λ-CoAP: An Internet of Things and Cloud Computing Integration based on the Lambda Architecture and CoAP

Manuel Díaz, Cristian Martín, and Bartolomé Rubio

Department of Languages and Computer Science, University of Málaga, Boulevar Louis Pasteur 35, 29071 Málaga, Spain.
{mdr,cmf,tolo}@lcc.uma.es

Abstract. The Internet of Things (IoT) is an emerging technology that is growing continuously thanks to the number of devices deployed and data generated. Nevertheless, an upper layer to abstract the limitations of storing, processing, battery and networking is becoming a mandatory need in this field. Cloud Computing is especially suitable technology that can supplement this field in the limitations mentioned. However, the current platforms are not prepared for querying large amounts of data with arbitrary functions in real-time, which are necessary requirements for real-time systems. This paper presents λ-CoAP architecture, a novel paradigm not introduced yet to the best of our knowledge, which proposes an integration of Cloud Computing and Internet of Things through the Lambda Architecture (LA) and a Constrained Application Protocol (CoAP) middleware. The λ-CoAP architecture has the purpose to query, process and analyze large amounts of IoT data with arbitrary functions in real-time. On the other hand, the CoAP middleware is a lightweight middleware that can be deployed in resource constrained devices and allows the way of the IoT towards the Web of Things. Moreover, the λ-CoAP also contains a set of components with well defined interfaces for querying, managing, and actuating over the system.[1]

Key words: internet of things, lambda architecture, coap middleware, cloud computing

1 Introduction

The Internet of Things (IoT) [1] has received a lot of attention in recent years, thanks to the continuous research in this field, the technological advances and the large amount of systems, components, software and devices release every day. The IoT is composed of a set of interconnected things (mobile devices, RFID tags, sensors, smart-phones, cameras, etc.) which have the ability to sense, communicate and/or actuate over the Internet. Moreover, thanks to the IoT capacity for adapting on many situations, it has carried the IoT suitable for monitoring,

[1] Published in CollaborateCom 2015. DOI: https://doi.org/10.1007/978-3-319-28910-6_18

control and manage in many areas such as Smart Ambient and Monitoring. Nowadays, it is possible to acquire an embedded personal computer for a few dollars [2] and check the free car parks in real-time in your city, something unthinkable just a few years ago. All of this has been originated from the IoT, and it is now one of the most interesting fields of research that will be present to help us everyday.

However, the devices involved in the IoT have serious constraints with respect to processing, storing and networking. These issues has originated integrations with new paradigms like Cloud Computing, known as Cloud of Things [3], which has solved such IoT limitations. Cloud Computing enables an on demand access to a pool of configurable resources providing virtually unlimited capabilities in terms of storage and processing power [4], which are the main drawbacks of the IoT. Even though the integration with Cloud Computing provides the IoT the requirements already mentioned, the current need for extracting knowledge of the data in real-time imply to think beyond. Moreover, there are a large amounts of heterogeneous devices deployed around the world, so they also generate large amounts of data which will keep on growing in the coming years [5]. Cloud Computing can solve the above requirements, but the needs for extracting knowledge with large amounts of data in real-time like in critical systems, means that Cloud Computing might be unsuitable for certain situations due to high latency.

The Lambda Architecture (LA) [6] is a novel paradigm introduced by Nathan Marz composed by Cloud Computing components—a distributed queue, batch and stream processing and a distributed database—designed to offer arbitrary and predicted queries over arbitrary real-time data. The key idea in the LA is the precomputed view, which contains the knowledge consequence of applying functions over the system data. The data in LA is split in two ways: historical data, which contains all data generated in the system, and real-time data with the stream data received. The LA also decomposes the paradigm in three layers: the batch layer, the real-time layer and the serving layer. All data received is delivered to a distributed queue which replies it into the real-time layer for processing and generating the real-time views and the batch layer which stores it in the master data and generates the batch views with the historical data. The LA therefore offers new opportunities for processing and analysis large amounts of arbitrary data in real-time.

Nevertheless, due to the heterogeneity of devices which compose the IoT, which have different functionalities, capabilities and programming languages among other things, an abstraction layer is necessary to abstract that heterogeneity offering a unique way to interconnect anything. Through a middleware, the IoT can hide communication and low-level acquisition aspects in anything [7]. Currently, there are a lot of Web services that work over the Internet for sharing and exchanging a large amount of data and services. So allowing a compatible middleware with the current systems and the Internet, the IoT leads to a part of the Internet, also known as Web of Things (WoT) [8], where the current Web services will be enriched with physical smart objects. Moreover, the Web-enabled applications are available on many platforms with Internet connection,

from computer to smart phone, PDAs and tablets, so the WoT will also enable the IoT on many platforms. However, Web services usually work over protocols such as HTTP and TCP, which are too heavy for resource constrained devices. Taking into account that, the Internet Engineering Task Force (IETF) launched the Constrained Application Protocol (CoAP) [9, 10]. CoAP is a web transfer protocol designed for resource constrained devices that work over UDP. CoAP follows the same style that RESTFul Web services, supporting the Uniform Resource Identifier (URI) and a HTTP model, but it uses UDP thereby reducing the transfer overhead and header of TCP. Furthermore, CoAP also supports asynchronous communication and multicast, not available in TCP and desirable in resource constrained devices with power, battery and network limitations. CoAP was not created as a common middleware, but nevertheless we propose a middleware based on CoAP for providing a layer to abstract communication, low-level acquisition and discovering aspects over the IoT. The IETF also defines the guidelines for HTTP-CoAP mapping, so the CoAP middleware is therefore a promising IoT middleware for integrating the IoT into the Web.

In this paper, we introduce λ-CoAP, an architecture for integrating the IoT and Cloud Computing through the LA and a CoAP middleware, with real-time and mobility requirements and resource embedded devices. The λ-CoAP architecture constitutes a novel paradigm not introduced yet to the best of our knowledge, which has several purposes: bring the IoT to the WoT; improve the IoT with the necessary capacities in terms of storage, processing and networking provided by the LA; and last but not least, define a framework for analysis and actuating over large amounts of real-time data in IoT environments.

The rest of the paper is structured as follows. In Section 2 a state of the art of the components of the architecture is presented. Section 3 presents the general scheme of the proposed architecture. Finally, some conclusions and future work are presented in Section 4.

2 Related Work

For a better organization of the λ-CoAP architecture requirements, related work is divided in three categories taking into account the main requirements proposed in this article. Firstly, we summarize different Cloud Platforms related to those that constitute the LA. Next, several middleware for the IoT are summarized. And lastly, we have summarized related work about the Smart CoAP Gateways.

2.1 Cloud Platforms

Batch Processing The components responsible of generating the batch views in the LA are the batch processing components. They allow the execution of a large amount of jobs without manual intervention, achieving a greater distribution and high throughput. Apache Hadoop [11] is a well-known batch processing platform for storing and processing large amounts of data. Apache Hadoop is

composed by three main components: Hadoop Distributed File System (HDFS), Map Reduce and Yet Another Resource Negotiator (YARN). HDFS is the distributed file system of Apache Hadoop, responsible for replicating and balancing the data over the slaves deployed, achieving high bandwidth, reliability and scalability. On the other hand, Map Reduce and YARN are Hadoop components responsible for processing jobs and managing the resource management respectively. Therefore, Apache Hadoop is a suitable component, compatible with multiple Cloud platforms, for generating batch views in the LA.

Apache Spark [12] is another batch processing platform for processing and analysis large amounts of data. In contrast to Apache Hadoop, Apache Spark does not contain its own file system, but trusts on different data stores such as HDFS, Apache Cassandra, Apache HBase and Amazon S3. Apache Spark follows a different approach than Apache Hadoop. Apache Spark knows the high latency due to reload continuous data from disk, and keeps it in memory for reducing the latency. Although Apache Spark is a desirable component to achieve high performance in a processing system, in the LA the latency requirements are addressed by the real-time layer, so it is not necessary. Moreover, Apache Hadoop presents more integration compatibility with other Cloud platforms.

Real-time Processing Real-time processing components are those located in the real-time layer, and are responsible for processing and analysis the stream data provided by the distributed queue. Apache Storm [14] is an open source distributed system for real-time processing. Apache Storm has an architecture based on directed graphs, where the edges represent the flow data between the vertices deployed, which are the computational components. The design based on directed graphs contribute to provide fault-tolerant and replication in addition to enable several layers to abstract the processing. Moreover, Apache Storm also offers semantics guarantees with 'at least once' and 'at most once' data processing. Also together with the integration with Apache Kafka and several data stores, lead to Apache Storm to be a suitable component for the real-time layer of the LA.

Apache Spark Streaming [15] is also an open source streaming processing component. Apache Spark Streaming does not act like Apache Storm, since it processes the stream data and then stores them in a specific format. Apache Spark Streaming is also integrated with Apache Spark, so they can reuse the same functions and data representation enabling a hybrid architecture with real-time and batch processing. However, due to needs of integration with several components which form the LA, Apache Storm is more suitable for this purpose.

Distributed Databases The serving layer is perhaps the main component of the LA, since it is responsible for serving and merging the real-time and batch views. To support the serving layer a distributed database is necessary for serving and querying data as soon as possible. Apache HBase [16] is an open source distributed database suitable for random access and read/write large amounts of data. Apache HBase uses HDFS as a storage system, where all data is stored in tables like traditional Relational Database Management Systems (RDBMS).

The Apache HBase tables are composed by multiples rows, where each rows can store a set of mixed and indeterminate key-values. Apache HBase also supports fault-tolerant, replication and load-balancing and is properly integrated with Apache Hadoop.

Nonetheless, platforms such as Apache HBase and Apache Hadoop do not guarantee how quickly data can be stored and accessed, which are the main features in the LA. Druid [17, 18] is an open source distributed column-oriented database for storing and querying data in real-time. Druid was created in order to resolve problems about applications which require low latency on ingestion and query data. The key idea of Druid is to persist data on memory for quick access. The historical data is persisted and obtained from the deep storage when real-time data changes to historical. Moreover, Druid has integrations with external Cloud platforms such as Apache Kafka [13], Apache Storm and Apache Hadoop, so the serving layer will be properly constituted with Druid.

2.2 IoT Middleware

The IoT middleware contributes to abstract the processes of communication, low-level acquisition and discovering in the IoT devices. Furthermore, it also keeps away the heterogeneity in the IoT devices, showing each device like a unified middleware.

LooCI [19] is a middleware for building component-based applications in Wireless Sensor Network (WSN). The component infrastructure allows mechanisms for deploying and managing components in runtime. LooCI also provides interoperability across various platforms: a Java micro edition for constrained devices known as Squawk, the OS for the IoT Contiki, and the OSGi.

On the other hand, the OMG Data-Distribution Service for Real-Time Systems (DDS) [20] is a centric publish/subscribe middleware for real-time and embedded systems. All data in DDS is exchanged between consumers and producers through typed data topics providing a level of safety.

However, the latter middlewares do not focus on offering interoperability with the Internet, and its system requirements can be too heavy for resource constrained devices, in contrast to the header of the CoAP middleware that only requires 4 bytes. In addition, the real-time requirement is only addressed by DDS, but its protocol stack can be too heavy for embedded devices too. Therefore, the CoAP middleware is a lightweight middleware suitable for embedded devices with real-time requirements for the λ-CoAP architecture.

2.3 Smart CoAP Gateway

For a HTTP-CoAP mapping, the integration between the CoAP devices and the LA, besides minimizing the overload in constrained devices, it is necessary to have a smart gateway that acts as an intermediate among the Web, the LA and CoAP devices. The Smart Gateway acts as a cross-layer proxy between HTTP and CoAP devices, creating a barrier to protect the embedded devices in case of

an overload of requests. In addition, the Smart Gateway is also responsible for transmitting the IoT data generated by the CoAP devices for further storing, processing and analysis. Furthermore, through the Smart Gateway, the great number of web applications deployed could make use of the physical devices deployed achieving a cyber-physical environment.

Ludovici et al. [21] present a HTTP-CoAP proxy to leverage the interconnection of 6LowPAN WSN with HTTP long-polling requests. Ludovici et al. know of the performance loss while doing continuous HTTP requests, so a WebSocket communication is proposed to improve it. In [22], a proxy is presented to interconnect a ZigBee—a wireless communication protocol for embedded devices—CoAP WSN with HTTP in order to enable an UPnP system. However, these works focus on proxies with specific communications for embedded devices such as 6LowPAN and ZigBee, and are not focus on integration with Cloud Computing components. Our Smart Gateway aims to abstract different communications (Ethernet and ZigBee) in order to build a Smart Gateway suitable for WSN as for mobile devices like smart-phones.

3 The λ-CoAP Architecture

The λ-CoAP is a novel architecture designed to offer a general framework for processing, storing, analysis and obtaining conclusions of IoT data in real-time. The λ-CoAP is the result of combining current components of Cloud Computing for abstracting the limitations of the IoT devices, with an IoT middleware suitable for resource constraint devices. The λ-CoAP architecture also includes the adoption of smart gateways, which has been established in order to abstract and protect the IoT devices from external sources, but also offers a new way to integrate the IoT in the WoT. Furthermore, the λ-CoAP also presents mechanisms for querying data and generating actions over the IoT as well as a Web UI where the users registered will can query and manage the system.

Fig. 1 shows the general architecture of the λ-CoAP. It is composed by three differentiated components: a Cloud, the Smart Gateways and end devices. The Cloud is responsible for allocating and serving the components of the LA—Apache Hadoop, Apache Storm, Apache Kafka and Druid—and other components which compose the architecture: the Actions Component, a Web UI, a Querying Interface and optional Smart Gateways. The Smart Gateways can be deployed locally for abstracting local IoT deployments like a smart-house, in addition to cloud deployment for mobile devices. Lastly, the end devices incorporate a middleware based on CoAP whose main function is to abstract the communications with the Smart Gateways for querying data or actuation over them as well as discovering and low-level acquisition aspects. The following sections describe each component belonging to the λ-CoAP architecture.

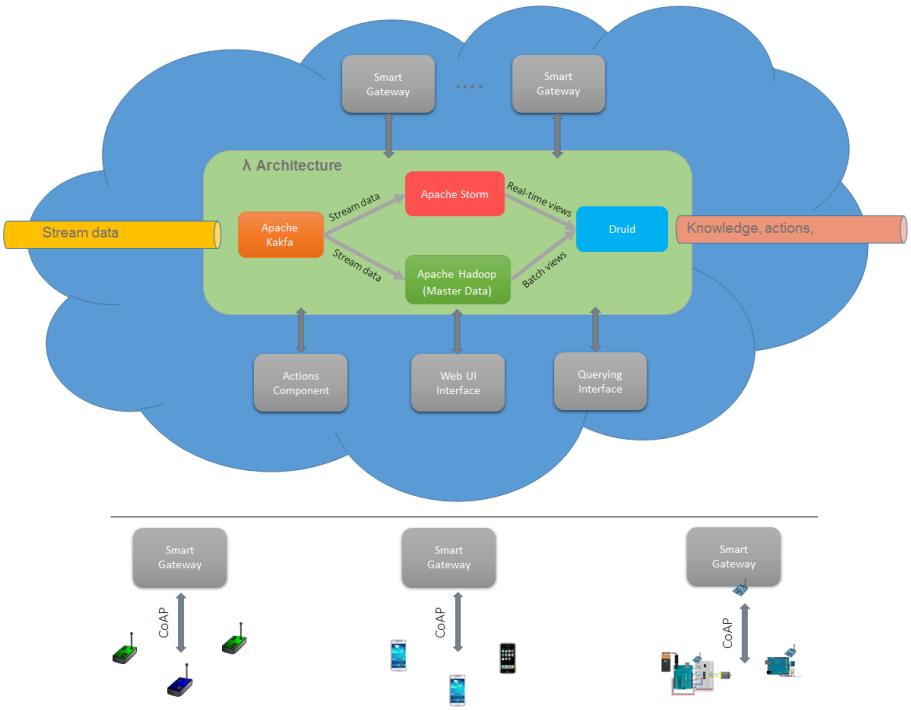

Fig. 1. The λ-CoAP Architecture

3.1 Lambda Architecture

As introduced, the LA is a paradigm composed by several Cloud Computing components in order to enable a framework for processing, analysis, storing and querying arbitrary data with arbitrary functions in real-time. Immutable data—data that does not change as time passes, only can be queried or added—besides the precomputed views are key concepts in the LA.

All data received in the LA is stored in the master data and it is converted into another form during the processing, so the initial data received never changes over time. The latter prevents human errors such as accidental deletions and updates. Also, Cloud Computing LA components replicate the data and offer mechanisms to high availability and fault-tolerant, so the probability of data lost is small. Generating arbitrary functions and complex analysis over large amounts of data may require a considerable latency, an issue for real-time systems. Precomputed views have been designed to have available the complex analysis required besides extracting knowledge over the system data. The LA is decomposed in the following layers for such purposes:

– **Batch layer**: responsible for processing the historical data and generating the batch precomputed views through batch processing. The batch layer also

stores all data of the system in an immutable form, known as master data, so it allows generating new precomputed views with new functions at any time. The batch layer is composed by the batch processing platform Apache Hadoop.
- **Real-time layer**: makes up the high latency of the batch layer, processing the stream data and generating real-time precomputed views. The real-time layer is composed by the real-time processing platform Apache Storm.
- **Serving layer**: responsible for abstracting the batch and real-time views offering a unique way to access them. The serving layer is composed by the distributed database Druid.

When a stream data is received in the LA through the Smart Gateways, the distributed queue composed by Apache Kafka distributes the stream data to the batch and real-time layers. The batch layer stores the stream data received by Apache Kafka and generates the batch views with the historical data periodically. The batch layer makes complex analysis over the historical data and generates actions for the IoT about such analysis. On the other hand, the real-time layer generates the real-time views and actions with the stream data received. All actions both the generated by the real-time layer and the batch layer are sent to the Actions Component which is responsible to communicate with the smart gateways in order to actuate over the underlying IoT devices. Lastly, the serving layer merge the batch and real-time views for the coming queries.

3.2 Smart Gateway

The Smart Gateway is the component responsible for abstracting the IoT in addition to connect it with the LA. The Smart Gateway is also responsible for incorporating a cross-layer proxy to interconnect the IoT with the Web. There are two ways of Smart Gateway deployment: a cloud and local deployment. The local deployment is intent to abstract local deployments like WSN or smart-home. On the other hand, the cloud deployment is intent to integrate mobile devices without a local Smart Gateway. The local Smart Gateways uses the ZigBee protocol and Ethernet/Wifi/UMTS for devices which support it, for the communication with the IoT devices. The cloud deployments trust only on the Ethernet/Wifi/UMTS module installed in each component, so a ZigBee network is not available. Moreover, the adoption of a lightweight protocol for resource constrained devices like ZigBee, which does not trust in UDP, has only been done to the best of our knowledge in a UPnP system in [22]. Each Smart Gateway is composed by the following components:

- **LA Connector**: connects the Smart Gateway with the LA. When a stream provided by the Smart Gateway is received, the LA Connector sends it to Apache Kafka for further processing of the LA.
- **Web Server**: is responsible for managing the HTTP requests of external sources such as an action provided of the LA and a query of a client. The Web Server checks firstly on the Database Cache if the data required is available. If

the data required is not present in the Database Cache, the Web Server obtains the address of the device associated to the request, and makes a CoAP request with it to obtain or act over the device that is sent to the CoAP Connector.

– **CoAP Connector**: receives requests from the Web Server to request data or actuate over a device. The CoAP Connector also connects the CoAP middlewares with the Smart Gateways to obtain the data or send the actions requested.
– **Database Cache**: has two main functions, in fact, it is both a cache and a lightweight database. On the one hand, the Database Cache stores information about end devices and its resources. On the other hand, the Database Cache also stores data of devices in order to provide a cache and protects the end devices of overload of requests.

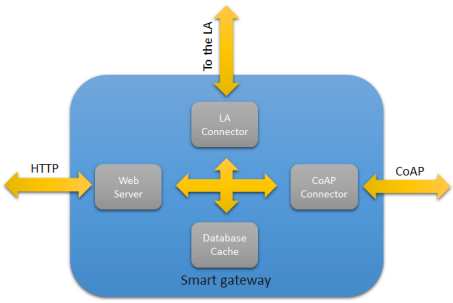

Fig. 2. Smart Gateway

Fig. 2 shows the architecture of the Smart Gateway. When a CoAP middleware is deployed in the system, it sends a broadcast message in local deployments in order to inform the rest of the network that is deployed. The Smart Gateway, upon the reception of the broadcast message, registers the device on its database and makes a request to the CoAP Connector for obtaining its resources provided. Once the Smart Gateway receives the requested message, by default the Smart Gateway subscribes to all resources contained in the CoAP middleware through the observe CoAP option. The Smart Gateway also stores information about CoAP resources in order to inform the HTTP clients with the resources provided by the CoAP middleware and translate operations. The CoAP middleware also sends broadcast messages periodically in order to inform the other components that are alive. In cloud deployments, there are not broadcast messages, but the end devices also inform to the Cloud Smart Gateway that they are deployed/alive. Every time that an observed CoAP data is received in the CoAP connector, it is transferred to the LA Connector for sending it to the LA.

3.3 CoAP Middleware

The CoAP middleware is responsible for abstracting communication, low-level acquisition and discovering aspects in the IoT devices. The CoAP middleware provides an abstract layer for accessing low-level components like sensors as well as actuating over the actuators components in the IoT devices. It depends on the networking established, the CoAP middleware can be installed in a WSN through the ZigBee protocol besides to be installed in mobiles devices through Ethernet/Wifi/UMTS.

The CoAP middleware also manages CoAP resources which can be identified by an URI. CoAP resources are components responsible for the abstract operations in the IoT devices such as obtaining the value of a specific sensor and performing the action over an actuator component. Querying the underlying sensors installed in each device is performed through a resource with CoAP GET requests, whereas actuating over a components is performed through a resource with CoAP POST requests. The CoAP requests are generated by the CoAP Connector of the Smart Gateways based on the HTTP requests. Therefore, through the Smart Gateways, users can query and actuate over the IoT by means of HTTP GET and POST operations.

The discovering process is initiated when a new CoAP middleware is deployed in the system. The discovering process has been designed to help the Smart Gateways to know how many CoAP middlewares are deployed, which are alive and which are its resources. Moreover, the discovering process also informs the rest of the network the status of the middleware, allowing it to create distributed CoAP resources among several CoAP middlewares.

3.4 External Components

In order to provide a unified way to actuate, query and manage the system, the λ-CoAP also incorporates the following components:

- **Cloud Smart Gateways**: are Smart Gateways deployed in the Cloud which manage and abstract the underlying IoT mobiles devices. Cloud Smart Gateways can be deployed based on the system needs, so there may be from 0 to n deployments.
- **Actions Component**: provides an API RESTFul for receiving the result actions generated by the LA. The Actions Component establishes the communications with the respective Smart Gateway in order to actuate over the action required CoAP resource.
- **Web UI interface**: provides a Web UI where the users registered can manage, query and monitor the λ-CoAP architecture.
- **Querying Interface**: provides an API RESTFul to query all data and conclusions belong to the LA.

The adoption of API RESTFul interfaces also allows that external systems can integrate the λ-CoAP architecture in their business models.

4 Conclusions and Future Work

This work presents a novel architecture to integrate the Internet of Things and Cloud Computing with real-time requirements, called the λ-CoAP architecture. The λ-CoAP architecture contributes to solve the problems of processing, analysis and storing large amounts of IoT data with arbitrary functions through a paradigm known as the LA. The λ-CoAP architecture also supports the variety, velocity and volume of data generated by the IoT, allowing the known Big Data paradigm.

The λ-CoAP architecture proposes a CoAP middleware in order to abstract the low-level aspects of the IoT and provides a lightweight way to actuate and send IoT data. Moreover, the IoT is abstracted through Smart Gateways, which provide a cross-proxy between HTTP and CoAP, allowing the access to the IoT through normal HTTP requests. In fact, the latter also contributes to carry the IoT towards the WoT. The Smart Gateways also take into account the mobility of the IoT, and supports mobile devices as local deployments. Finally, the architecture is composed of external components for querying, actuating and managing the system through well defined interfaces.

Therefore, we think that the λ-CoAP architecture is a promising architecture for the IoT with real-time and mobility requirements, that can be applied in many areas both with smart-phone networks and with WSNs. We are currently involved in the development of the architecture in order to evaluate the performance and real-time requirements on an IoT deployment.

Acknowledgment

This work was funded by the Spanish projects TIC-1572 ("MIsTIca: Critical Infrastructures Monitoring based on Wireless Technologies") and TIN2014-52034-R ("An MDE Framework for the Design and Integration of Critical Infrastructure Management Systems").

Author Contributions

All authors contributed to the conception of the proposal, defined the methodology and contributed to writing and revising the article.

Conflicts of Interest

The authors declare that there is no conflict of interest regarding the publication of this manuscript.

References

1. Ashton, K.: That 'internet of things' thing. J. RFID. 22, 97–114 (2009)
2. C.H.I.P.: The World's First 9 dollars Computer. http://nextthing.co/
3. Aazam, M., Khan, I., Alsaffar, A. A., Huh, E. N.: Cloud of Things: Integrating Internet of Things and cloud computing and the issues involved. In: 11th International Bhurban Conference on Applied Sciences and Technology (IBCAST), pp. 414–419. IEEE, Anchorage, Alaska USA (2014)
4. Botta, A., de Donato, W., Persico, V., Pescap, A.: On the Integration of Cloud Computing and Internet of Things. In: 2nd International Conference on Future Internet of Things and Cloud (FiCloud-2014), pp. 23–30. IEEE, Barcelona Spain (2014)
5. Zaslavsky, A., Perera, C., Georgakopoulos, D.: Sensing as a Service and Big Data. In: International Conference on Advances in Cloud Computing (ACC-2012), pp. 21–29. Bangalore India (2012)
6. Marz, N., Warren, J.: Big Data: Principles and best practices of scalable realtime data systems. Manning Publications Co., (2015)
7. Calbimonte, J. P., Sarni, S., Eberle, J., Aberer, K.: XGSN: An Open-source Semantic Sensing Middleware for the Web of Things. In: 7th International Workshop on Semantic Sensor Networks (SSN2014), 13th International Semantic Web COnference. Riva del Garda, Trentino Italy (2014)
8. Zeng, D., Guo, S., Cheng, Z.: The web of things: A survey. J. of Communications. 6, 424–438 (2011)
9. Bormann, C., Castellani, A. P., Shelby, Z: Coap: An application protocol for billions of tiny internet nodes. J. Internet Computing. 16, 62–67 (2012)
10. Shelby, Z., Klaus, H., Carsten, B.: The Constrained Application Protocol (CoAP). https://tools.ietf.org/html/rfc7252/
11. Shvachko, K., Kuang, H., Radia, S., Chansler, R.: The hadoop distributed file system. In: 26th Symposium on Mass Storage Systems and Technologies (MSST), pp. 1–10. IEEE, Incline Villiage, Nevada USA (2010)
12. Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., Stoica, I.: Spark: cluster computing with working sets. In: 2nd USENIX conference on Hot topics in cloud computing, p. 10. Boston, MA USA (2010)
13. Apache Kafka. http://kafka.apache.org/
14. Toshniwal, A., Taneja, S., Shukla, A., Ramasamy, K., Patel, J. M., Kulkarni, S., ... , Ryaboy, D.: Storm@ twitter. In: 2014 ACM SIGMOD international conference on Management of data, pp. 147–156. ACM, Snowbird, Utah USA (2014)
15. Apache Spark Streaming https://spark.apache.org/streaming/
16. Apache HBase. http://hbase.apache.org/
17. Yang, F., Tschetter, E., Léauté, X., Ray, N., Merlino, G., Ganguli, D.: Druid: a real-time analytical data store. In: 2014 ACM SIGMOD international conference on Management of data, pp. 157–168. ACM, Snowbird, Utah USA (2014)
18. Druid. http://druid.io/
19. Hughes, D., Man, K. L., Shen, Z., Kim, K. K.: A loosely-coupled binding model for Wireless Sensor Networks. In: SoC Design Conference (ISOCC), 2012 International, pp. 273–276. IEEE, 2012. Jeju Island, Korea (South) (2012)
20. The OMG Data-Distribution Service for Real-Time Systems (DDS). http://portals.omg.org/dds/
21. Ludovici, A., Calveras, A.: A Proxy Design to Leverage the Interconnection of CoAP Wireless Sensor Networks with Web Applications. J. Sensors. 15, 1217–1244 (2015)

22. Mitsugi, J., Yonemura, S., Hada, H., Inaba, T.: Bridging upnp and zigbee with coap: protocol and its performance evaluation. In: Proceedings of the workshop on Internet of Things and Service Platforms, pp. 1. ACM, Tokyo Japan (2011)