future internet

MDPI

Article

An Integrated Platform for the Internet of Things Based on an Open Source Ecosystem

YangQun Li

College of Internet of Things, Nanjing University of Posts and Telecommunications, Nanjing 210003, China; yqli@njupt.edu.cn; Tel.: +86-025-8349-2013

Received: 21 September 2018; Accepted: 30 October 2018; Published: 31 October 2018

Abstract: The Internet of Things (IoT) is increasingly part of daily life. However, the development of IoT applications still faces many problems, such as heterogeneity, complex management, and other difficulties. In this paper, first, the open source technologies of IoT are surveyed. We compare these technologies from the point of view of different levels of technical requirements, such as device management, data management, communication, intelligent data processing, security and privacy protection; we also look at requirements of application development and deployment. Second, an IoT integrated development platform architecture for IoT applications based on open source ecosystem is proposed and evaluated in an industrial setting. We applied P2P technology to distributed resource management and blockchain-based smart contract mechanics for resource billing management. The results show that the IoT gateway based on an open source ecosystem had a stable and reliable system performance with a certain data size and concurrency scale. These conditions satisfy the application requirements of the IoT in most sensing environments.

Keywords: IoT middleware; open source ecosystem; industrial IoT; integrated development platform; performance evaluation

1. Introduction

With the development of hardware and software technologies, Internet of Things (IoT) technology has extended intelligence from ordinary computers into people's daily lives. In IoT, many sensors transmit monitoring data through the network to a Cloud Computing platform for storage and intelligent processing. According to the processing results, the needs of people are met by controlling and changing their environment.

In 2014, ITU-T released a general requirement proposal for IoT applications [1]. Shen [2] summarized the general requirements of IoT technologies and the specific functions of each type of requirement. These common requirements include non-functional, application support, service, communication, device data management and security and privacy protection requirements. Using this framework, Section 2 presents further analysis of the actual requirements faced by each layer in the architecture of the IoT.

To understand and satisfy IoT application real requirements, several IoT research projects have been conducted. The WoT (Web of Things) test bed WoTT is proposed in [3], where the goal is to provide a stable, open, flexible, and secure infrastructure that simplified the design and development of applications. The WoTT test bed is based on a standard protocol that facilitates development and deployment. The specific objectives include hiding details of the underlying implementation, strengthening network self-configuration management (to reduce manual interventions), transparently and simultaneously managing multiple protocols and multiple platforms, and providing a platform for designing and testing task interactions. The IoT-LAB [4] technology mainly tests the link layer protocol performance, such as resource consumption and packet success rate. SmartSantander [5,6]

provides a large-scale test bed with approximately 20,000 nodes deployed in multiple cities in Europe. It focuses on smart city services and environmental data collection.

The functional and non-functional requirements of IoT middleware are given in [7]. Various commercial and open source middleware considering application support, application development, and device connection and management are compared. Based on this analysis, an IoT middleware reference model is introduced. By analyzing the characteristics of IoT architecture and applications, the functional and non-functional requirements of IoT middleware technology are also given in [8]. The review summarizes the IoT middleware platforms from the perspectives of resource discovery and management, data management, event management, and code management. However, these platforms mainly focus on wireless sensor networks and were developed early on by scientific research institutions. Palade et al. [9] studied the evaluation criteria of IoT middleware and used the analytic hierarchy process (AHP) method to determine the weights of different standards. They then analyzed and compared the middleware performance in service registration and service composition. Finally, they compared the application development process of four middleware through four different scenarios.

Claudio S. et al. [10] surveyed and compared the most relevant autonomic and cognitive architectures for the IoT. They investigated those IoT architectures with self-management and cognitive capabilities for managing many devices and different applications. These architectures try to minimize human intervention and shield the devices' heterogeneity, which are interesting research topics and worthy of deeper study in the future. Giancarlo F. et al. [11] proposed an INTER-IoT approach for layer interoperability between different IoT domains, including INTER-IoT, INTER-FW, and INTER-Meth. INTER-IoT provides interoperability of different layers, such as the device layer, networking layer, middleware layers, and services layer. INTER-FW is a framework for IoT application development. INTER-Meth provides an engineering methodology based on CASE tools for IoT application integration. The method shows the concept model fulfilled the interoperability requirement between domains in two use cases, but does not explain how it was implemented. Claudio S. et al. [12] further described both the abstract and instantiated process schema of INTER-Meth methodology, and provided a reference architecture with different layer interoperability.

Compared with the experimental platforms [3–6], the proposed platform is based on micro-service architecture. It focuses on integrated application development platform of the IoT through an open source system that establishes an end-to-end integrated application development and deployment environment for IoT. Compared with related research works in [10–12], here, we propose a novel integrated application development architecture for different IoT platforms from a practical point of view. We also applied WoT technology—such as data description, data transfer, and semantic data processing—to improve the interoperability of different IoT applications.

Although there are many technologies and experimental platforms available, they do not consider the requirements of IoT standardization and real scenario. To solve these problems, we built an end-to-end platform that meets the complex requirements of real-world scenarios and the standardization requirements. The platform supports heterogeneous devices, a variety of application scenarios and can realize multi-source data intelligent processing and personalized IoT service development.

The contributions of this research are as follows:

1. We studied and compared open source IoT technologies. The IoT-related requirements for different levels of IoT architecture were analyzed. For example, open source middleware platforms were analyzed and compared from different aspects, such as heterogeneous device access, device management, data transmission, intelligent data processing, system security, cloud platform integration, API style, application support, and activeness.
2. By comparing the open source systems listed with the various requirements of different layer, the requirements implemented and not yet implemented by open source systems were analyzed.

3. A cloud–fog collaboratively integrated development platform for IoT is given. Some technologies are adopted to implement requirements that have not yet been solved by open source systems. For example, P2P is used for distributed resource management of IoT and blockchain-based smart contract is applied for resource billing management.
4. An industrial IoT scenario was used for evaluation and the preliminary result shows the feasibility of the platform.

The rest of the paper is organized as follows. In Section 2, the existing IoT open source technologies are comprehensively reviewed and compared. In Section 3, integrated application development platform architecture for the IoT is presented according to the characteristics of IoT applications. Section 4 gives the specific design of the development platform, which is evaluated in an industrial IoT application scenario. Finally, our conclusions and future work are provided.

2. Open Source Ecosystem for the IoT

At present, the overall architecture of IoT technology consists of three layers: the device layer, network layer, and the service support and application layer.

For the device layer, it is necessary to solve the problem of how and when the device is accessed by the IoT. These problems are as follows:

1. Device access and state management: Access management refers to which protocols the device layer supports for devices to access the network. State management means how to manage the life cycle of the device, i.e., device access, data-read and device control, status maintenance, and device failure management, which provides status information for the upper application.
2. Device identification: Many devices need to be distinguished by identifiers. Does the device ID need to include location information? How can the relative positional relationship between devices be reflected? For example, a hierarchical resource structure corresponding to a geographical location can be applied for managing device resources in an IoT application environment. However, this implementation requires manually associating relative relationships between resources. If the identifier can automatically include the relative geographical position relationship, it can automate the hierarchical management of resources. When a device accesses the network, can the device identifier be generated automatically? How can we locate smart things based on their device IDs?
3. Data description: Different devices have different data description methods, which are text-based or semantic-based. Therefore, the device layer needs to flexibly support and process commonly used data description methods, and be able to perform functions such as data format conversion and semantic reasoning for the different data descriptions.

The network layer is responsible for the network communication and connection management between devices and the remote server. The data transmission protocol and the networking mode can be determined from the application. For example, in an IoT environment, there are many sensing nodes with limited resources, which intermittently transmit few data. This kinds of new requirements are imposed on the network transmission mechanism in a large-scale network environment.

The service support layer provides developers with service development platforms and basic platform functions. These basic functions may include:

1. Service discovery: It should be easy for developers to discover the service capabilities of the device.
2. Resource management: This function monitors the status of resources. When the resource fails, or when the resource is about to be invalid due to low battery levels, it selects alternative resources to ensure that the application has a certain degree of robustness.
3. Security mechanism: They provide a variety of efficient security mechanisms to ensure the safety and reliability of data transmission, storage, and processing.

4. Intelligent data processing and personalized service provisioning functions: Intelligent algorithms are applied to multi-source sensing data for intelligently inferring and gaining new knowledge. It can also provide users with personalized services by using sensors to obtain contextual information.

With the development of IoT technology, the combination of many open source technologies constitutes an open source ecosystem for the IoT, with each part implementing different functions of IoT architecture. These functions range from data sensing to applications for end users. This section analyzes the existing IoT open source technologies which can satisfy the different layer requirements. We also mention some IoT requirements that have not been implemented by the open source ecosystems thus far.

2.1. Open Source IoT Technology for Device Access

Currently, device layer requirements implemented by open source technologies mainly include requirements related to the device connection and the device's immediate state management. Specific technologies include the following.

2.1.1. Short-Distance Communication

1. Bluetooth technology [13–17]: This has been developed and applied for many years, and there are many open source Bluetooth platforms. For example, Bluez technology provides a complete Bluetooth system which supports a variety of CPUs, Linux and Android operating systems. Android BLE built into Android implements Bluetooth device access management. Gatt, based on the Go language, is an open source software that implements the GATT framework. Noble implemented the central module of Bluetooth by using Node.js and Bloon implemented a Bluetooth peripheral module based on Node.js.
2. NFC (Near Field Communication) [18]: There are many open source systems that support NFC. For example, Libnfc provides platform-independent, low level NFC SDK, and supports event-based device interaction mechanisms. NFC device read capabilities for the Android platform are also available.

2.1.2. Long-Distance Communication

1. LoRa/LoRaWAN [19,20]: This technology is used for low-power long-distance communication in the IoT. LoRAWAN is the long-range communication protocol and architecture, and LoRA is the physical layer. They use a long-distance star structure to achieve communication between nodes and servers, enabling sensor data acquisition. LoRAWAN open source technologies include Lora-gate-bridge, Loraserver, Lora-app-server, packet forwarder, and the things network. Lora-gate-bridge provides a service that converts UDP packet to JSON or MQTT format. Loraserver provides Lora network server functions, which are responsible for uplink data reception and scheduling downlink data transmission. Lora-app-server, as an application server, mainly provides the LoRAWAN infrastructure node inventory, handling the received uplink application payload and downlink payload queue. It also comes with a web GUI for device management and provides the developer with a RESTful, JSON and gRPC interface. LoRAWAN also supports publishing the received payload to an MQTT broker. The Lora packet forwarder runs on the Lora gateway to convert between RF signals and IP/UDP. The things network provides Cloud Platform for accessing LoRa devices. Currently, many LoRa manufacturers use the things network to provide services where there are more than a thousand gateway devices in operation. The Lora server supports Linux, OS X, and Microsoft Windows operating systems. The Lora-gate-bridge, Lora packet forwarder, and Lora-app-server support Linux operating systems. Together, these open source software systems build a data acquisition, transmission, and application development platform for the IoT, as shown in Figure 1.

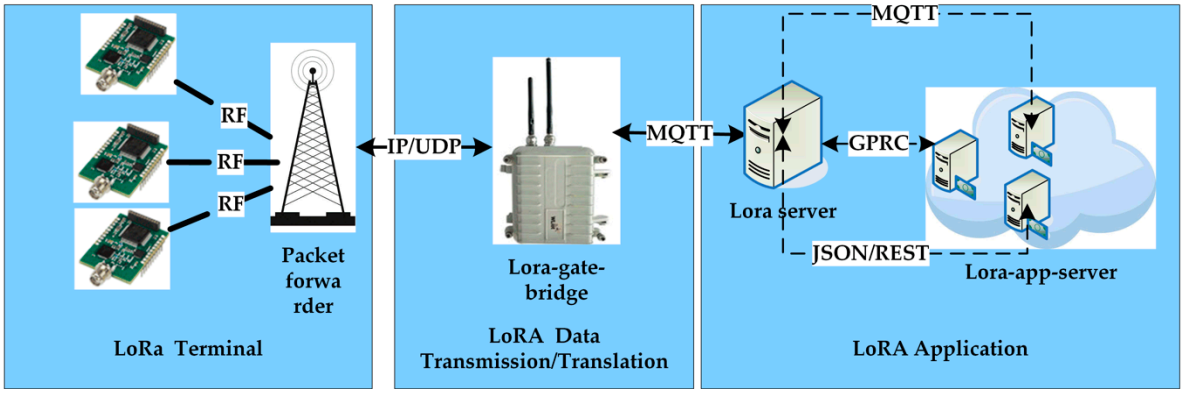


Figure 1. LoRA architecture.

2. SIGFOX open source technology: This is an IoT access technology in competition with LoRA. It is a business-operated protocol with features such as simple usage, low power consumption, low cost, and short messages. It cannot create a private network and the user can only use the "connect as a service" (Caas), which is provided by the SIGFOX operator. Currently, the open source technology related to SIGFOX is the Sigfox Platform, which was developed and maintained independently of SIGFOX as a visualization cloud platform for SIGGOX terminal device access [21].
3. Narrowband-IoT (NB-IoT) open source technology: The NB-IoT technology uses the operator's base station as a gateway for data access and transmission. Currently, searching on the GitHub.com website using the keyword "NB-IoT" shows that there are open source platforms implementing the NB-IoT development framework for terminal modules (including open source hardware platforms) and cloud platforms. Relatively complete and highly active frameworks include EasyIoT [22] and Huawei's CodeLab [23].

2.1.3. Automatic Access of Devices

At present, automatic access requirements for devices are mainly fulfilled through the device's auto-discovery mechanism. Open source technologies in this space include mDNS, Bonjour, uPnP, etc. mDNS uses multicast technology to implement the service discovery in the LAN and also supports P2P service discovery, which was released as a draft RFC in 2013. uPnP also uses multicast technology to implement communication between devices. It was developed earlier and is more mature which was released as the ISO/IEC 29341 standard in 2008 [24].

2.2. Data Management Requirements

IoT applications generate many data, and hence require appropriate mechanisms for efficient data transmission and data integration. These mechanisms include data representation, data visualization, resource descriptions, and semantic processing, as detailed in the following.

2.2.1. Data Representation

At present, there are mainly three mechanisms for data transmission on resource constrained devices: CBOR, FlatBuffers, and protocol buffers [25–28]. CBOR (Concise Binary Object Representation) is a data format with good scalability that minimizes the size of the code, has a relatively small message size and no version negotiation. The current draft is RFC 7049. TinyCBOR is a C/C++ based CBOR implementation from Intel that meets industrial standards. FlatBuffers, originally developed by Google for games and other performance-critical applications, is an efficient cross-platform serialization library that supports languages such as C++, C#, C, Go, Java, JavaScript, PHP, and Python.

Protocol buffers is a language- and platform-neutral, extensible XML structured data serialization method developed by Google, but it is smaller, faster, and simpler. It is easy to use, and can define customized structured data and automatically generate the read/write method for the structured data. Open source implementations of this mechanism support a variety of platforms and languages.

From the above three data description mechanisms, it can be seen that the goals of these methods are to reduce the number of data transmissions, reduce the resource consumption of data processing, and simplify the data processing process to meet the requirements of IoT applications.

2.2.2. Data Semantic Processing

We found 28 open source projects through a search on GitHub.com using the keyword "Semantic IoT". After removing some invalid results, they were classified according to their main functions and are compared in Table 1.

These open source semantic platforms are the product of academic research and most of them lack the support of a specific team or company. Therefore, there is a lack of an open source IoT semantic data processing platform that is relatively complete and mature.

Table 1. Open source system comparison of semantic data processing.

	Product	Function	Language	Licenses	Update	Maturity
Semantic description	iotdb-vocabulary [29] iotdb-models [30]	Describe the semantic model of the items such as sensors and controllers. The model is relatively simple.	Python JS	GPL-3.0 Apache 2.0	2016 2016	Personal development
Semantic framework	M3Framework [31]	Provides semantic processing framework such as M3/SWoT/SSN/S-LOR/LOV4IOT etc.	Web Ontology	GPL-3.0	2016	Professional academical team and some companies
Semantic interface	SemanticInterface [32]	Provide semantic interface for service discovery	Java	MIT	2017	Personal development
Semantic discovery	fiware-iot-discovery-sr [33]	Semantic registration of IoT resources, entities and service descriptions	Java	/	2016	University of Surrey
IoT data semantic annotation	semantic_Annotator [34]	Semantic data annotation of one M2M resources, currently supporting Smart Park scenarios	JS	/	2018	Personal development
Semantic middleware for IoT	aura-middleware [35]	Facilitates the development of IoT applications and shields the differences between sensors so that they can work together	Python	GPL-3.0	2016	Personal development

2.2.3. Resource Description

Resource descriptions provide information about physical entities or service resources to help service users. The HyperCat specification [36] is an open source, lightweight, JSON-based IoT resource description mechanism that implements mapping between physical entities and virtual resources. It provides the metadata description of resource objects, description of relationships between objects, CRUD-based API interfaces, and support for publish/subscribe and event flows.

2.3. Application Layer Transmission Requirements

This section concerns data transmission requirements between IoT devices or between the devices and the cloud platform. Current application-layer data transmission protocols include Modbus, MQTT (Message Queuing Telemetry Transport), CoAP (Constrained Application Protocol), OPC-UA (OPC Unified Architecture), HTTP protocols, etc. Here, we introduce the first four mentioned protocols.

2.3.1. Modbus Protocol

Modbus technology [37] is an application layer protocol that implements client/server communication between devices connected to different buses or networks, enabling communication between multiple devices. The protocol is widely used in industrial environments and can operate on networks such as TCP, Ethernet, Serial, and RS-485. At present, there are many sensors using the Modbus protocol. Open source software that supports the Modbus protocol includes a variety of development language versions, such as Java, C, Node.js, Python, and Go.

2.3.2. MQTT Protocol

MQTT is a lightweight publish/subscribe messaging protocol that can be used for M2M IoT connections. It is used for communication via satellite link or dial-up link between the sensor node and the MQTT broker. It has the characteristics of small code size, low power consumption, minimized data packet size, and can distribute packets to multiple applications at the same time. It also achieves loose coupling of application programs and became the OASIS standard in 2014. Eclipse Paho [38] provides an open source implementation of the MQTT protocol client, supporting languages such as Java, Python, Javascript, Go, C, .Net (C#), Embedded C/C++, etc. Moquette [39] implemented the Java-based Broker function of MQTT. MQTT.js is a Node.js-based MQTT client library that can be used in web development. MQTT-sn-tools [40] implemented the MQTT protocol version for use in sensor networks. These open source systems are regularly updated at the time of writing.

2.3.3. CoAP Protocol

CoAP [41] is a data transfer protocol between resource-limited devices. It provides four methods for resource data operation: PUT, DELETE, GET, and POST. It is based on the UDP protocol and uses its own lightweight protocol to ensure the reliability of packet transmission. Currently, it is standardized in draft RFC 7252. There are many open-source implementations of this protocol that are based on languages such as Java, C, Go, Node.js and can run on Arduino and other micro controllers.

2.3.4. OPC-UA Architecture

OPC-UA is an interoperable standard that enables industrial automation data to communicate securely and reliably, maintaining platform neutrality and vendor independence. It provides the following functions: defining server and client interfaces, real-time data access, alarm and time monitoring, historical data access, and data modeling [42]. It also features secure communication and user access control.

Table 2 gives a comparison of these transmission protocols and analyzes the role and characteristics of their data transmission in IoT applications.

2.4. Security and Privacy Protection Requirements

Open source systems for embedded system security in IoT applications implement the datagram transport layer security (DTLS) mechanism and transport layer security (TLS). The open source software for the former include C-based Eclipse TinyDTLS [43], Java-based Eclipse Scandium [44], and C-based mbedTLS running on embedded systems [45]. These security mechanisms require a certain amount of computing resources, and their consumption and use of resources in the context of resource-constrained IoT devices require further testing and analysis.

2.5. IoT Device Management

The following gives examples of open source systems for the management of many IoT devices.

2.5.1. Eclipse hawkBit

Eclipse hawkBit [46] supports device software updates and controls, and connects devices to IP networks. It can update the software, operating system, and firmware running on the device, in addition to supporting the monitoring and management of the device. The supported management protocols include OMA-DM/LwM2M. Two kinds of APIs are used to connect to IP networks: the direct device integration API via HTTP, and the device management federation API; these allow the connection of devices with different protocols adapters.

2.5.2. LwM2M Technology

The LwM2M is a lightweight protocol based on the RESTful style for managing sensor devices and M2M devices. It defines a scalable resource or data model and implements the remote management of devices by the CoAP protocol. It is standardized by the OMA device management group [47]. Open source implementations of this protocol include Betwixt [48], based on Go, for LwM2M server and client; Eclipse Wakaama [49] and AwaLWM2M [50], based on C; and Eclipse Leshan [51] based on the Java language.

Table 2. Comparison of IoT application layer protocols.

	Resource Discovery	Supported Protocol	Communication Mode	Openness	Security	Scenario
Modbus	-	TCP/Serial/Ethernet	Master–slave polling, Request/response. and *1-to-n*	No open API and requiring professional knowledge for development	-	Industrial automation, monitoring and data acquisition
OPC-UA	Local or network server discovery	Protocol independent	Publish/subscribe and event-based notification	Device integration with enterprise application, No high-level API	encryption/audit/authentication	Industrial automation
MQTT	-	TCP/IP, WebSocket	*1-to-n* based on publish/subscribe mode	Strong openness	encryption/authorization/authentication	Loosely coupled integration between applications
CoAP	Resource discovery and storage	UDP	Request/response, publish/subscribe, Multicast	Strong openness	encryption	Device integration with WWW

2.6. Application Support Layer (IoT Middleware)

2.6.1. Open Source Middleware Platforms

The role of the application support layer is similar to that of IoT middleware technology. They shield the diversity of the underlying technologies and provide various functions for applications to facilitate the development of IoT applications. At present, the major open source IoT middleware platforms include:

1. Eclipse Kura [52]: The goal of this platform is to build an IoT application gateway. It is an application container that supports remote device management and provides a series of APIs for IoT application development. It supports a variety of peripheral interfaces and application layer protocols. Kura offers external device interfaces, data storage and forwarding, and services publishing, as well as provides an API interface to integrate with cloud services. This project is active and supported by Eclipse, using the EPL-1.0 license.
2. Zetta [53]: This is an API-first application platform for the IoT. It is based on Node.js and can build a distributed IoT server and cloud platform. It communicates with the device through the RESTful API. It can be used to build a distributed IoT application server and can be used as a gateway. It supports publish/subscribe mode and data stream. It features device discovery, registration, and device identification. Zetta is an active project that uses the MIT license.
3. IoTivity [54]: This is a reference implementation of the Open Interconnect Consortium (OIC) standard, which includes functions for the device layer, network layer, and application support layer. It provides RESTful interfaces and supports application layer protocols such as CoAP and MQTT. It implements resource discovery by mDNS and uPNP, resource storage directory, message routing, and security mechanisms. The functions provided by the service layer include simulation tools, protocol conversion, agents, publish/subscribe services, device management, resource encapsulation, resource containers, etc.
4. AllJoyn framework [55]: This is a framework for connecting multiple heterogeneous devices and APP applications. It provides device discovery and communication functions for applications and cloud platforms. It has the following functions: point-to-point and group sessions; external API interfaces; supporting MQTT, XMPP and TR-069 protocols; supporting proximity device discovery and communication; and supporting point-to-point encryption (AES128) and authentication (PSK, ECDSA) security mechanisms. It is also an implementation of the Open Connectivity Foundation (OCF) reference architecture.
5. OpenIot [56]: As an IoT middleware platform, OpenIot can easily deploy various algorithms and applications for data processing, and can generate application events. It provides ontologies, semantic modeling, and semantic development interconnection technologies for data and interconnected devices. It integrates well with Cloud Computing platforms, supports sensor discovery, and can dynamically compose with other services. The project is supported by a number of research institutions and some companies.
6. SiteWhere [57]: This open source software is built on micro-service architecture and hence has good flexibility, scalability, and availability. It uses application layer protocols such as MQTT/AMPP to transfer data and supports data storage by big data technology. SiteWhere also offers an object model to describe data relationships. The system provides RESTful, MQTT/AMPP application development interfaces, which can be used to integrate with Cloud Computing platforms and provide users with a web application runtime environment through web containers. The devices are managed by device self-registration, RESTful service, or batch processing. It also supports security mechanisms, such as encryption, authentication, and role-based access control. It uses the open source CPAL1.0 license.
7. Thinger.io [58]: This middleware provides cloud-based device access management and data service interfaces at the gateway layer that can be deployed in the cloud and local site. It can

run on open source hardware platforms such as Arduino, Raspberry Pi, Intel Edison, and ARM Mbed, and operating systems such as Linux, and Android. It supports the SIGFOX protocol. The security mechanism used is unknown. It has been integrated with NodeMCU to implement motion detection applications [59] and also used in smart emergency response systems [60].

8. WSO2IoT [61]: This includes gateway layer and application layer functions, providing terminal equipment access management, data flow, and CEP processing. It can customize web portals for application development and supports devices that run Android and iOS. It has relatively complete security mechanisms, including encryption, authorization, identity authentication, single sign-on, and transport-layer security mechanisms. The Apache License 2.0 is adopted.
9. ThingsBoard [62]: This provides data acquisition, processing, visualization, and device management functions. HTTPs and MQTTs, SIGFOX, OPC-UA, etc. are supported, while CoAPs are not yet supported. It supports device identity authentication based on X.509. Apache License 2.0 is used and about 10 partners are involved. Recently, it has been adopted by different applications, such as data collection platform for situational awareness-centric microgrids [63]; and real-time processing of data streams received from sensor devices, via integration with Spark [64].
10. DeviceHive [65]: This provides management, communication and data processing functions for devices, providing RESTful API interfaces. Authentication mechanisms based on JWT (JSON Web Token), TLS encryption, and role-based access control are supported. It has good flexibility and reliability by using micro-service architecture. It is released under an Apache License 2.0 license. DeviceHive is used as a connector service to route commands between DeviceHive and other services in a smart home [66]. The services are registered into DeviceHive as devices and are polled for commands. In the smart metering of electrical power systems, DeviceHive has been used as a cloud and client to communicate with remote sensors. It abstracts resources into seven components: Network, User, Device, Device Class, Equipment, and Access key [67].
11. ThingSpeak [68]: This IoT platform developed by the Mathworks company implements data acquisition, analysis, and action triggering functions. The data processing is done by Matlab. It provides cloud platform and supports mobile terminal application development. ThingSpeak has been used in applications such as intelligent agriculture field monitoring systems [69], and automatic car parking systems [70].
12. VSCP (Very Simple Control Protocol) [71]: This is used as a gateway in IoT to provide device discovery and identification, device configuration management, secure device firmware updates, and UI interfaces. It stores and transmits data by a secure encryption mechanism.
13. Macchina.io [72]: This is an edge and fog computing tool set for IoT application development, which supports multiple devices and terminal equipment access, data transmission, integration with a Cloud Computing platform. Seven partners are involved and the Apache 2.0 license has been adopted.
14. T6IotApp [73]: This provides device connectivity, data flow with a timestamp, data processing, a dashboard interface, and a JWT security mechanism. A few different hardware is currently supported.
15. Distributed Services Architecture for IoT (DSA) [74]: DSA realizes the integration of a variety of data or protocols used for communication between heterogeneous devices. It also supports group-based authorization management, LDAP-based authentication, and SSL-based transmissions. The Apache License 2.0 is adopted.
16. Kaa [75]: This is a middleware platform for building end-to-end applications of IoT, which can be used as a gateway or as an application server. It supports device management, device interaction, remote device configuration and distributed remote device firmware updates, cloud service creation, data collection and analysis, user behavior analysis, target event notification, and data storage based on big data. In the literature [76], Kaa has been deployed in clusters as an application server for distributed data collection.
17. PlatformIO [77]: This is an open source IoT development ecosystem. It operates at the application layer and provides a cross-platform build system that integrates a continuous development

environment and library management environment. Its main role lies in being an integrated development environment and framework that supports multiple hardware platforms. There is no support for device networking, discovery, and semantic processing. Companies such as ARM, Freescale, and TI provide support and the Apache 2.0 license is used.

18. Mainflux [78]: This is a hardware-independent IoT cloud platform based on the Go language and micro-service architecture, supporting multiple protocols, device management, OTA-FW-based firmware update, application management, data storage, and fine-grained access control based on customizable API keys and scoped JWT. TLS and DTLS security mechanisms are also supported. The Apache-2.0 License is used.
19. Patchwork Toolkit [79]: Patchwork provides lightweight tools for the interconnection of a variety of heterogeneous devices which are within the LAN. It only retains the necessary features and therefore is easy to deploy with good scalability. It also provides service registration and discovery functions.

2.6.2. Comparison of Open Source Middleware

Through a comprehensive analysis of the main functions of open source IoT middleware and the requirements of IoT applications, the open source platforms were compared using seven features, including both functional and non-functional properties. Other non-functionalities such as flexibility, extensibility, reliability, etc. are not covered here. These seven dimensions are shown in Figure 2. The details are as follows:

1. Access and management of heterogeneous devices. The platform should support various hardware using different protocols and provide management of device status and firmware updates. The open source middleware technologies introduced above almost all support a variety of mainstream open source hardware and mobile terminals, such as the Raspberry Pi, BeagleBone, Arduino, and so on. The underlying access protocols include ZigBee, CAN, RS-232, Bluetooth, GPIO, LoRA, SIGFOX, and others. Some open source platforms provide identity management for access devices.
2. Application layer data transmission. At present, open source platforms mostly support HTTP/MQTT application layer protocols, and some also support application layer protocols such as WebSocket, Modbus, OPC-UA, CoAP, and AMPP.
3. Data storage and intelligent processing. Data can be stored by the middleware itself to provide edge computing capabilities and can also be stored on cloud platforms. Data processing capabilities include complex event processing (CEP), rule-based event processing, and stream processing; some platforms also support machine learning capabilities.
4. Providing API interface and application support capabilities. The platforms above provide applications with various capabilities through RESTful interfaces.
5. Deployment mode. Some open source systems are deployed as gateways, some are simply deployed as IoT applications, and others have both capabilities.
6. Completeness of security. The platform needs to provide data encryption, secure transmission, identity authentication, and access control mechanisms to meet the needs of IoT applications.
7. Activeness. The activeness of development of the open source system is also an important factor when developers choose a platform. The activeness of the 19 platforms investigated here is shown in Figure 3. The number of results obtained through a Google search and the number of topics in the open source system's own forums can roughly reflect its activeness. The topic in the figure indicates the number of topics of each platform, which mainly comes from the Stack Overflow website and self-built forums.

Based on the above features, Table 3 shows a comparison of the open source platforms listed in this paper. The symbol "/" indicates a feature is unknown. In Table 3, we can see that the degree

of activity of the project has a strong correlation with the support it receives. For example, Eclipse Kura is supported by the Eclipse open source organization, and IoTivity has received support from big companies such as Intel, Cisco, and Samsung. WSO2IoT has about 70 partners, and AllJoyn includes more than 100 companies as partners and is supported by the Linux Open Source Foundation. Table 3 also shows that the function of these open source projects is more comprehensive. Most systems are still regularly updated and developed, in addition to some platforms that are mature or have not been maintained. For example, 44 systems were updated in 2018 and 15 were updated in 2017.

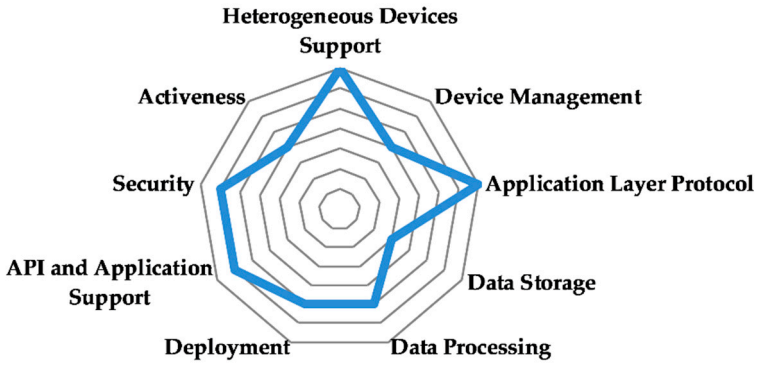


Figure 2. Key features of open source middleware for the IoT.

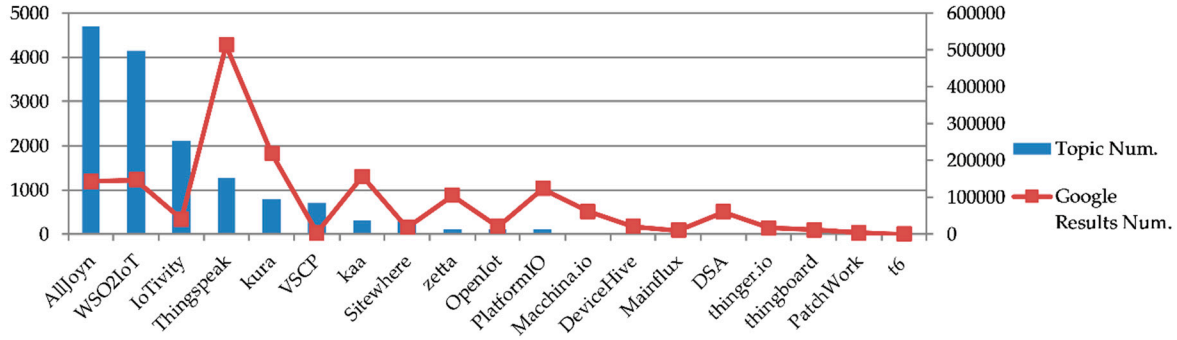


Figure 3. Activeness of development of open source middleware platform for IoT.

Table 3. Comparison of open source IoT middleware platforms.

	P1 [1]	P2 [2]	P3 [3]	P4 [4]	P5 [5]	P6 [6]	P7 [7]	P8 [8]	P9 [9]
Kura	√	√	√	√	X	√	GW [11]/AP [12]	SSL	Great
Zetta	√	X	√	CS [10]	Data stream	√	√	X	Good
Iotivity	√	√	√	CS	X	√	√	√	Excellent
Alljoyn	√	√	√	√	X	√	GW/AP	√	Excellent
OpenIot	X	X	X	CS	SP [13]	X	AP	AA [14]	Good
SiteWhere	√	√	√	CS/LS [15]	√	√	AP	√	Good
Thinger.io	√	√	/	CS	/	√	√	/	Good
WSO2IoT	√	√	√	CS	√	√	√	√	Excellent
ThingsBoard	√	√	√	CS	√	RPC	GW	AA	Excellent
DeviceHive	√		√	√	√	√	GW	√	Good
ThingSpeak	√	X	√	CS	√	√	AP	/	Great
VSCP	√	√	√	X	X	X	GW	/	Good
Macchina.io	√	X	√	√	X	√	GW	√	Good
T6Iot	A few	X	X	/	√	√	AP	√	Poor
DSA	√	X	WS [16]/Http	LS	X	X	GW	√	Good
Kaa	√	√	√	√	√	√	√	/	Good
PlatformIO	√	X	X	X	X	X	X	/	Good
Mainflux	√	√	√	√	√	√	√	√	Good
Patchwork	√	X	√	X	X	√	GW	/	Poor

[1] Heterogeneous devices; [2] Device Management; [3] Application Layer Transfer; [4] Data storage; [5] Intelligent processing; [6] RESTful Interface; [7] Deploy Mode; [8] Security; [9] Activeness; [10] Cloud storage; [11] Gateway; [12] Application platform; [13] Semantic Processing; [14] Authentication Authorization; [15] Local storage; [16] WebSocket.

2.7. Comparison between IoT Requirements and Open Source System

By comparing the open source systems listed above with the various requirements of the IoT listed in [2], the requirements implemented and not yet implemented by open source systems were analyzed. The results are shown in Table 4.

Table 4. Comparison between the IoT requirements and current open source systems.

Layer	Sub-Requirement		Layer	Sub-Requirement	
Device layer	Connection management based on identification	No	Data layer	Data classification	No
	Plug and play	Yes		Rule-based data fusion and data mining	Yes
	Mobility management	No		Data semantic annotation and query	Yes
	Device integrity check	No		Data exchange/aggregation	Yes
				Data lifecycle management	No
				Data description	Yes
Communication layer	Event-based/multicast communication	Yes	Service layer	Service composition	Yes
	Content-aware communication	No		Semantic-based service	Yes
	Self-Management, self-heal, self-optimization, self-protection	No		Mobility service	No
	Periodic communication	No		Autonomous service	No
	Heterogeneous communication integration	Yes		Personalized service	Yes
				Scenario aware service	No
Application development	Programming API	Yes	Security and privacy	Authentication/ Authorization/Audit	Yes
	Group management	Yes		Privacy protection	No
	Collaboration requirement	No		Security policy	No
	Resource accounting	No			

3. Architecture of the Integrated IoT Application Development Platform

3.1. The IoT Application Development Platform

Figure 4 shows the proposed architecture of the integrated IoT application development platform. It includes the following functions.

1. Heterogeneous devices supporting various sensing devices with different functions, operating systems, transmission protocols, power consumptions, and other characteristics. Current mainstream devices are supported, in order to meet the application requirements. Gateways are used to implement the management of heterogeneous devices. The gateway implements device interconnection, state management, and protocol conversion as well as unified description, storage, forwarding, and basic processing of data. It can also automatically produce the ID of the access device according to the device function, protocol, physical location, etc., for identifying and locating the device.
2. Application layer protocols. Data transmission is achieved by using standard application layer protocols, such as HTTP, CoAP, MQTT, or AMQP, to implement integration among different applications. The publish/subscribe mechanism is also supported to implement loose coupling systems and meet real-time application requirements.
3. Distributed resource management. Compared with traditional software development, IoT application development is based on specific sensing devices. During the development, it is necessary to use the functions and specific features provided by the existing devices. Therefore, it is necessary to provide a common public service or resource discovery mechanism to efficiently

and accurately find the available resources it needs. P2P (peer-to-peer) is a distributed computing technology that has been widely used in resource sharing and virtual currency trading. This platform utilizes P2P technology to realize the distributed storage and retrieval of IoT resources. IoT applications developed on P2P therefore have good scalability, flexibility, and high availability. However, these details are beyond the scope of this article.

4. API interface and security mechanism. The platform supports Web Service and RESTful style API interfaces. Network layer security mechanisms such as the DTLS and TLS mechanisms for the embedded environment are adopted to implement data encryption and identity authentication. The JWT is used in the application layer to implement secure transmission of compact and self-contained information. The transferred information is signed to ensure data integrity and non-repudiation.
5. Service delivery platform and development environment. The open source system Platform API is used to provide a cross-platform build system, and integrate a continuous development environment and a variety of development library. IoT applications can be developed and deployed through cloud platforms. The cloud platform stores the global information gathered by the sensors distributed in various places. It processes the data to obtain new knowledge by using intelligent technologies, such as data mining, machine learning, semantic web reasoning, and others. The Eclipse Kapua [80] cloud platform was used as a data processing platform for the IoT and combined with Apache Kafka, an open source real-time streaming processing technology.
6. Resource bill, based on blockchain technology, is used to ensure the safe, reliable, and efficient use of sensor resources. Using P2P resource management, blockchain and smart contract technology were combined to achieve resource management for the IoT. For example, when and where the owner of a resource can publish resource, and how the resource will be used and billed, which is a module that will be further studied at a later date.

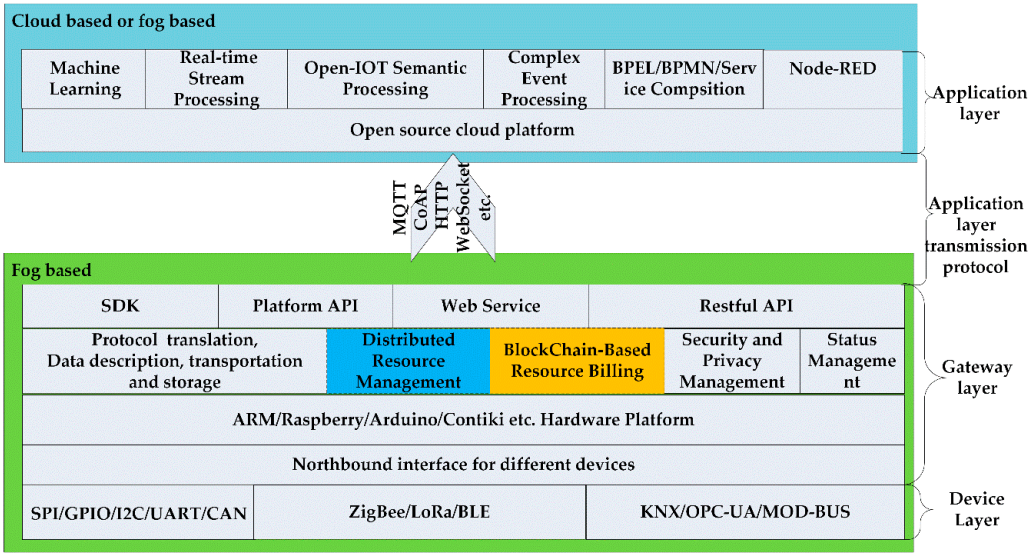


Figure 4. The IoT application development platform, based on open source technology.

3.2. Characteristics of the IoT Application Development Platform

Da Cruz [7] proposed an IoT middleware reference model that included an interoperability module that operates on a gateway for integrating; a persistence and analytics module that stores and processes data; a context module that provides relevant information for services and users; a semantic operation module; and a resource and event module that manages the device capability and generates events according to the capability and context. Compared with this proposal, the system architecture proposed here has the following characteristics:

1. Cloud–fog collaborative computing mode. The system can be deployed in the IoT environment near the sensor devices in a fog computing form. The data do not need to be transmitted to a

remote cloud computing platform and can be processed locally. Then, the processed data can be transmitted to intelligent processing functions which are deployed in the remote cloud computing platform to implement collaborative computing between the cloud and the fog.

2. The northbound interface is utilized to shield the underlying heterogeneity and support different kinds of devices to meet system scalability requirements. At the same time, a southbound interface is provided to the user application for development. The southbound interfaces support different styles of API interfaces, such as Web Service, the open source Platform API, RESTful style interface, and SDK.
3. Multiple open source platforms are integrated. In a large-scale IoT application environment—for example a smart city—different subsystems may adopt different open source platforms or self-development platforms. One of the goals of the system is to use Web of Things technology to implement service interactions between different platforms.
4. The architecture proposed here implements the context processing mechanism through application layer business logic, such as complex event processing, which provides more intelligent application processing capabilities.
5. Distributed resource management and blockchain-based resource billing.

4. Evaluation

The IoT integrated application development platform proposed here was evaluated by an industrial IoT scenario. In an industrial production environment, functions such as environment monitoring, data acquisition processing, and equipment management can be implemented by IoT technology to improve production efficiency and ensure production safety. At present, a typical industrial IoT application is where a factory uses sensors to monitor various data in the production environment [81]. The smart factory has typical IoT application characteristics, such as a diversity of devices and transmission protocols, stringent real-time business requirements, and complexity of data processing. The sensing data collected about the smart factory also need to be transmitted to a remote cloud computing platform for intelligent processing, which can detect abnormal events. The factory production environment is complex and often located far from enterprise management personnel. Therefore, it requires a long-distance wireless data transfer mechanism to achieve remote data collection and management. These protocols also can be easily integrated with enterprise applications.

Figure 5 shows an industrial IoT scenario that implements the acquisition, monitoring, and management control functions of a factory. The simulation environment is shown in Table 5. The Modbus protocol does not have open API and requires professional knowledge for development. For easy integration with various application systems, application-oriented protocols, such as CoAP/HTTP/MQTT/WebSocket, were applied. At present, the CoAP and HTTP protocol are used for remote data transmission. In the future, more application layer protocols will be added, analyzed and compared.

Below, we present the evaluation results, including the system functions and performance analysis.

Table 5. Devices list used for evaluation.

Technology	Product Name and Quantity
Gateway	Raspberry Pi 3 B+ running Eclipse Kura and CoAP Server
Link	LoRA protocol: Lora-gate-bridge, Loraserver, Lora-app-server, packet forwarder, or wireless and wired LAN established via Tp-Link
LoRAWAN Gateway	1
LoRA Terminal Module	2
Transfer protocol	CoAP/HTTP/MQTT/WebSocket
Security	DTLS Eclipse Scandium
Sensors	One temperature and humidity sensor DHT11 connected with GPIO, Two LED lights
MQTT Client	PC
Cloud Platform	Docker-based Eclipse Kapua running on ThinkPad T470 Laptop
CoAP Client	Copper1.0.1 + Firefox55 plugin
GUI	Node-Red +node-red-contrib-coap 0.3.0

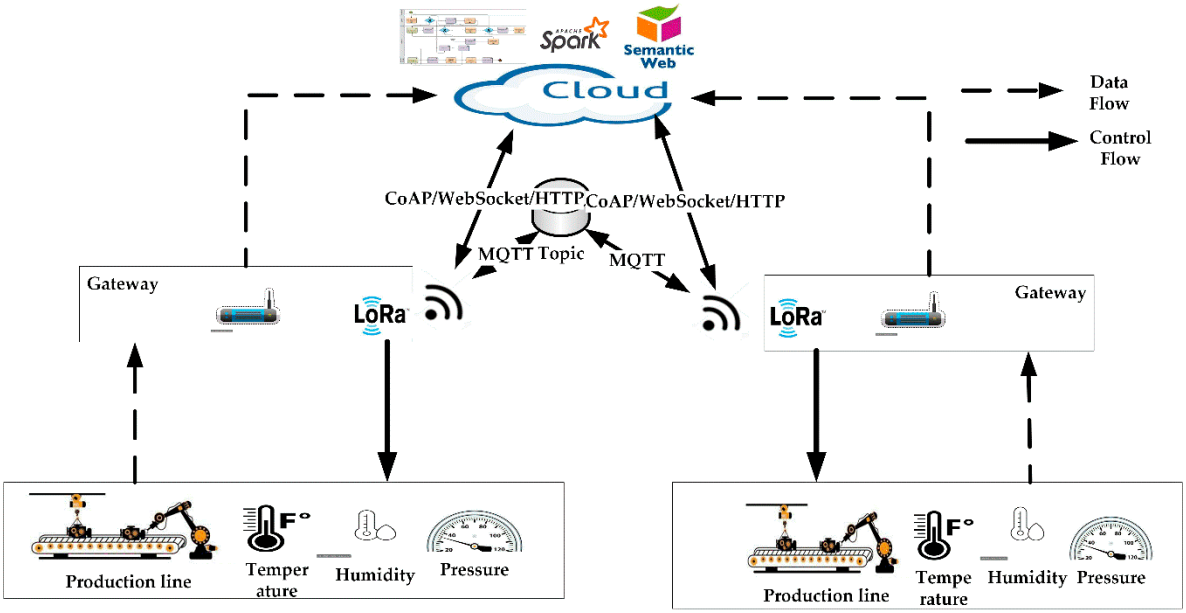


Figure 5. An industrial IoT simulation environment.

4.1. System Function

The system function evaluation results are given in Figure 6. For the evaluation, the following were considered

1. The integrated application development platform is described here using the system functions implementation.
2. The experiment was conducted using a campus network environment. Each measurement result was obtained by performing the test 50 times. Experiments under more complex network conditions will be conducted in the future.
3. We tested the availability of architecture that uses only one kind of gateway technology; the interoperability between different subsystems using different kinds of open source gateways will be studied further.

As shown in Figure 6a, the open source Cloud Computing platform Eclipse Kapua was used to implement MQTT-based IoT gateway management and MQTT broker functions, which included application deployment, system configuration, data publishing and subscription. By using Kapua, the IoT gateway could be managed by an administrator at a remote office, and the data published and subscribed by different applications running in different locations.

Figure 6b shows the resource list for the smart factor from the "well-known" interface of the CoAP protocol by querying the P2P gateway node. These CoAP resources can be operated by the GET, PUT, and DELETE operations, which are invoked by web applications. The resources are accessed by URL. For example, the light resource of factory 1 can be invoked using "GET coap://192.168.1.115:5683/factory1/light\T1\textquotedblright. The resource is managed by a gateway which is run as a P2P node. In our simulations, dozens of P2P nodes are running on the same gateway.

Figure 6c presents the node-red visual interface for developing an IoT application. It displays the visual flow programming mode for IoT application. Figure 6d shows the result of Figure 6c, which shows various environmental data with the Dashboard interface. Figure 6c,d shows that it is easy to develop IoT applications with open source systems.

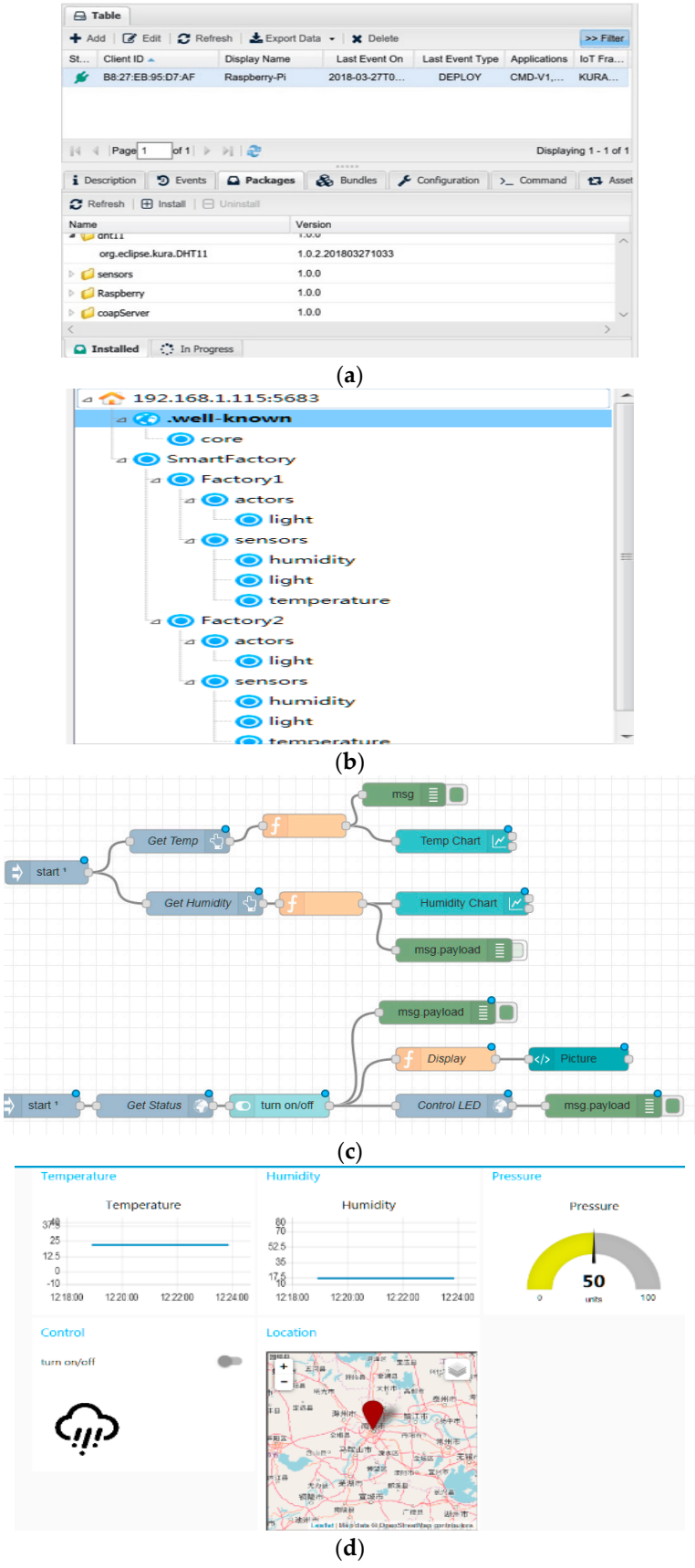


Figure 6. IoT application platform based on open source ecosystem: (**a**) gateway management and data collection; (**b**) CoAP resource discovery; (**c**) Node-Red development; and (**d**) Dashboard GUI.

4.2. System Performance Analysis

Esquiagola et al. [82] proposed a comprehensive test methodology for IoT platform testing. The methodology includes functional, connectivity, performance, security, compatibility, and exploratory testing. It was applied to their SwarmOS platform, but only showed the HTTP response time with a different number of requests on three hardware platforms. They did not report the platform performance under different conditions. Babovic, Z. et al. [83] compared different web applications and IoT message protocol performances in a specific scenario. However, this test is not suited to industrial Internet of Things applications and does not include function testing.

We compared the performance of different mainstream application layer transmission protocols under the condition of the industrial IoT and with the LoRA protocol used as the link layer protocol. Figure 7 shows the transmission performance test model of this system. Node-Red stands for the browser and runs the IoT application developed by Node-Red. The Eclipse Kura Gateway is the IoT gateway. The communication performance between the IoT application and the gateway is found by testing RTT (Round Time Live). T_{trans1} is the request time, $T_{process}$ is the request processing time, T_{trans2} is the response time of request, and T_{render} refers to the time taken by the browser to display the received data. The performance tests were conducted according to Equations (1) and (2). The RTT performance of the CoAP and HTTP protocols were tested and compared using Jmeter and Jmeter-iot-lib based on Equation (1). The GET and PUT operations of the CoAP (POST for HTTP) protocol were compared and are shown in Figure 8 for the CON and NON-CON retransmission mechanism, respectively.

$$RTT = T_{trans1} + T_{process} + T_{trans2} \quad (1)$$

$$RTT = T_{trans1} + T_{process} + T_{trans2} + T_{render} \quad (2)$$

In smart factories, some measurement data are small-sized, such as temperature, humidity, pressure, etc. However, in some cases—for example, when the network is congested—frequent data transmission may affect network performance, so the application can transmit the data measured multiple times at once. To check this situation, the transmission performance under different payloads was tested.

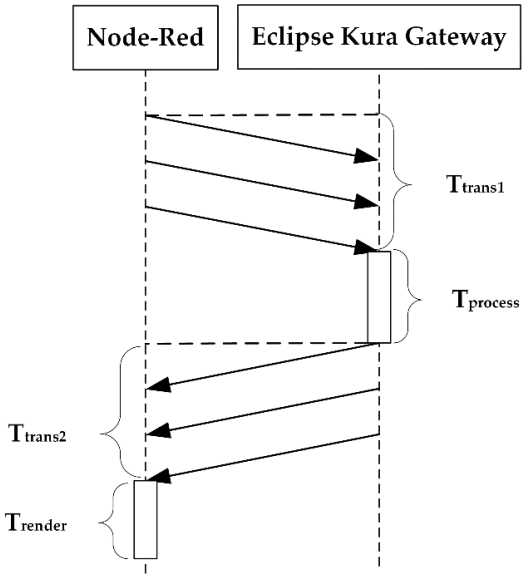
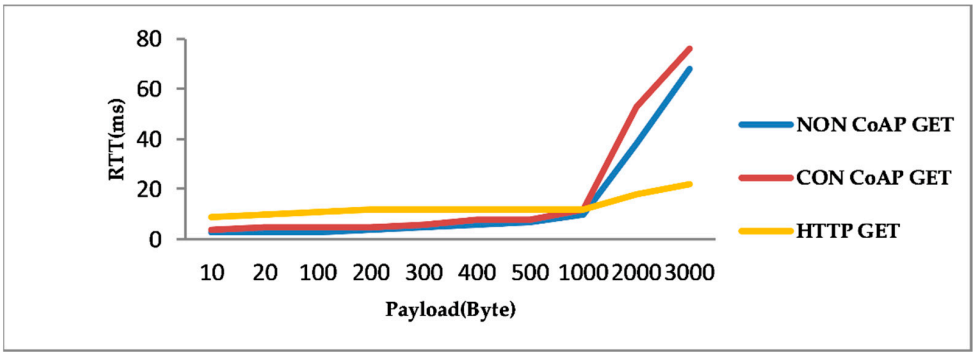


Figure 7. RTT measurement.

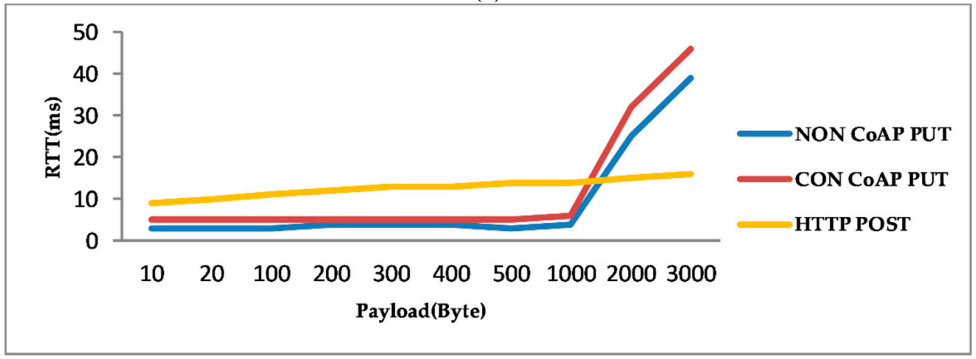
The GET operation was used to obtain the resource data and the PUT operation was used to change the resource state. The CoAP block size was set to the default value. In Figure 8, it can be seen that the performance of the NON and CON mechanisms of the CoAP protocol was close when the

payload was small. When the payload was less than 1000 bytes, the RTT time was less than 10 ms in a wireless LAN environment, regardless of the CON or NON PUT and GET operations. When the transmission data were larger than the CoAP data block size, they increased faster because the data needed to be divided into smaller blocks to be transmitted several times. For example, when the payload was 1000 bytes, the RTT time of the NON CoAP was 10 ms; when the payload was 2000 bytes, the RTT time increased to 38 ms. When the payload was 3000 bytes, the RTT time was increased to 68 ms. The RTT time of the CON CoAP was 12, 53, and 76 ms, respectively. The overhead of CON mechanics also increased with the payload size. This indicates that, when the payload size is bigger than the size of the CoAP block size, the message will be transferred multiple times, and so the CON time is increased.

In Figure 8, we show that the HTTP RTT time was twice that of NON CoAP when the payload size was less than 400 bytes. This is because the HTTP is connection-oriented and based on the TCP protocol. However, when the payload size is more than 1000 bytes, the RTT time of HTTP increased steadily. This is because of the different block size of the two protocols. For example, when the payload size was 2000 bytes, the RTT time of the HTTP GET operation was 18 ms and the RTT time of the CON CoAP GET operation was 53 ms. When the payload size was 3000 bytes, the RTT time of the HTTP GET operation as 22 ms, but the RTT time of the CON CoAP GET operation was 76 ms. In conclusion, when the payload size was small, CoAP is preferred over HTTP. The performance of the CoAP GET and CoAP PUT operations under different data block sizes was tested based on Equation (2) with the Firefox browser. The RTT was measured by setting the CoAP block size to 64 bytes and 1024 bytes, respectively. In Figure 9, it can be seen that the larger the data block, the smaller the RTT. When the package size exceeded 64 bytes or 1800 bytes, the RTT grows faster because the data are transmitted in several segments. This also shows that when the CoAP protocol is integrated with other wireless sensor networks, the transmission performance will be affected if the other network data frames are small. For a block size of 64 bytes (the default size), the CON CoAP GET RTT for Equations (1) and (2) were compared, as shown in Figure 10. The T_{render} was much larger than both the RTT, indicating that the application process time of Firefox was much larger than the CoAP protocol transmission needs.

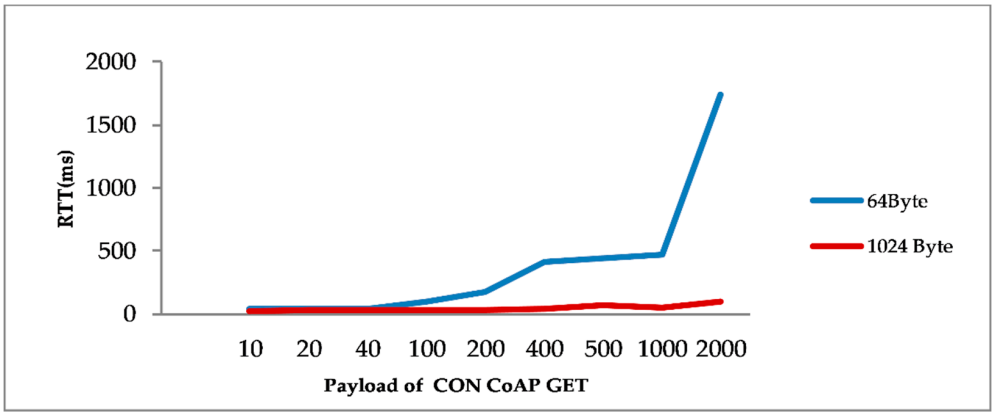


(**a**)

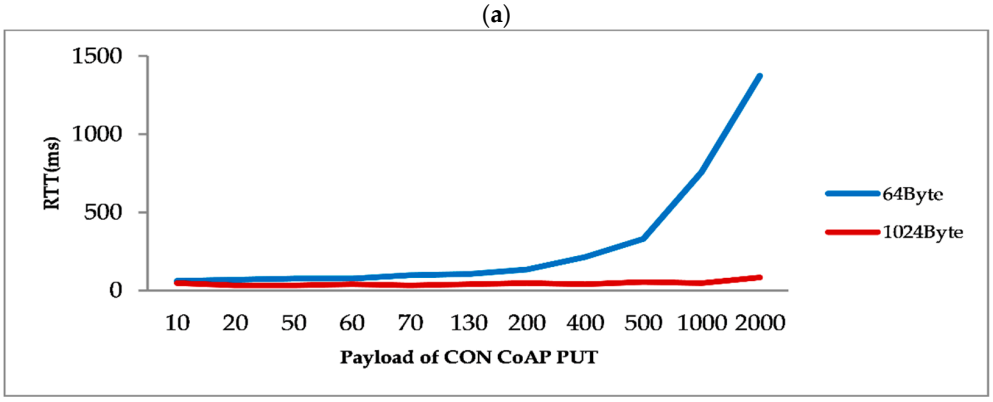


(**b**)

Figure 8. CoAP vs. HTTP: (**a**) CoAP GET and HTTP GET; and (**b**) CoAP PUT and HTTP POST.



(**a**)



(**b**)

Figure 9. Impact of different CoAP data block size on performance: (**a**) CoAP GET RTT; and (**b**) CoAP PUT RTT.

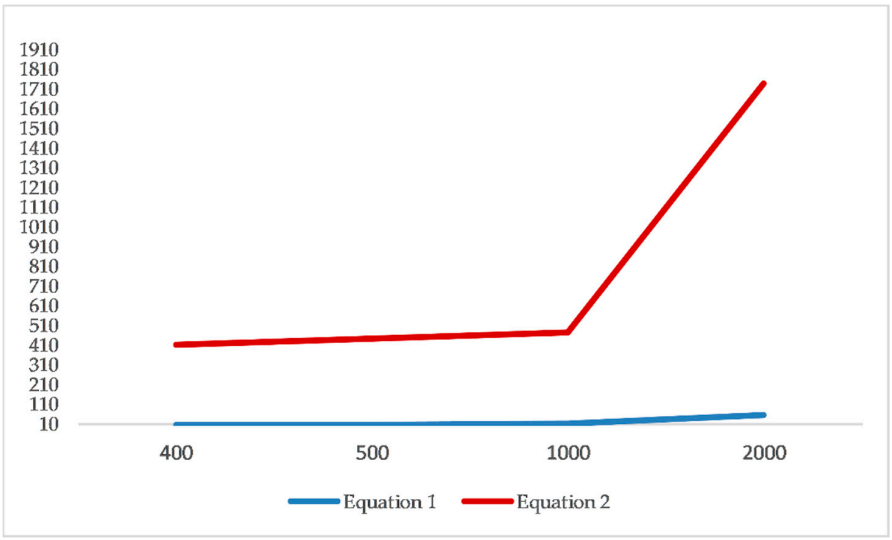


Figure 10. The CON CoAP GET performance comparison between Equations (1) and (2).

Figure 11 shows the availability of the Eclipse Kura gateway under different load conditions. In Figure 11a, it can be seen that, when the payload size was 2000 bytes, the system success rate decreased quickly with increasing concurrency number. When the number was 60, the success rate decreased to 80. When it exceeded 100, the success rate was reduced to less than 50%. At the same time, the memory usage of the Eclipse Kura gateway remained stable, and the CPU usage increased rapidly when the number of concurrencies exceeded 400. In Figure 11b, the payload was set to 1000 bytes.

In this case, the success rate of the system was 100% when the concurrency was less than 400. Even if it was increased to 1000, the success rate was still above 96%. Therefore, it can be concluded that the success rate was high and the system stable and reliable with small payload data. In both cases, the system memory usage was stable and the CPU usage increased rapidly when the concurrency number exceeded 200.

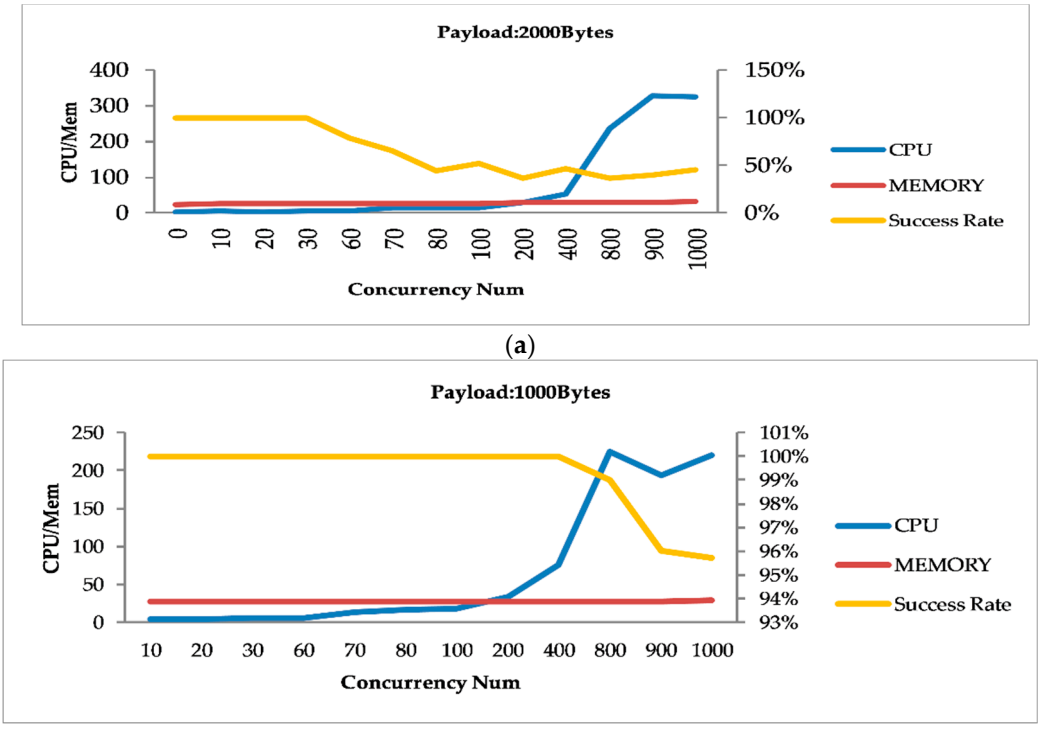


Figure 11. System performance under different loads: (**a**) a payload of 2000 bytes; and (**b**) a payload of 1000 bytes.

5. Conclusions

We surveyed and compared the mainstream open source technologies for the IoT based on the technical requirements of different aspects of IoT, such as device management, data management, data communications, security and privacy protection, and application support. The requirements for IoT unimplemented by open source were analyzed. Then, A kind of cloud–fog cooperated integrated application development platform architecture for IoT applications based on an open source ecosystem was proposed and evaluated in an industrial IoT scenario. The architecture also applied the P2P technology for distributed resource management and blockchain-based smart contract mechanics for resource billing management.

The preliminary function tests allowed us to confirm that the architecture is viable for IoT application development. The performance results showed that, when the sensor data size was small, which it is in most cases, the CoAP protocol performed better than HTTP. The gateway availability test also showed that the IoT gateway based on the open source ecosystem had a stable and reliable performance with a certain data size and concurrency scale, and that this scale could meet the application requirements of the IoT in most sensing environments. However, it is necessary to conduct further research to implement more functions.

The Kura middleware and Eclipse Kapua Cloud Computing platform were adopted to perform a preliminary implementation and performance analysis of the application development platform in a simple simulation topology. In future work, it is necessary to further study how to use a variety of open source middleware to collaboratively build a comprehensive platform for the IoT for complex

application scenarios. In terms of network transmission, the CoAP/HTTP network performance in a LoRA network environment with long-distance communication will be further analyzed and compared. Meanwhile, device access and device life cycle management, such as automatic description, automatic access, automatic registration, automatic release, and automatic update for physical entities, will be further studied. In the future, we will integrate intelligent technologies together into the platform for data processing and customized services. For example, machine learning will be adopted to learn from sea of IoT data to help people make correct and smart decision. Semantic technology will be used combining with Knowledge Base and Rule system for reasoning to obtain useful information and implementing application interoperability as well. SDN (Software Defined Networking) will be adopted for efficient and personal IoT service providing for users.

Funding: This work was funded by the National Natural Science Foundation of China (No. 61502246), the Research Innovation Program for College Graduates of Jiangsu Province (No. CXZZ12_0482), and the Research Project of Nanjing University of Posts and Telecommunications (No. XK0160915170).

Conflicts of Interest: The author declares no conflict of interest.

References

1. ITU-T Recommendation Y, 2066. Common Requirements of the Internet of Things. Available online: https://www.itu.int/ITU-T/recommendations/rec.aspx?rec=12169&lang=en (accessed on 18 July 2017).
2. Shen, S.; Yang, Z. Architecture of Internet of Things and its standardization. *J. Nanjing Univ. Posts Telecommun.* **2015**, *35*, 1–18.
3. Belli, L.; Cirani, S.; Davoli, L.; Gorrieri, A.; Mancin, M.; Picone, M. Design and Deployment of an IoT Application Oriented Testbed. *IEEE Comput.* **2015**, *48*, 32–40. [CrossRef]
4. Fit IOT-LAB. Available online: https://www.iot-lab.info/ (accessed on 18 June 2018).
5. Sotres, P.; Santana, J.R.; Sanchez, L.; Lanza, J.; Munoz, L. Practical Lessons from the Deployment and Management of a Smart City Internet-of-Things Infrastructure: The SmartSantander Testbed Case. *IEEE Access* **2017**, *5*, 14309–14322. [CrossRef]
6. Sánchez, L.; Gutiérrez, V.; Galach, J.A.; Sotres, P.; Santana, J.R.; Casanueva, J.; Muñoz, L. SmartSantander: Experimentation and service provision in the smart city. In Proceedings of the 16th International Symposium on Wireless Personal Multimedia Communications (WPMC), Atlantic City, NJ, USA, 24–27 June 2013; pp. 1–6.
7. Da Cruz, M.A.A.; Rodrigues, J.J.P.C.; Al-Muhtadi, J.; Korotaev, V.; Albuquerque, V.H.C. A Reference Model for Internet of Things Middleware. *IEEE Internet Things J.* **2018**, *99*, 871–883. [CrossRef]
8. Razzaque, M.A.; Milojevic-Jevric, M.; Palade, A.; Clarke, S. Middleware for Internet of Things: A Survey. *IEEE Internet Things J.* **2016**, *3*, 70–95. [CrossRef]
9. Palade, A.; Cabrera, C.; White, G.; Razzaque, M.A.; Clarke, S. Middleware for Internet of Things: A quantitative evaluation in small scale. In Proceedings of the IEEE 18th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), Macau, China, 12–15 June 2017; pp. 1–6.
10. Savaglio, C.; Fortino, G. Autonomic and Cognitive Architectures for the Internet of Things. In Proceedings of the IDCS 2015 Proceedings of the 8th International Conference on Internet and Distributed Computing Systems, Windsor, UK, 2–4 September 2015; pp. 39–47.
11. Fortino, G.; Savaglio, C.; Puga, J.S.D.; Ganzha, M.; Paprzycki, M.; Montesinos, M.; Liotta, A.; Llop, M. Towards Multi-layer Interoperability of Heterogeneous IoT Platforms: The INTER-IoT Approach. In *Integration, Interconnection, and Interoperability of IoT Systems*; Internet of Things Book Series (ITTCC); Springer: Dordrecht, The Netherlands, 2018; pp. 199–232. ISBN 978-3-319-61299-7.
12. Savaglio, C.; Fortino, G.; Gravina, R.; Russo, W. A methodology for integrating internet of things platforms. In Proceedings of the 2018 IEEE International Conference on Cloud Engineering (IC2E), Orlando, FL, USA, 17–20 April 2018; pp. 317–322.
13. BlueZ. Available online: http://www.bluez.org/bluez-architecture-overview/ (accessed on 2 May 2018).
14. Android BLE. Available online: http://www.huwei.tech/ (accessed on 7 June 2018).
15. Yoon, C.; Choi, H.; Cho, J.; Kim, Y.W. CoAP over BLE-GATT for OCF. In Proceedings of the International Conference on Information and Communication Technology Convergence: ICT Convergence Technologies Leading the Fourth Industrial Revolution, ICTC, Jeju, Korea, 18–20 October 2017; pp. 32–34.

16. Noble. Available online: https://github.com/noble/noble (accessed on 5 May 2018).
17. A Node.js Module for Implementing BLE (Bluetooth Low Energy) Peripherals. Available online: https://github.com/noble/bleno (accessed on 5 May 2018).
18. NFC Tools. Available online: http://nfc-tools.org/ (accessed on 8 May 2018).
19. Sinha, R.S.; Wei, Y.; Hwang, S.H. A survey on LPWA technology: LoRa and NB-IoT. *ICT Express* **2017**, *3*, 14–21. [CrossRef]
20. LoRa Gateway Bridge Abstracts the Packet Forwarder Protocol into JSON over MQTT. Available online: http://docs.loraserver.io/lora-gateway-bridge/ (accessed on 5 May 2018).
21. Raza, U.; Kulkarni, P.; Sooriyabandara, M. Low Power Wide Area Networks: An Overview. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 855–873. [CrossRef]
22. Easy-Iot. Available online: https://www.easy-iot.cn/ (accessed on 9 May 2018).
23. CodeLab. Available online: https://github.com/softbaddog/iot-codelabs (accessed on 9 May 2018).
24. Information Technology—UPnP Device Architecture. Available online: https://www.iso.org/standard/69286.html (accessed on 10 May 2018).
25. Bormann, C. Internet Engineering Task Force (IETF) Request for Comments: RFC 7049. Available online: http://cbor.io/ (accessed on 10 May 2018).
26. Pavel, K. Implementace a Evaluace Protokolu CBOR. Bachelor Thesis, Charles University, Prague, Czech Republic, 2015.
27. Flatbuffers Overview. Available online: https://google.github.io/flatbuffers/ (accessed on 10 May 2018).
28. Protocol Buffers. Available online: https://developers.google.com/protocol-buffers/ (accessed on 10 May 2018).
29. Iotdb-Vocabulary. Available online: https://github.com/dpjanes/iotdb-vocabulary (accessed on 12 May 2018).
30. Iotdb-Models. Available online: https://github.com/dpjanes/iotdb-models (accessed on 12 May 2018).
31. Gyrard, A.; Datta, S.K.; Bonnet, C. Cross-Domain Internet of Things Application Development: M3 Framework and Evaluation. In Proceedings of the 3rd International Conference on Future Internet of Things and Cloud, Rome, Italy, 24–26 August 2015; pp. 9–16.
32. Schachinger, D.; Kastner, W. Semantic interface for machine-to-machine communication in building automation. In Proceedings of the IEEE 13th International Workshop on Factory Communication Systems (WFCS), Trondheim, Norway, 31 May–2 June 2017.
33. Fiware-Iot-Discovery-sr. Available online: https://github.com/UniSurreyIoT/fiware-iot-discovery (accessed on 10 May 2018).
34. Semantic Annotator. Available online: https://github.com/komi786/SemanticAnnotator (accessed on 20 May 2018).
35. Aura-Middleware. Available online: https://github.com/AuraMiddleware/aura-middleware (accessed on 20 May 2018).
36. Blackstock, M.; Lea, R. IoT interoperability: A hub-based approach. In Proceedings of the International Conference on the Internet of Things, Cambridge, MA, USA, 6–8 October 2014; pp. 79–84.
37. Kuang, Y. Communication between PLC and Arduino Based on Modbus Protocol. In Proceedings of the 4th International Conference on Instrumentation and Measurement, Computer, Communication and Control, Harbin, China, 18–20 September 2014; pp. 370–373.
38. Eclipse Paho. Available online: https://www.eclipse.org/paho/ (accessed on 20 May 2018).
39. Moquette. Available online: https://github.com/andsel/moquette (accessed on 25 May 2018).
40. Command Line Tools Written in C for the MQTT-SN (MQTT For Sensor Networks) protocol. Available online: https://github.com/njh/mqtt-sn-tools (accessed on 25 May 2018).
41. Constrained Application Protocol (CoAP) Draft-Ietf-Core-Coap-08. Available online: https://datatracker.ietf.org/doc/draft-ietf-core-coap/ (accessed on 25 November 2017).
42. Unified Architecture. Available online: https://opcfoundation.org/about/opc-technologies/opc-ua/ (accessed on 25 May 2018).
43. Eclipse TinyDTLS. Available online: https://projects.eclipse.org/projects/iot.tinydtls (accessed on 30 May 2018).
44. Eclipse Scandium. Available online: http://www.eclipse.org/californium (accessed on 30 May 2018).
45. Mbed TLS. Available online: https://tls.mbed.org/ (accessed on 30 May 2018).

46. Eclipse hawkBit. Available online: https://github.com/eclipse/hawkbit (accessed on 30 May 2018).
47. Rao, S.; Chendanda, D.; Deshpande, C.; Lakkundi, V. Implementing LWM2M in constrained IoT devices. In Proceedings of the IEEE Conference on Wireless Sensors, Melaka, Malaysia, 24–26 August 2015; pp. 52–57.
48. Betwixt. Available online: https://github.com/zubairhamed/betwixt (accessed on 30 May 2018).
49. Wakaama. Available online: https://eclipse.org/wakaama/ (accessed on 30 May 2018).
50. AwaLWM2M. Available online: https://github.com/ConnectivityFoundry/AwaLWM2M (accessed on 30 May 2018).
51. Eclipse Leshan. Available online: https://eclipse.org/leshan/ (accessed on 30 May 2018).
52. Eclipse Kura. Available online: https://www.eclipse.org/kura/ (accessed on 30 June 2017).
53. Zetta. Available online: http://www.zettajs.org/ (accessed on 30 June 2017).
54. Lee, J.C.; Jeon, J.H.; Kim, S.H. Design and implementation of healthcare resource model on IoTivity platform. In Proceedings of the International Conference on Information and Communication Technology Convergence, Jeju, Korea, 19–21 October 2016; pp. 887–891.
55. Costa, D.; Mingozzi, E.; Tanganelli, G.; Vallati, C. An AllJoyn to CoAP bridge. In Proceedings of the IEEE 3rd World Forum on Internet of Things, Reston, VA, USA, 12–14 December 2016; pp. 395–400.
56. Kim, J.; Lee, J.W. An open service framework for the Internet of Things. In Proceedings of the IEEE World Forum on Internet of Things (WF-IoT), Seoul, Korea, 6–8 March 2014; pp. 89–93.
57. SiteWhere LLC., SiteWhere System Architecture. Available online: http://documentation.sitewhere.org/architecture.html (accessed on 30 May 2018).
58. Thinger.io. Available online: https://www.thinger.io/ (accessed on 10 June 2018).
59. Kodali, R.K.; Sundeep, V.; Gorantla, K. RESTful Motion Detection and Notification using IoT. In Proceedings of the International Conference on Computer Communication and Informatics (ICCCI), Coimbatore, Tamilnadu, India, 4–6 January 2018; pp. 1–5.
60. Zander, J. Smart emergency response system. In Proceedings of the TENCON 2017–2017 IEEE Region 10 Conference, Penang, Malaysia, 5–8 November 2017; pp. 712–717.
61. WSO2IoT. Available online: https://wso2.com/iot (accessed on 10 June 2018).
62. ThingsBoard. Available online: https://thingsboard.io/ (accessed on 12 June 2018).
63. Alavi, S.A.; Rahimian, A.; Mehran, K.; Alaleddin, J.A.M. An IoT-Based Data Collection Platform for Situational Awareness-Centric Microgrids. In Proceedings of the IEEE Canadian Conference on Electrical & Computer Engineering (CCECE), Quebec City, QC, Canada, 13–16 May 2018.
64. Paolis, L.T.D.; Luca, V.D.; Paiano, R. Sensor Data collection and analytics with ThingsBoard and Spark Streaming. In Proceedings of the IEEE Workshop on Environmental, Energy, and Structural Monitoring Systems (EESMS), Salerno, Italy, 21–22 June 2018; pp. 1–6.
65. DeviceHive. Available online: https://devicehive.com/ (accessed on 15 June 2018).
66. Lyaskov, M.; Spasov, G.; Petrova, G. A practical implementation of smart home energy data storage and control application based on cloud services. In Proceedings of the 2017 XXVI International Scientific Conference Electronics (ET), Sozopol, Bulgaria, 13–15 September 2017; pp. 1–4.
67. Shopov, M.P. An M2M solution for smart metering in electrical power systems. In Proceedings of the 39th International Convention on Information and Communication Technology, Electronics and Microelectronics, Opatija, Croatia, 30 May–3 June 2016; pp. 1141–1144.
68. ThingSpeak. Available online: https://thingspeak.com/ (accessed on 15 June 2018).
69. AshifuddinMondal, M.; Rehena, Z. IoT Based Intelligent Agriculture Field Monitoring System. In Proceedings of the 8th International Conference on Cloud Computing, Data Science & Engineering (Confluence), Noida, India, 11–12 January 2018; pp. 625–629.
70. Mendiratta, S.; Dey, D.; Rani Sona, D. Automatic car parking system with visual indicator along with IoT. In Proceedings of the International Conference on Microelectronic Devices, Circuits and Systems, ICMDCS 2017, Vellore, India, 10–12 August 2017; pp. 1–3.
71. VSCP (Very Simple Control Protocol). Available online: http://www.vscp.org/ (accessed on 15 June 2018).
72. Macchina.io. Available online: https://www.macchina.io/ (accessed on 15 June 2018).
73. T6IotApp. Available online: https://api.internetcollaboratif.info/ (accessed on 15 June 2018).
74. Distributed Services Architecture for IoT (DSA). Available online: http://iot-dsa.org/ (accessed on 16 June 2018).
75. Kaa. Available online: https://www.kaaproject.org/ (accessed on 16 June 2018).

76. Cruz Huacarpuma, R.; de Sousa Junior, R.; de Holanda, M.; de Oliveira Albuquerque, R.; García Villalba, L. Distributed Data Service for Data Management in Internet of Things Middleware. *Sensors* **2017**, *17*, 977. [CrossRef] [PubMed]
77. PlatformIO. Available online: https://platformio.org/ (accessed on 20 June 2018).
78. Mainflux. Available online: https://www.mainflux.com/ (accessed on 20 June 2018).
79. Patchwork. Available online: http://patchwork-toolkitgithub.io/ (accessed on 20 June 2018).
80. Eclipse Kapua. Available online: https://www.eclipse.org/Kapua/ (accessed on 25 June 2018).
81. Fortino, G.; Savaglio, C.; Zhou, M. Toward opportunistic services for the industrial Internet of Thing. In Proceedings of the 13th IEEE Conference on Automation Science and Engineering (CASE), Xi'an, China, 20–23 August 2017; pp. 825–830.
82. Esquiagola, J.; Costa, L.; Calcina, P.; Fedrecheski, G.; Zuffo, M. Performance Testing of an Internet of Things Platform. In Proceedings of the 2nd International Conference on Internet of Things, Big Data and Security, Porto, Portugal, 24–26 April 2017; pp. 309–314.
83. Babovic, Z.; Protic, J.; Milutinovic, V. Web Performance Evaluation for Internet of Things Applications. *IEEE Access* **2016**, *4*, 6974–6992. [CrossRef]