Enabling IoT Connectivity and Interoperability by using Automated Gateways

Jasminka Matevska and Marvin Soldin

City University of Applied Sciences, Flughafenallee 10, 28199 Bremen, Germany
jasminka.matevska@hs-bremen.de
marvin.soldin@stud.hs-bremen.de

Abstract. As an essential part of the Industry 4.0 strategy, the Internet of Things is developing to "Internet of Everything". The number of interconnected devices and the amount of data produced increases constantly. Various devices are communicating using various protocols, exchanging data using various data formats and connecting to various software applications. In an IoT-Architecture, a gateway is a fundamental component needed for enabling device interoperability. While a great deal of research has already been done on IoT in cloud computing, fog computing, and edge computing, there is still no intensive activity in the field of gateways in particular. Even the basic gateways can act as a proxy between low-end IoT devices and data centres, automated gateways can provide significantly higher functionality to solve the problems of diversity of protocols, data formats and the custom needs of various devices including used applications. This paper presents a concept of an automated gateway dealing with the problems of protocol conversion, device management, middleware abstraction, resource management and traffic optimisation. The gateway is designed as a modular plug-and-play architecture and was evaluated for MQTT, ZigBee, WebSocket and Amazon WebServices. The architecture can be extended by including additional modules to support further protocols and services. Finally, the gateway defines its protocol and translates incoming messages into an optional uniform format, which can be used by the client to enable more complex message flows. Thus, it provides a solid foundation for further development towards standardization of communication interfaces and interoperability of IoT devices.

Keywords: Industry 4.0 · Internet of Things · IoT Architecture · Interoperability · Connectivity · Automated Gateway · Communication Protocols.

1 Introduction

In recent years, many smart devices have been connected to the Internet to record and process data for various purposes. For example, the applications smart health, smart city and smart home are most common. However, also various private applications fall into this field. In general, this concept can be described as the Internet of Things (IoT). [7]

With the growing number of devices, especially in the scope of the Industry 4.0 digitisation of the production, this environment is becoming more and more important. Nowadays, almost everything is networked, so the amount of data being produced is constantly increasing. In 2025, it is predicted that the total number of IoT devices will be approximately 75.44 billion [1]. Furthermore, in the industrial IoT context (IIoT) various industrial and production processes require an efficient and reliable capture, transmission and processing of device data. The devices, including sensors and actuators, rely on a gateway to aggregate the data transmitted by heterogeneous devices using various communication protocols. The gateway usually forwards the data from one to another destination using different processing and storage platforms like cloud computing. Thus, a gateway is a decisive part of the Internet of Things. As per IoT architecture, a gateway is a device that acts as a connection point between IoT devices and their applications. It is an essential aspect of an IoT system since most IoT devices cannot establish a direct connection to the cloud. [4]

While a great deal of research has already been done on IoT in cloud computing, fog computing, and edge computing, little seems to have happened in the field of gateways in particular [4]. Therefore, there are still many problems to be solved in this field. One of the obstacles to IoT development is the lack of standardization of communications, especially considering the heterogeneity of protocols for communication with the devices and the high variability of these devices. It is common for sensor manufacturers to use closed standard protocols or different messaging protocols that prevent or hinder their use by third-party applications. This situation leads to the so-called interoperability crisis in IoT. [8] Furthermore, the protocols heterogeneity causes problems and the device management itself. Each device must be configured in order to connect to the cloud. This involves configuration on the device side and the cloud side. Unfortunately, current gateways operate either passively or semi-automatically. This means that the user has to add new devices manually. [19] Unfortunately, these are not the only challenges a gateway has to deal with. Another problem that emerges from this challenge is that IoT gateways need to forward a large amount of data in a short time, which requires IoT gateways to have the ability to handle concurrent mass data. If more devices are added, the gateway may be overloaded and no longer be able to process the amount of data. [20]

A concept that can be called automatic gateway is required to solve these problems by minimizing manual intervention. In this paper we present a modular extensible architecture of an automated gateway as a possibility to deal with these problems thus pawing the way for further development towards a standardized solution. The initial prototype of the gateway was developed in the context of the master's thesis [23] and is a subject of ongoing research work in the IoT Systems Lab at the Bremen City University of Applied Sciences.

This paper is structured as follows. First we present the state of the art and define the terms (e.g. gateways types and characteristics) used. The following two sections identify the challenges of solving the open issues and summarize the related work. In the section 4 we propose a modular extensible architecture and a

corresponding prototype addressing the challenges. In the section 5 we show that such an approach is feasible at principle. Additionally, the section 6 compares our proposed solution with already existing ones from the economic sector. Finally, in the section 7 we conclude the achievement and point to necessary and possible future work.

2 State of the Art

In the Internet of Things, a gateway acts as an intermediary between several different sensors and cloud platforms. These gateways aim to solve the heterogentity created by different devices and protocols at the sensor level and to forward the resulting data to the cloud. [4]

2.1 Types of Gateways

IoT Gateway can be fundamentally divided into the basic gateway and the smart gateway. While basic gateways only act as a proxy between low-end IoT devices and data centres, smart gateways include significantly more functionality. In contrast, a smart gateway handles data efficiently by preprocessing, filtering, analyzing, and delivering only the related or necessary data to the data centre. Furthermore, these intermediate devices are designed to handle harsh environmental conditions to recover from failure while solving the communication gap in minimal time. A smart gateway can be divided into three subtypes based on its functionalities: passive, semi-automated and automated gateways [4]. Sometimes gateways are designed to act smart by discovering IoT devices, registering them to the network, or removing them for better performance. When using passive gateways, these functionalities are performed manually. The user is adding or removing devices by himself/herself while using a specific setup manual/guide. However, these gateways are not customizable and flexible. [4]

On the other hand, a semi-automated gateway manages a link between newly added devices and automatically creates connections to new devices. These gateways support a pluggable configuration architecture, which means they can be plugged in based on the requirement of the network device. As a result, these gateways are more flexible than passive gateways and perform better in real-time applications. As a next level, there is a fully-automated gateway, that is self-configurable and self-manageable. There is no human intervention, and the devices can be added and removed automatically. Furthermore, these gateways can operate efficiently in a heterogeneous network and communicate over many different protocols such as WI-FI, MQTT, Constrained Application Protocol (CoAP), Bluetooth, ZigBee or Ethernet and many more. Work has been done in this specific field, but it is still in the growing phase, and researchers focus on making gateways smart enough to provide better performance and Quality of Service (QoS). [4]

According to [4], the criteria a gateway shall fulfill in order to be considered an automated gateway are highlighted as follows.

- *Protocol conversion*: The gateway should contain an approach to solve the so-called interoperability crisis in the IoT area. Therefore, the gateway should be able to support several protocols. The incoming messages should then be transformed into another protocol explicitly selected for the cloud communication.
- *Device management*: Device management should be as automatic as possible, i.e. devices that connect to the gateway should be automatically registered and integrated into the communication with the cloud.
- *Middleware abstraction*: As already described, the gateway is to mediate between intelligent devices and IoT middleware/cloud applications. It is possible to connect to different cloud providers available on the market with such a function.
- *Ressource management*: The gateway should work as reliably as possible and react automatically to overloads. This means that the available resources are monitored and automatically distributed. Therefore, the gateway should be able to recognise high-priority applications. The goal of operability is crucial here because the gateway should always be accessible for remote maintenance.
- *Traffic optimization*: There are some promising possibilities to gain performance using streaming optimisation techniques, such as segmenting data into queues, performing packet prioritisation on specific queues, and queuing compression and deduplication for transmission.

2.2 Open Issues and Challenges

Based on 2347 articles from the last then years, i.e., from 2011 till July 2021, 75% of the research was conducted in the field of passive gateways, while 18% was on semi-automated and only 7% on fully-automated gateways. For this reason, there are still various problems to be solved. Some of the open issues and research challenges are summarized as follows. [4]

- *Heterogeneity*: IoT applications deal with many heterogeneous smart devices to gather data in different formats and sizes. The gateway is a bridge between all the connecting devices and should handle heterogeneous data among different protocols. While several models and architectures have been proposed to solve this problem, there is still no standardized solution.
- *Scalability*: Gateways should handle the increasing amount of IoT devices without deteriorating the services applied to them. With growing technology, smart devices are also exponentially increasing, leading to issues regarding scalability. For this reason, it is an open issue to distribute the computing resources smartly so that failures do not occur.
- *QoS*: While dealing with a large amount of data and services, the gateway should also provide a quality of service while dealing with real-time data. Many parameters were individually considered, like reliability, latency, energy efficiency, scalability, but parameters like throughput, roundtrip delay and jitter were overlooked. Furthermore, the simultaneous parameters should also be considered for improving QoS at gateways in IoT.

- *Security and Privacy*: Since gateways are one of the significant data entry points to the network, it is more vulnerable to threats and malicious activities. There was already some work focused on network security, but algorithms and architectures can also opt for more secure gateways and IoT architecture.
- *Intelligence*: Gateways should be smart enough to deal with intelligent data offloading. It is still an open problem how much data should be offloaded, in which case, for how much processing.
- *Robustness*: Based on the previous work, no concepts of recovery of the gateway, fault tolerance, and self-healing were found. Thus, no work has been invested in these topics yet.

Based on these existing problems and research gaps, several exciting research directions have been identified. While much work has already been done to deal with heterogeneity, scalability and interoperability problems individually, there is still a need for a standardized solution to solve these problems simultaneously. Additionally, QoS parameters can also be measured simultaneously to improve IoT systems' performance. Furthermore, machine learning and deep learning techniques are exciting topics because they could be used in combination with traditional security algorithms to deal with privacy and security concerns in IoT gateways. However, working on fully automated gateways is particularly important. Gateways must make spontaneous decisions based on user behavioural patterns and complex situations. Therefore, working more with intelligent and smart gateway can be an exciting challenge. Nevertheless, this aspect is not only important; in the future, there is a need for self-healing and fault-tolerant gateways to deal with the colossal amount of data. In dealing with Big Data, gateways should be reliable, fault-tolerant, self-manageable and self-healed. [4]

3 Related Work

The open problems and challenges discussed in the previous section are treated by different research and development approaches. Some of them are partly covering the aspects related to the presented approach and thus presented as follows.

In [6] a Semantic Gateway as a Service (SGS) was presented that enables the translation of message protocols such as the Extensible Messaging and Presence Protocol (XMPP), CoAP and MQTT via a multi-protocol proxy architecture. The SGS also provides intelligent solution by integrating Semantic Web technologies with existing sensor and service standards. The SGS is also integrated with semantic service such as SemSOS [15] to further elevate interoperability at service level.

Another gateway was presented in [18]. The proposed result is similar to the client/server model of the Hypertext Transfer Protocol (HTTP) and uses the Constrained Application Protocol (CoAP) for device-to-device (D2D) communication. An operation is equivalent to that of HTTP and is sent by a client to

request action on a resource identified by a Uniform Resource Identifier (URI) on a server. The server then sends a response with a response code. The gateway was especially designed to be suitable for in-home-scale environments. Additionally to this, future work is planned to extend the gateway to large-scale and energy-aware mobile environments.

The last gateway examined in [14], introduces configurable smart gateway system which can be flexibly configured to adapt to different application requirements. It can also reduce the development cycle, difficulty, costs, and more easily and quickly be applied to new applications. To achieve this goal, the gateway uses its own message structure in combination with a pluggable design, whose modules with different communication protocols can be customized and plugged in according to different networks. The proposed result packages data on the gateway in its data format to overcome different data problems.

Additionally to design aspects of the system, packet scheduling is an essential function in most networks systems. In [5] this topic was further investigated. This work proposes an approach to include a classifier, which analyzes the messages received by the gateway and separates them into queues. The messages were divided into High Priority (HP) and Best Effort (BE). A modified version of the Hierarchical Token Bucket (HTB) algorithm was used, which aimed to achieve fair treatment of the requests. The experimental evaluation showed that high priority messages achieved a 24% reduction in latency without discards or losses of packages in this queue.

4 Design and Implementation of the proposed automatic Gateway

The design of the proposed gateway [23] is presented in more detail in the following sections. Only the chosen subset of problems and their solutions are considered in more detail. The Open Service Gateway Initiative (OSGi) [2] was used for the development, since this framework supports the implementation of component-based, service-oriented applications in Java [25]. For this reason, Java version 11 was also used for the first prototype. In addition, the software was developed specifically for the Raspberry Pi 3B+ as an evaluation system.

4.1 Overall Architecture

For reference, the overall architecture used to evaluate the prototype is shown in the following figure 1. However, the Raspberry Pi device (gateway) is particularly important for this article, as this is the target component to be developed. It runs an Apache Karaf [9] server (OSGi implementation), which also allows OSGi compatible software bundles to be installed at runtime. The following chapters will further explain the most important functionalities of the actual gateway. In addition, the test instruments are displayed. These include Digi XBee SMT Module (Xbee 3) [16], as well as Eclipse Mosquitto [10] and AWS IoT Core [17]. There are also lightweight test clients for each protocol used to simulate loads on the gateway.

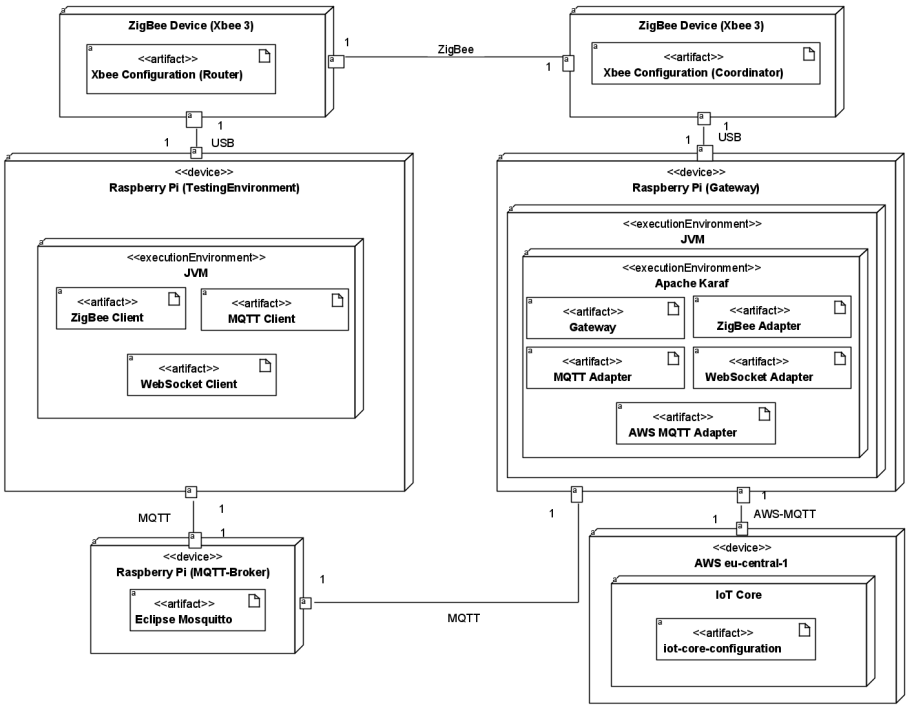

Fig. 1. Deployment diagram of the evaluation system.

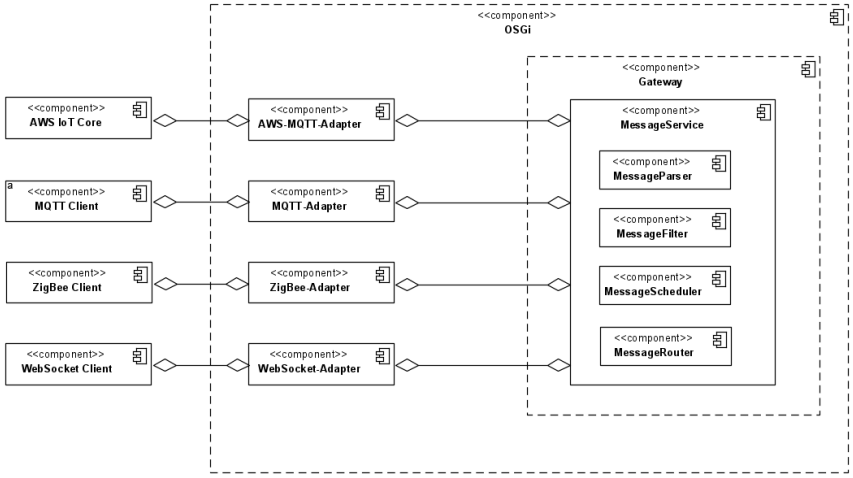

Fig. 2. OSGi container with provided software modules.

4.2 Protocol Conversion

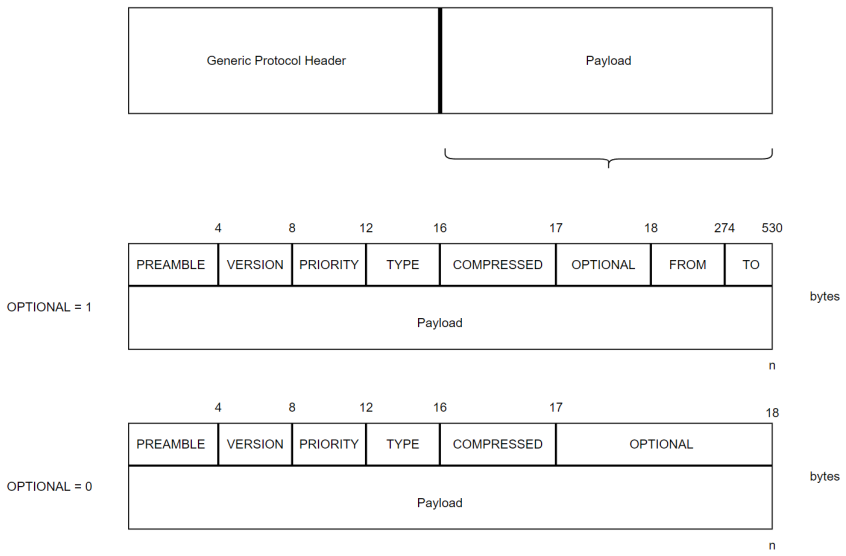

Fig. 3. Customized message format of the gateway.

Protocol conversion is one of the biggest problems in IoT, as there is no standardised interface for communication due to heterogeneity. Therefore, many manufacturers use their standards or fall back on one of the many freely available standards. In order to solve this problem, a modularised software environment is required, which can be changed at runtime, as well as a clearly defined interface so that the exchange of messages can be guaranteed via different protocols.

Based on these requirements, OSGi was used as an example to separate the logic of the gateway from the utilised protocols, as shown in figure 2. This makes it possible to add, configure or remove software packages for new protocols at runtime. These software modules are utilizing the adapter pattern, thus also called adapters.

The adapter-based approach makes it possible to react to existing requirements to embed new protocols or interfaces. However, this approach does not yet enable a cross-protocol interfaces. For this reason, this approach is complemented with a custom message format for IoT devices. The message format should enable supporting functionalities such as compression, prioritisation, and end-to-end encryption of messages. In addition, direct device addressing should be supported in any case. For this reason, an example message format as shown in figure 3 was implemented. This is only used as a proof-of-principle and can be exchanged with another message format, if necessary. The message format can

be used by clients, but it is not mandatory. If this is not the case, messages are interpreted as legacy and reformatted by the gateway to simply send them to a defined end consumer. This only works if the used protocol can supply enough metadata to transform this message into the uniform format. This metadata includes data such as IP addresses or names to uniquely identify the devices.

These two exemplary approaches show a way to solve the problem of protocol conversion. However, the data format used so far is still a rudimentary and not fully developed version. Therefore, further work has to be invested in this topic to create a possible general IoT standard message format for a comprehensive solution.

4.3 Device Management

Device management is a topic that increases in complexity with growing depth. For this reason, the proposed approach only provides a fundamental problem solution.

In the proposed approach, each gateway manages a database of all connected devices. From this, the gateway can detect whether devices have already been registered or not. When an unregistered device connects to the gateway, a registration process is automatically performed. To include other consumer systems, the gateway propagate this registration requests to other software modules responsible for protocols or end-systems like Amazon Web Services.

This simple solution shows the functional feasibility of the approach and allows devices to be registered in the cloud without much effort, since the adapter is managing the registration. This means that there is less need for configuration in the cloud itself. However, this approach is not covering a precise security mechanism yet. This shall be extended in the future work, because providing a suitable security mechanism is essential for the professional usage of this feature. In addition, further possibilities of configuration on the device side were not considered in this prototype. In order to enable this feature, this concept can be extended with user-predefined device configurations for specific device types. This can be done by extending the already existing message format. The last step would be to extend this concept to include a device discovery so that devices can be included in the communication without much effort.

4.4 Middleware Abstraction

Middleware abstraction is an additional problem besides the already existing heterogeniety of protocols. Nowadays, there are many cloud providers, and in some case, even several applications are used simultaneously. Moreover, these cloud providers always differ in their interfaces and partly include a vendor lock-in, so no consistent communication is possible.

Therefore, this paper proposes to supplement the already used adapter approach to solve this problem. The adapter allows the protocols and the cloud-specific interfaces to be translated. Of course, this presupposes that these can be covered by the message format and the internal communication of the gateway.

This simple approach is able to demonstrate what an accurate middleware abstraction can look like. Furthermore, due to the modular approach, extending the already presented solution does not require high effort.

4.5 Resource Management

In order for resource management to be automated, several parameters must be monitored. For this reason, the following metrics are measured in an interval.

- CPU utilisation of the system
- CPU utilisation of the process
- Heap memory usage
- Non-heap memory usage
- Throughput of messages

Based on the related work concentrating on this aspect, this is only a part of the metrics, but it can be measured without much effort and is providing enough initial information for estimation of the feasibility of our concept as an integrated solution. In addition, these measurements are used to prioritize messages and adjust QoS levels. Adjusting QoS levels is only possible by utilizing the unified message format, which is used to define such messages. This allowing, for example, to lower or raise the intervals of messages of clients. As a consequence, devices that do not implement this format can not be managed.

At principle, there are still various possibilities for expansion. For example, mechanisms could be built to increase the prioritisation of processes. For example, individual protocols could be prioritised if the load on the respective interface increases. For a network of individual gateways, these could even exchange information to distribute loads. These are only two examples, but the potential in this environment is still large.

4.6 Traffic Optimization

Traffic optimization is a very complex topic. There are many different approaches to increase throughput, and they all need to be evaluated and assessed for this use case. In this approach only simple methods were used to improve the system behaviour as a part of the solution integrating the different aspects.

For this purpose, the incoming messages are segmented into two queues. Then, the messages are placed in this queues based on priorities. The exact algorithm used is a simple version of Weighted-Fair-Queuing (WFQ) with fixed weights. In addition to scheduling, compression of messages was introduced. This is important for protocols like ZigBee, as the the rate of data transfer is only about 250 kbps [24].

Of course, as a future work, different algorithms need to be investigated and compared in order to determine the best suited ones. The prototype allows an exchange of the algorithm through adaptation of the configuration. This is possible without any problems due to the modular design.

5 Experimental Results

In the following section, we provide our test results and measurements of the gateway. The main objective of the measurements was to answer the following questions.

1. How much delay is caused by the system design, and can the system communicate in near real-time?
2. How many messages can the gateway handle simultaneously, and what impact does a high message count have on the gateway?
3. What impact the prioritisation of messages has on response time?
4. How long does the gateway run without errors, or more precisely, what are the mean time to failure (MTTF) and mean time between failures (MTBF)?

For this reason, loads were simulated on the MQTT and WebSocket protocols. Unfortunately, this was not possible for ZigBee since this protocol does not allow high data transfer rates and, therefore, only consumes low resources. In each measurement, the message had a size of 579 bytes in total and the measurement duration was one hour.

Table 1. Average measured values of the evaluation

Messages/s	CPU system	CPU process	Heap memory	Non Heap memory	Response Time	Test successfull
Baseline						
-	0.6%	0.35%	49.48 MB	56.23 MB	-	✓
MQTT						
1000	18%	18%	40.18 MB	61.94 MB	5.25 ms	✓
2000	27%	27%	76.05 MB	52.93 MB	5.9 ms	✓
4000	54%	54%	55.75 MB	52.95 MB	12.19 ms	✓
8000	43%	42%	168.92 MB	53 MB	212.36463 s	-
WebSocket						
1000	10%	8%	46.37 MB	55.08 MB	14.74 ms	✓
2000	17%	13%	47.43 MB	55.24 MB	15.67 ms	✓
4000	24%	19%	46.2 MB	55.08 MB	18.84 ms	✓
8000	43%	34%	45.48 MB	55.19 MB	91.82 ms	✓

The measurements have shown that a message load of 4000 messages per second can be considered acceptable, independent of the protocol. While even 8000 messages were still within the realm of possibility for WebSocket, this does not apply to MQTT. Nevertheless, the response time increased to almost 100 ms for WebSocket as well. Measurements with more messages per second could not be performed because the connections were constantly interrupted. In addition to the measured values, prioritising meassages improved the response time approximately 242%. For this, the response times between two clients with 2000 messages per second were compared. One of the two had a high priority and the other a normal priority. As a complement to the performance measurements, the runtime behaviour was monitored over two monts. No critical errors occured, therefore it can be assumed that the presented design runs considerably reliable. For productive use, extended time periods shall be considered.

6 Comparison to existing Frameworks

As an assessment of the added value, our proposed approach was compared with existing solutions from the economic sector. For this purpose, various open source projects were examined for protocol conversion, device management, middleware abstraction, resource management and traffic optimisation. The frameworks examined include Eclipse Kura, EdgeX Foundry, OpenHAB and ThingsBoard IoT Gateway.

Eclipse Kura is an extensible open-source IoT Edge Framework based on Java/OSGi. Kura offers API access to the hardware interfaces of IoT Gateways (serial ports, GPS, Watchdog, GPIO or I2C). It features ready-to-use field protocols (including Modbus or OPC-UA), an application container, and a web-based visual data flow programming to acquire data from the field, process it at the edge, and publish it to leading IoT Cloud Platforms through MQTT connectivity. While Eclipse Kura already partially solves the protocol conversion for devices, only MQTT is offered on the cloud side. In addition, only one device is created in the cloud, the gateway itself. However, the gateway can still call all endpoints and thus modify the data of other devices, but it cannot automatically register devices due to the implementation. There is no middleware abstraction either, as only the cloud platforms such as Eurotech Everyware Cloud, Amazon AWS IoT Core, Azure IoT Hub, Eclipse Kapua and Eclipse Hono are supported. At the moment, nothing can be found in the documentation about resource management. [12]

EdgeX Foundry is a highly flexible, scalable and vendor-neutral open-source framework hosted by the Linux Foundation that facilitates the development of data collection, analytics and cloud connector services. The applications act as cloud-to-edge middleware with a plug-and-play distributed microservice architecture. EdgeX Foundry was designed to provide broad industry support. For this reason, not only many device protocols are supported, but also different protocols on the cloud side. Furthermore, EdgeX Foundry's architecture enables middleware abstraction, so nobody depends on specific cloud providers. Also, the application supports resource management mechanisms by measuring parameters such as round trip time to achieve the lowest possible latency. In addition, EdgeX can identify which data needs to be sent to the cloud and processed cost-effectively on edge. However, there seem to be no prioritization options for individual packets. In device management, nevertheless, there is static and automatic device discovery, so devices no longer need to be added manually. [13]

The open Home Automation Bus (openHAB) is an open-source home automation software based on Java and OSGi. It is deployed on-premise and connects to devices and services from different vendors. As of 2019, openHAB already included around 300 bindings available as OSGi modules. While openHAB is a home automation software, that does not mean it cannot be used for IoT purposes. For example, support for different device protocols and services has been implemented, yet integrating different cloud applications is missing. In fact, openHAB only focuses on the openHAB Cloud. There seems to be no concrete resource management or traffic optimization mechanism either. Nevertheless,

the application has a device discovery function so that devices can be added automatically. [11]

The ThingsBoard IoT Gateway is an open-source solution to integrate devices with ThingsBoard. However, the IoT Gateway is built on top of Python and is different from similar projects that leverage OSGi technology. The gateway supports custom connectors to connect to new devices or servers and custom converters for processing data from devices. The ThingsBoard IoT Gateway also solves the protocol translation on the device side, yet it also lacks different cloud protocols. This fact leads to inevitably having to work with the ThingsBoard solution. Furthermore, there is no indication of resource management or a traffic optimization mechanism. However, the gateway can create devices automatically, so there is no need to create them manually. [3]

Comparing the summarized solutions with the one, developed within the proposed approach, we can notify that the developed solution offers significantly more modularity. The architecture design makes it possible to install different protocols and services at runtime. However, the most important processes such as scheduling are still controlled by the gateway. For this reason, points such as protocol conversion and middleware abstraction are fully implemented in the proposed design. Only EdgeX Foundry offers this modularity to the same extent. The device management of the proposed solution is not yet mature enough to be able to compare it with the other solutions, but enables including additional features such as the automatic change of QoS levels. Regarding resource management and traffic optimisation, the proposed solution and EdgeX Foundry are the only applications that implement these to a certain extent. However, it must be mentioned that in this area EdgeX Foundry implements other functionalities than our proposed gateway, so a comparison of these aspects is difficult. However, it should be noted that the proposed solution, in contrast to EdgeX Foundry, already includes prioritisation mechanisms. In summary, it can be stated that the developed solution is only a prototype and does not yet implement all the designed aspects into detail. Therefore, a comparison to other productive solutions cannot be done at this early stage of development. Nevertheless, it shows the first steps towards an integrated solution, which is obviously not the aim of the existing solutions.

7 Conclusion and Future Work

Internet of Things as an essential part of the Industry 4.0 strategy is a fast-growing area and is thus becoming increasingly important. Unfortunately, strong growth is also associated with many different problems. While a great deal of research has already been done on the topic of IoT in the fields of cloud computing, fog computing, and edge computing, little seems to have happened in the field of the gateways in particular.

A concept that can be called automatic gateway is required to solve these problems and minimize the manual intervention in order to recognize, add or remove devices and establish the communication link. In this paper we proposed an

integrated modular and extensible concept for enabling of an automatic protocol conversion, device management, middleware abstraction, resource management and traffic optimisation. In order to demonstrate the feasibility of the proposed approach, we implemented an initial prototype of the automatic gateway and evaluated the performance of this gateway for a Raspberry Pi 3B+. The measurements have shown that a load of 28.96 kbs (4000 messages/s with 579 bytes per message) up to 57.9 kbs (8000 messages/s with 579 bytes per message) can be considered acceptable. The prioritization of messages improved the response time by approximately 242% in this approach. In addition to the performance measurements the gateway was able to run over two months without critical errors. Therefore this approach can be considered feasible and reliable.

The proposed gateway provides a modular solution to solve the mentioned problems. The gateway can be easily extended to support additional protocols and consumers. Device management is also automated, reducing a significant administrative burden. Also, some aspects are supported to increase the scalability and robustness of the system. Nevertheless, the approach has to be considered as an initial feasibility study. In contrast to the approaches presented as related work, that are concentrating on one main aspect, our concept aims at providing an integrated extensible solution on conceptual and architectural level for the main problems addressed.

The proposed solution does not yet completely implement the individual subset of problems for the variety of protocols, data formats, devices and storage. In this context additional trade-offs considering feasibility vs. effort for each aspect have to be performed in order to determine the reasonable limits of such an integrated generic solution for a practical and productive use. The possibility of integrating the compared existing solutions will also be analysed and considered for further development. Additionally, the prototype needs to be further optimised to improve performance (to utilise the whole bandwidth of the Raspberry Pi 3B+). Therefore, existing traffic optimization approaches have to be considered and evaluated. Furthermore, the problem of security has to be addressed as a next step, since it is necessary for productive use.

Finally, an extension of the gateway in the context of the OPC/UA infrastructure under consideration of the UA Companion Specifications [22] is planned as a future work. In this context both the client/server approach and the enhancement to publish/subscribe [21] will be analysed.

References

1. Alam, T.: A reliable communication framework and its use in internet of things (iot). CSEIT1835111| Received **10**, 450–456 (2018)
2. OSGi Alliance: OSGi Core, Release 8. https://docs.osgi.org/specification/ (Oct 2020), accessed: 2022-21-06
3. ThingsBoard Authors: ThingsBoard IoT Gateway documentation. https://thingsboard.io/docs/iot-gateway/ (Aug 2022), accessed: 2022-08-04
4. Beniwal, G., Singhrova, A.: A systematic literature review on iot gateways. Journal of King Saud University-Computer and Information Sciences (2021)

5. de Caldas Filho, F.L., Rocha, R.L., Abbas, C.J., Martins, L.M.E., Canedo, E.D., de Sousa, R.T.: Qos scheduling algorithm for a fog iot gateway. In: 2019 Workshop on Communication Networks and Power Systems (WCNPS). pp. 1–6. IEEE (2019)
6. Desai, P., Sheth, A., Anantharam, P.: Semantic gateway as a service architecture for iot interoperability. In: 2015 IEEE International Conference on Mobile Services. pp. 313–319. IEEE (2015)
7. Fan, Q., Ansari, N.: Towards workload balancing in fog computing empowered iot. IEEE Transactions on Network Science and Engineering **7**(1), 253–262 (2018)
8. Filho, F.L.d.C., Martins, L.M.e., Araújo, I.P., Mendonça, F.L.d., da Costa, J.P.C., Júnior, R.T.d.S.: Design and evaluation of a semantic gateway prototype for IoT networks. In: Companion Proceedings of the10th International Conference on Utility and Cloud Computing. pp. 195–201 (2017)
9. The Apache Software Foundation: Apache Karaf container 4.x - documentation. https://karaf.apache.org/manual/latest/ (Aug 2022), accessed: 2022-08-04
10. Eclipse Foundation: Eclipse Mosquitto™. https://mosquitto.org/ (Aug 2022), accessed: 2022-08-04
11. openHAB Foundation: openHab documentation. https://www.openhab.org/docs/ (Aug 2022), accessed: 2022-08-04
12. Foundation, E.: The extensible open source java/osgi iot edge framework. https://www.eclipse.org/kura/ (August 2022), accessed: 2022-08-04
13. EdgeX Foundry: Edgex foundry documentation. https://docs.edgexfoundry.org/2.1/ (Aug 2022), accessed: 2022-08-04
14. Guoqiang, S., Yanming, C., Chao, Z., Yanxu, Z.: Design and implementation of a smart iot gateway. In: 2013 IEEE international conference on green computing and communications and IEEE internet of things and IEEE cyber, physical and social computing. pp. 720–723. IEEE (2013)
15. Henson, C.A., Pschorr, J.K., Sheth, A.P., Thirunarayan, K.: Semsos: Semantic sensor observation service. In: 2009 International Symposium on Collaborative Technologies and Systems. pp. 44–53 (2009). https://doi.org/10.1109/CTS.2009.5067461
16. Digi International Inc.: DigiXBee® 3 ZigBee®. https://www.digi.com/resources/documentation/digidocs/pdfs/90001539.pdf (Aug 2022), accessed: 2022-08-04
17. Amazon Web Services IoT Core: AWS IoT Core- Developer Guide. https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html (Aug 2022), accessed: 2022-08-04
18. Kang, B., Choo, H.: An experimental study of a reliable iot gateway. ICT Express **4**(3), 130–133 (2018)
19. Kang, B., Kim, D., Choo, H.: Internet of everything: A large-scale autonomic IoT gateway. IEEE Transactions on Multi-Scale Computing Systems **3**(3), 206–214 (2017)
20. Min, D., Xiao, Z., Sheng, B., Quanyong, H., Xuwei, P.: Design and implementation of heterogeneous IOT gateway based on dynamic priority scheduling algorithm. Transactions of the Institute of Measurement and Control **36**(7), 924–931 (2014)
21. OPC-Foundation: OPC UA is enhanced for Publish-Subscribe (Pub/Sub). https://opcconnect.opcfoundation.org/2016/03/opc-ua-is-enhanced-for-publish-subscribe-pubsub/ (2016), accessed: 2022-21-06
22. OPC-Foundation: OPC Unified Architecture (U/A) Specification. https://opcfoundation.org/developer-tools/specifications-unified-architecture (2017-2022), accessed: 2022-21-06

23. Soldin, M.: Design and implementation of an automated gateway prototype for the Internet of Things. Master's thesis, City University of Applied Sciences, Bremen, Germany (June 2022)
24. Somani, N.A., Patel, Y.: Zigbee: A low power wireless technology for industrial applications. International Journal of Control Theory and Computer Modelling (IJCTCM) **2**(3), 27–33 (2012)
25. Tavares, A.L., Valente, M.T.: A gentle introduction to OSGi. ACM SIGSOFT Software Engineering Notes **33**(5), 1–5 (2008)