we look more closely at the area of Web API Management. We then we review related work that gives us a basis for defining in Section 4 the unique challenges of Web API management for the IoT, especially in connection with binary protocols such as MQ Telemetry Transport (MQTT) or the Constrained Application Protocol (CoAP). Section 5 introduces our prototype, a messaging gateway we call IGNITE, designed to allow us to evaluate the viability and performance of our approach; in Section 6 we present the design and results of our performance experiments. We conclude the paper in Section 7 with a summary and a discussion of further work we expect in this open research area.

2 Web API Management

There is no universal definition of this space, and little academic research as yet, but the authors' industrial experience in this area, together with a review of [14, 16, 21] identifies a set of key areas to be addressed:

- Publishing details of the APIs, documentation, SDKs and other human- and machine-readable material in a portal aimed at developers.
- Allowing developers to sign up, define application clients, test out Web APIs, and subscribe to them.
- Managing access control and authentication of API clients using "API keys" or tokens.

– Usage control and throttling of traffic to specific clients based on a Service Level Agreement (SLA) or other factors.
– Monitoring the usage of specific clients in order to be able to limit access or charge for API usage.

One of the key aims of API Management is a desire to manage these aspects *orthogonally* from the creation of the API itself. This is a major benefit to developers or organizations that wish to expose APIs, because these capabilities can be added in a standard way to their systems without requiring custom development that is specific to the application or business logic.

This is often achieved by the use of a pattern called a "reverse proxy", where the client believes it is connecting to the API itself, but the reverse proxy intercepts these calls, and acts upon them before passing them onto the "target" API. This pattern of infrastructure is also often known as a server-side gateway.

3 Related Work

While there is a great deal of industrial effort and research on Web API management, the academic literature is sparse. In the industrial sector, much of the literature is provided by vendors. However, the report by Forrester [14] provides a good overview. In the academic literature, Raivio et al. [18] explore the business models around Open APIs for the telecommunications industry, and we discuss in [15] the challenges and approaches of managing Web APIs.

In the IoT space, there are a number of efforts around creating open APIs for IoT: for instance, HyperCat [17] is a JSON-based catalogue format for exposing IoT information over the web, developed by a consortium of academic and industrial partners, and ZettaJS [6] is an open source Web API for IoT devices.

There are a number of existing IoT gateways, including [8, 10, 22], that deal with the problem of connecting wireless devices to the wider Internet. They typically bridge multiple low-power devices in a house or factory into a traditional Internet connection. However, our literature search did not identify any server-side gateways/reverse proxies specifically designed for IoT.

We identified two significant gaps in the current literature and existing work in this space. Firstly, most of the work on using APIs with IoT are very limited: there is a common assumption that devices will only communicate with a single API, and there is no discussion of management of these APIs beyond access control. In the access control space, there is a reliance on using outdated models of authentication and authorization (passwords and/or client-side certificates) that are not suitable for device-to-server communication. Two papers address this with token-based authentication schemes: in [13], we addressed the use of OAuth2 with MQTT, and in IOT-OAS [9], Cirani et al. address the use of OAuth2 with CoAP. However, neither of these publications deal with the wider issues around API Management including monitoring, usage control, simplicity of key issuing, developer portals, and monetisation.

Secondly, when looking at the API Management related work, we found no research that addresses how API Management techniques can be used in the face

of IoT specific challenges, especially when using IoT-friendly binary protocols such as MQTT and CoAP. These protocols are important for IoT because of the lower requirements for energy and the lower cost of components required to support them.

4 Challenges for the Internet of Things and Web APIs

There is no accurate number of connected devices, but the best estimates all agree that there are more devices currently than humans on the planet. Cisco forecasts that there will be 50 billion connected devices by 2020 [12].

From our experience working with commercial customers of Web API management software, we find that most Web APIs have tens, hundreds or maybe thousands of known clients. These clients act as machine-to-machine systems where one Web server connects to another Web server.

Most new Web APIs are working with the OAuth2 [11] standard and utilising the "Bearer Token" as the API Key. One of the challenges of moving from a model where the API clients are themselves Web servers to a more diverse model where the clients are devices is that the security of these devices is typically much easier to compromise than the security of Web servers. This problem has become apparent with mobile devices. Mobile application developers must embed the OAuth2 credentials into their mobile apps, and because those mobile devices can be "rooted", these credentials can be stolen. There are solutions to this such as Samsung Knox [20], but these are proprietary and only suitable for high-end devices. This rules them out for many IoT devices.

Therefore we envisage that IoT devices will be more likely to need their own OAuth2 credentials per device. It is impractical to think that these client keys will be issued manually to the IoT devices: this process must be automated. This is enabled by the extension to the OAuth2 specification called Dynamic Client Registration (DCR) [2]. DCR automates the process that a developer would go through on the API portal to gain OAuth2 credentials on behalf of their API client. We therefore intend to use our prototype to explore the use of DCR in IoT scenarios.

We are not aware of any API yet in production where millions of devices each have their own API key, their own set of throttling measures, etc. It can therefore be seen that API management systems will need to evolve to support very large numbers of keys, with millions or even tens of millions of concurrently connected devices.

Another challenge is that IoT devices are often low-powered and reliant on low energy usage. Protocols such as MQTT and CoAP are lower in bandwidth which has a direct effect on energy usage, especially in wireless transmission scenarios. Nicholas [3] shows that MQTT uses considerably less energy that HTTPS in comparative scenarios. This is particularly true in scenarios where notifications need to be sent to devices ("push" scenarios). The traditional way to do this in Web APIs was to require the client to poll the server on a regular basis for updates, which is very expensive in energy and bandwidth usage.

In summary, our work is addressing how to adapt the existing Web API management capabilities to support:

- Large numbers of clients, each with their own credential.
- Devices communicating with public APIs via binary and low-energy protocols such as MQTT and CoAP.
- Usage control, access control, throttling and other API management techniques applied to IoT scenarios.
- How to apply these capabilities orthogonally to existing systems.

5 IGNITE - an API Gateway for IoT protocols

To solve these issues we are building a system that allows using the capabilities of existing API management solutions with IoT protocols. We call the system IGNITE (Intelligent Gateway for Network IoT Events)[1]. Our initial work focuses on the MQTT protocol, but in future we intend to extend this to CoAP.

For our proof-of-concept prototype, we extended three major existing open source projects:

- The WSO2 API Manager [4] project provides the main capabilities for Web API Management including a developer portal, subscription management system, key server, API gateway, access control, throttling, monitoring and analytics system;
- The MITREid-Connect project implements of OAuth2 and OpenID Connect [19] and includes new capabilities such as Dynamic Client Registation and Token Introspection.
- The Mosquitto MQTT broker provides an open source messaging broker for the MQTT protocol.

As of March 2015, upcoming releases of both the WSO2 API Manager and the MITREid-Connect projects plan to support using these two projects in conjunction with each other. Both projects are written in Java.

In conjunction with these projects we have created an API management gateway for IoT — a reverse proxy for IoT protocols that plugs into the existing key server architecture and monitoring capabilities.

We currently have built a first prototype of this gateway in Python and we are porting it to Java to improve performance. Figure 1 shows the overall architecture with the capabilities of the existing projects plus our added capabilities.

The IGNITE component implements the following logic: On a `CONNECT` packet arriving, it extracts the OAuth2 Bearer token from the username field in the packet. It then invokes the Token Introspection service on the MITREid-Connect server to validate the token. If the token is valid, the gateway replaces the token in the request with the userid returned from the introspection call, and forwards the request on to the existing MQTT Broker, which may implement its own validation checks as well. If the token is invalid or no longer active, the IGNITE

[1] The source code is available at `https://github.com/pzfreo/ignite`

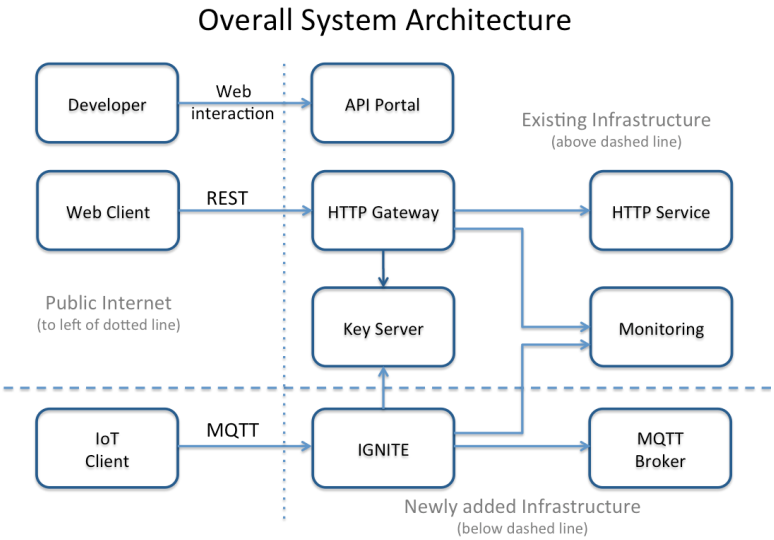


Fig. 1: System architecture

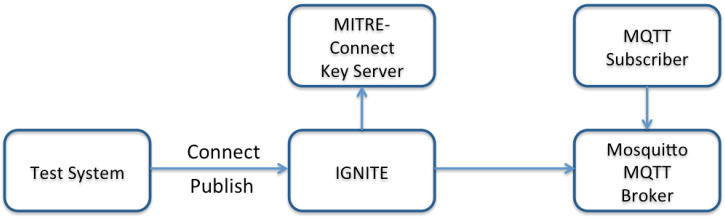


Fig. 2: Test architecture

responds to the client with a packet that indicates that the credential was invalid (a `CONNACK` packet with `ReturnCode=5`).

The monitoring and usage control/throttling aspects of IGNITE are still in development.

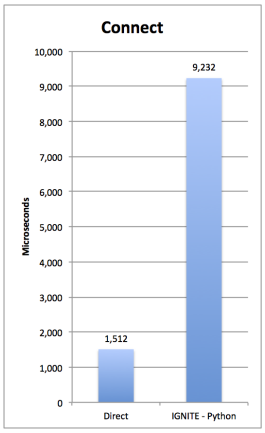
6 Results

To test the system, we evaluated the performance of this system compared to a direct call to the MQTT broker. In this case, the MQTT broker was not running any authentication of its own, so the comparison is not completely like-for-like. Figure 2 shows the architecture of the test set-up.

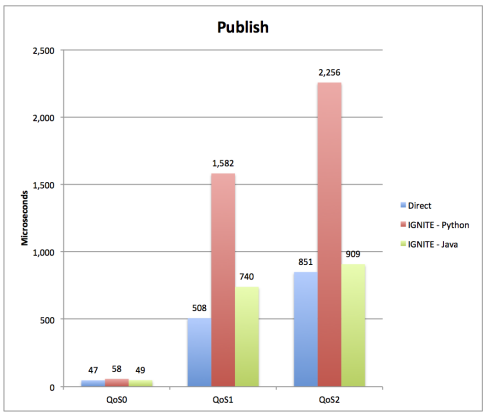
We used the open source Mosquitto [1] broker as the backend of the tests and ensured that there was a subscriber attached so that the messages would require

delivery. For the tests we sampled two flows: A `CONNECT` flow and a `PUBLISH` flow. For `PUBLISH` we tested all three levels of QoS: fire-and-forget (QoS0), at least once (QoS1) and exactly-once (QoS2). QoS1 and QoS2 involve multiple packets transferring between the client and the server.

The tests were all run on a single machine[2] using the localhost networking. The gateway tests include both the more functional Python prototype of IGNITE and an early prototype of the Java version. The tests show the average result over 1,000 `CONNECT`/`CONNACK` messages and 10,000 `PUBLISH` messages, in both cases giving the system time to warm up before capturing timing data. The QoS 1 and 2 tests inherently capture the use of `PUBACK`, `PUBREC`, `PUBREL`, and `PUBCOMP` messages. The focus on connection was because the authentication step during connection is where the most work takes place, and on publication because this is the most used flow in MQTT, as subscriptions are rare compared to publication.



(a) CONNECT



(b) PUBLISH

Fig. 3: Performance Results

The `CONNECT` results are shown in Figure 3a. The results show that the overhead of using the Python IGNITE for `CONNECT` is around 7,700µs per request. Given that this includes a HTTP REST call to the key server this is not unexpected. In the WSO2 API Manager this overhead has been reduced by implementing a binary key validation protocol instead of HTTP. However given that MQTT is a persistent connection compared to existing Web API gateways and HTTP where each request needs to be validated we feel this is a very effective result. We did not (yet) implement caching of token introspection results which could improve this considerably.

The `PUBLISH` numbers (Fig. 3b) show a much lower overhead. For QoS0 the overhead of going through the IGNITE is around 11µs. The QoS2 case has

[2] Mac OS/X 10.10 running on a 3Ghz Intel Core i7 with 16Gb RAM and SSD storage

a significant higher overhead due to considerably more complex message flow. Even in this case the overhead is less than 1500μs and the preliminary data from the Java implementation shows an overhead of less than 60μs. Note that at this stage we have not yet implemented usage control and monitoring into the `PUBLISH` flow so these numbers do not yet reflect the full workload required. On the other hand there is as yet no optimisation of this prototype code.

To put these numbers into perspective, the typical overhead of such a gateway in the HTTP world is around 500μs without implementing any OAuth2 token introspection [7]. In addition, these numbers are all likely to be dwarfed by average internet latencies. For example, the speed of light requires a minimum latency of 40,000μs between the East and West Coasts of the USA, and typical real-world latencies are twice this. Even with prototype code and no optimization, these numbers are respectable and would fit into the tolerance of many existing IoT projects. Therefore we can conclude that this approach is eminently practicable.

7 Conclusions and further work

In this paper we have outlined the challenges around applying the newly emerging area of Web APIs and Web API to connected devices and the Internet of Things. We outline our model of enhancing existing Web API management systems with a new gateway — IGNITE — that focuses on IoT protocols and demonstrates how protocols such as MQTT can be integrated into existing API Management models with some success, in a completely orthogonal manner. In addition, the model of the server-side IoT gateway that we have introduced with IGNITE offers a considerable number of possibilities for managing usage control, access control, monitoring, etc.

We have identified a number of areas for future research. There is work on improving the IGNITE: to support CoAP, to optimize performance; to integrate into the monitoring framework; and to support throttling of traffic. In addition we believe there is considerable scope for research to be done on description languages for using MQTT and CoAP for IoT Web APIs.

Finally we present preliminary data on performance which shows that the overheads of such approaches are reasonable even before optimisation, caching and other techniques are introduced.

8 Acknowledgements

The travel expenses of presenting this research paper were funded by the University of Portsmouth, Faculty of Technology Research Capital Investment Fund (RCIF) number 46175.

References

1. An Open Source MQTT v3.1 Broker, `http://mosquitto.org/`, (Visited on 2013/13/11)

2. Final: OpenID Connect Dynamic Client Registration 1.0 incorporating errata set 1, http://openid.net/specs/openid-connect-registration-1_0.html
3. Power Profiling: HTTPS Long Polling vs. MQTT with SSL, on Android, http://stephendnicholas.com/archives/1217, (Visited on 2013/06/04)
4. WSO2 API Manager - 100% Open Source API Management Platform — WSO2 Inc, http://wso2.com/products/api-manager/
5. Xively by LogMeIn – Business Solutions for the Internet of Things, https://xively.com/
6. Zetta - An API-First Internet of Things (IoT) Platform - Free and Open Source Software, http://www.zettajs.org/
7. Abeyruwan, D.: ESB Performance Round 6.5 — WSO2 Inc. http://wso2.com/library/articles/2013/01/esb-performance-65/#latency, (Visited on 2015/03/24)
8. Chen, H., Jia, X., Li, H.: A brief introduction to IoT gateway. In: IET International Conference on Communication Technology and Application (ICCTA 2011). pp. 610–613 (2011)
9. Cirani, S., Picone, M., Gonizzi, P., Veltri, L., Ferrari, G.: IoT-OAS: An OAuth-based Authorization Service Architecture for Secure Services in IoT Scenarios (2015)
10. Datta, S.K., Bonnet, C., Nikaein, N.: An IoT gateway centric architecture to provide novel M2M services. In: Internet of Things (WF-IoT), 2014 IEEE World Forum on. pp. 514–519. IEEE (2014)
11. (ed), D.H.: The OAuth 2.0 Authorization Framework. RFC 6749, IETF (October 2012), available at http://www.rfc-editor.org/rfc/rfc6749.txt
12. Evans, D.: The internet of things. How the Next Evolution of the Internet is Changing Everything, Whitepaper, Cisco Internet Business Solutions Group (IBSG) (2011)
13. Fremantle, P., Aziz, B., Scott, P., Kopecky, J.: Federated Identity and Access Management for the Internet of Things. In: 3rd International Workshop on the Secure IoT (2014)
14. Heffner, R.: The Forrester Wave^{TM}: API Management Solutions, Q3 2014 (2014)
15. Kopecky, J., Fremantle, P., Boakes, R.: A history and future of Web APIs. Information Technology (2014)
16. Lane, K.: API Evangelist Blog. http://apievangelist.com/blog/, (Visited on 2015/03/24)
17. Lea, R.: HyperCat: an IoT interoperability specification (2013)
18. Raivio, Y., Luukkainen, S., Seppala, S.: Towards Open Telco-Business models of API management providers. In: System Sciences (HICSS), 2011 44th Hawaii International Conference on. pp. 1–11. IEEE (2011)
19. Richer, J., Greenwood, D., Bakis, B.: Componentization of security principles. In: Symposium on Usable Privacy and Security (SOUPS) (2014)
20. Samsung: Mobile Enterprise Security — Samsung KNOX. https://www.samsungknox.com/en, (Visited on 2015/03/24)
21. Williams, A.: 5 Rules For API Management — TechCrunch. http://techcrunch.com/2012/11/11/5-rules-for-api-management/, (Visited on 2015/03/24)
22. Zhu, Q., Wang, R., Chen, Q., Liu, Y., Qin, W.: IoT gateway: Bridging wireless sensor networks into internet of things. In: Embedded and Ubiquitous Computing (EUC), 2010 IEEE/IFIP 8th International Conference on. pp. 347–352. IEEE (2010)