Digital Twin Data Pipeline Using MQTT in SLADTA

C. Human, A. H. Basson(✉), and K. Kruger

Department of Mechanical and Mechatronic Engineering, Stellenbosch University, Stellenbosch 7600, South Africa
{humanc,ahb,kkruger}@sun.ac.za

Abstract. Digital twins, defined here as the virtual representations of real-world entities to facilitate their integration with digital systems, have become a popular concept in Industry 4.0. The Six-Layer Architecture for Digital Twins with Aggregation (SLADTA) is a framework for a digital twin suitable for complex systems with a large network of devices. An important part of this architecture is using an asynchronous, secure and vendor-neutral communication protocol suitable for a large network of devices. MQTT is here evaluated as a communication protocol for a digital twin data pipeline based of SLADTA. The evaluation includes a case study for a simulated heliostat field, using MQTT through Google Cloud Platform's (GCP) IoT Core broker and the accompanying Cloud Pub/Sub service. For comparison, the case study also uses MQTT through the Eclipse Mosquitto broker. The case study demonstrates the potential of MQTT in SLADTA for complex systems' digital twins.

Keywords: Digital Twin · MQTT · SLADTA

1 Introduction

The fourth industrial revolution (also known as Industry 4.0) has sparked increased interest in the concept of a digital twin. Various definitions for digital twins have been proposed, but in this paper a digital twin is taken to be a virtual representation of a real-world entity (the physical twin) to facilitate integration with digital systems [1]. Digital twins facilitate this integration through models in a virtual environment that are updated by sensor outputs so that the models reflect the state of the physical twin. According to some authors, a digital twin should also be able to influence the physical twin and, therefore, bi-directional communication is required [2]. Digital twins can satisfy various needs, such as [3–6]:

- Simulation and analysis based on real-time and historical data, to accurately predict system behaviour.
- Centralized and integrated data models that contain all relevant data to assist in decision making.
- Improved insight into system dependencies for enhanced future designs.

T. Borangiu et al. (Eds.): SOHOMA 2020, SCI 952, pp. 111–122, 2021.
https://doi.org/10.1007/978-3-030-69373-2_7

- Real-time remote monitoring of complex systems.
- Improved failure diagnostics and fault detection for system maintenance purposes.

Various frameworks for digital twins have been proposed, such as the 5C Architecture [7], the C2PS Architecture [8], the Digital Twin Shop Floor architecture [6] and the Six Layer Architecture for Digital Twins with Aggregation (SLADTA) [9]. SLADTA, which is outlined in Sect. 2, was chosen for the data pipeline described in this paper due to its general applicability, clear functional separation, vendor-neutral approach and provision for scaling and granularity through aggregation. These qualities make it an attractive architecture for complex systems.

Exchanging data and information between a physical twin and its corresponding digital twin, as well as between digital twins, requires a communication platform such as Message Queuing Telemetry Transport (MQTT), Advanced Message Queuing Protocol (AMQP), Constrained Application Protocol (CoAP), Hyper Text Transport Protocol (HTTP) or Open Process Control Unified Architecture (OPC UA). The choice depends on the application because some systems require highly configurable and reliable communication, while others prioritize speed or resource constraints [10].

CoAP is a lightweight protocol, designed for machine-to-machine communication and supports request/response communication, as well as resource/observe communication (similar to publish/subscribe). CoAP uses User Datagram Protocol (UDP), but also has functionality to improve reliability. AMQP is also a lightweight machine-to-machine communication protocol that supports request/response and publish/subscribe messaging. AMQP has a wide variety of features and was designed for reliability, security, provisioning (additional services) and interoperability. HTTP is predominantly a web messaging protocol that supports request/response Representational State Transfer (RESTful) Web communication [10].

MQTT, which was chosen for the data pipeline in this paper and is further discussed in Sect. 3, is a publish/subscribe messaging protocol that is suited to large networks of small devices, particularly if the devices have very limited resources or if network bandwidth is small. MQTT, compared to AMQP and HTTP, requires less bandwidth, has a lower latency and is generally more reliable. The drawback of these advantages is that MQTT has less built-in security, less provisioning (fewer additional services) and is not as standardized as the other protocols. Compared to CoAP, MQTT has higher latencies, requires more bandwidth, and uses more device resources. CoAP, which uses UDP, is more efficient than MQTT which uses TCP/IP. Despite this, MQTT is often preferred because it is much more reliable and only slightly less efficient, which makes MQTT more popular than CoAP. Due to its reliability and efficiency, MQTT is an attractive protocol to use for large networks (which aligns with SLADTA's target) of simple devices. AMQP and HTTP offer functionality that is often not required by small devices and therefore do not justify their larger message overhead and message size [10].

The objective of this paper is to evaluate MQTT as a communication protocol for use in a digital twin data pipeline, where the digital twin is based on SLADTA. MQTT is evaluated for communicating information to the cloud, as well as communicating information between digital twins. The findings about MQTT are expected to extend to other digital twin architectures too.

The remainder of this paper is structured as follows: Sects. 2 and 3 outline SLADTA and MQTT, respectively. Section 4 describes how a data pipeline can be setup using MQTT and SLADTA and Sect. 5 illustrates how the pipeline was implemented using a case study. Finally, Sect. 6 provides a conclusion.

2 The Six Layer Architecture for Digital Twins with Aggregation

The Six Layer Architecture for Digital Twins (SLADT) is a reference architecture for digital twin development and has been applied to a manufacturing cell for close to real-time monitoring and fault detection [11, 12]. SLADTA (the added A represents Aggregation) is an extension that includes twin-to-twin communication that facilitates system orientated decision making [9, 11]. Figure 1 illustrates SLADT and SLADTA.

SLADTA was designed as a general framework that is vendor-neutral and suitable for adding digital twins to existing systems. SLADTA can be considered for a wide variety of systems, including manufacturing cells, renewable energy systems, ships and building information models (BIM), all of which are being investigated by the authors' research group.

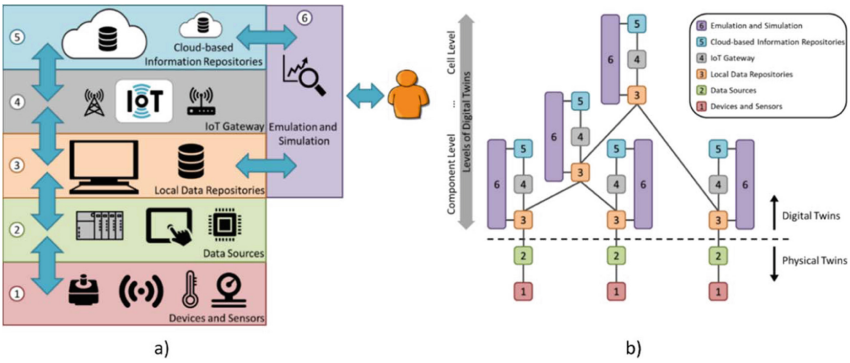


Fig. 1. a) SLADT framework. b) SLADTA. (*Adapted from* [11]).

The functions allocated by the architecture to its six layers are as follows: Layer 1 contains the *devices and sensors* that generate data, and Layer 2 contains *data sources* that interface with the sensors (for example controllers). Together the first two layers form the physical twin. Layer 3 contains *local data repositories*, which is part of the physical twin or, if necessary, added by the digital twin. Layer 4 is an *IoT Gateway*, which is a custom software application that manages the communication between physical twin and digital twin, and between digital twins. Layer 5 is a *cloud-based information repository* and, finally, Layer 6 is the *emulation and simulation* layer. The software used in Layer 6 is application specific, but is generally a data-endpoint and user interface.

Functionally, Layers 1 and 2 are responsible for data generation, Layers 3 and 4 are responsible for the data and information flow between a device and the cloud, as well as between different digital twins, and Layers 5 and 6 extract value from the information

being gathered. Further details about the functions of Layer 4 are given in Sect. 4 when discussing the pipeline.

The aggregation ability of SLADTA facilitates communication between digital twins, while limiting the data flows to what is necessary and what is compatible with privacy and confidentiality considerations. Higher level digital twins, i.e. aggregate digital twins, do not contain Layers 1 and 2, but aggregate the data of lower level digital twins for decision-making that requires information from multiple digital twins.

3 Message Queuing Telemetry Transport (MQTT)

As described in the Introduction, MQTT is a lightweight client-server, publish/subscribe messaging transport protocol designed for machine-to-machine communication and the Internet of Things (IoT). The protocol runs on TCP/IP, has a relatively small transport overhead and is suitable for devices that can maintain only a small code footprint or for devices that have a small network bandwidth. The standards applied to MQTT are developed by OASIS [13].

The MQTT server, usually referred to as the broker, acts as an intermediary between clients and facilitates publish/subscribe messaging. Its functions include authenticating and accepting connections, processing subscriptions, accepting published messages and forwarding published messages to subscribed clients. The client is responsible for opening the connection to the broker, publishing messages to defined topics and subscribing to topics of interest [13].

MQTT allows the user to select one of three *Quality of Service* (QoS) specifications for publishing and subscribing, respectively [13]:

- QoS 0: *At most deliver once*. The message is sent only once and no acknowledge message is required upon receiving the message. Therefore, network connectivity faults could result in a message not being received by the broker or the subscriber.
- QoS 1: *At least once delivery*. The receiver (either the broker or a subscriber) must acknowledge receipt, otherwise, the message is published again. Note that this is a non-blocking function and, therefore, the publisher can publish other messages while it is waiting for an acknowledge message.
- QoS2: *Exactly once delivery*. The receiver must acknowledge the published message and additional acknowledgement steps are applied to ensure that the message is not duplicated on the receiver's end. This ensures that no data is duplicated or lost, but the messages have additional overhead to ensure this.

To exchange information, a client connects to the broker, which usually includes some authentications of the client, and then the client can publish and subscribe to topics. Topics are dynamically created when a client publishes or subscribes to them and the topics can have a hierarchical structure, where a certain topic can have sub-topics. The forward slash '/' operator is used to separate the levels of a topic. Subscriptions can be made to either higher or lower level topics. For example, a topic *sensor* could have sub-topics, *sensor/temperature* and *sensor/humidity*. A subscription can then be made to *sensor* for both temperature and humidity readings or a subscription can be made to

only one of the sub-topics. The wildcard operator '#' may be used when subscribing to an unknown or unspecified sub-topic. The dollar topic prefix '$' is used to make subscriptions private and prevents wildcard subscriptions [13].

MQTT supports the use of various other application layer transport protocols to enhance its features, such as the *Transport Layer Security* (TLS) protocol and WebSockets. TLS, in particular, is used for security and TCP ports 8883 and 1883 are registered with Internet Assigned Numbers Authority (IANA) for MQTT TLS and MQTT non-TLS communication, respectively [13].

In terms of security, MQTT does not specify any security solutions since technology changes quite rapidly and the required security is situation dependent. That said, the MQTT documentation does provide general guidance to ensure communication security, such as [13]:

- Servers can authenticate clients by using the username and password field of the CONNECT packet. The implementation is situation dependent, but common practices include using the Lightweight Directory Access Protocol (LDAP), OAuth tokens or operating system authentication.
- It is good practice to hash text before sending it.
- Virtual Private Networks (VPN) can be used when available, to ensure better data security.
- Clients can authenticate servers by using the TLS certificate sent by the server when TLS is used.
- Normal messaging containing application-specific authentication information may be used to authenticate a server.

4 Data Pipeline Communication

4.1 Overview

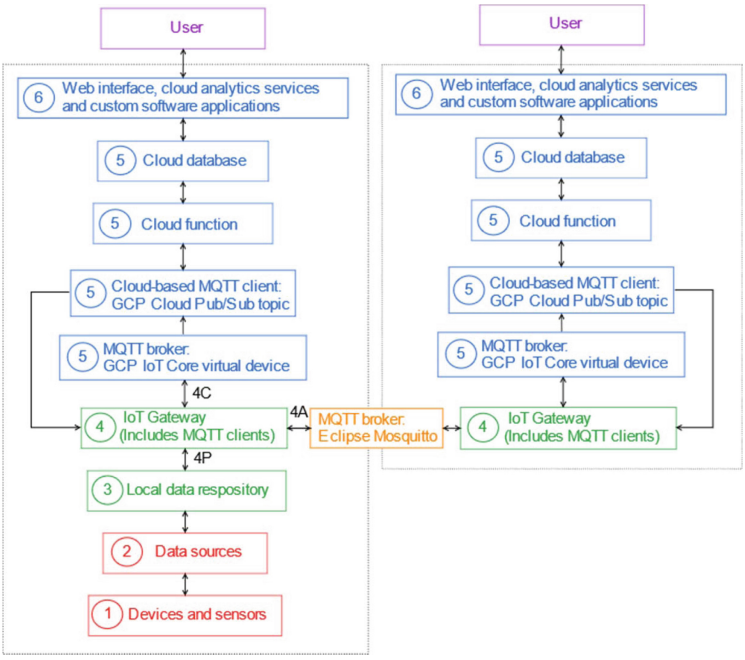
This section considers the communication within the data pipeline that connects data sources in a physical twin to data consuming applications, and in particular the parts of the communication where MQTT can play a significant role. Figure 2a illustrates the pipeline mapped onto SLADTA for a lower level digital twin and Fig. 2b for an aggregate digital twin. Google Cloud Platform (GCP) is used here for cloud storage and related services (in blue in Fig. 2). GCP was chosen as a matter of convenience and other cloud services also should be suitable, although the implementation details may differ.

The key parts of the data pipeline where MQTT plays an important role are Layer 4 (the IoT Gateway) and the MQTT broker and client parts of Layer 5. It is unlikely that MQTT will play a significant role in Layers 1 and 2 (the physical twin). Layer 3 (the local data repository) may use MQTT for communication with Layer 4, but that will be similar to the other MQTT communication in the digital twin and is therefore not considered in detail here. The parts of Layer 5 not relevant to MQTT, i.e. the cloud data storage and processing, and Layer 6 (custom cloud-based software) are also not considered in detail here.

It should be noted that twin-to-twin communication in Fig. 2 is through the respective Layers 4, and not Layers 3 as in Fig. 1(b). In [11], Layer 3 played a dual role of storing

data and providing a communication means, since it was an OPC UA server. In general, however, the required communication functions can be achieved through MQTT without involving Layer 3.

The following subsections first consider the requirements for Layer 4 and then MQTT's use in the data pipeline as a means of secure asynchronous communication.



(a) Lower level digital twin (b) Aggregate digital twin

Fig. 2. The digital twin data pipeline mapped onto the SLADTA framework

4.2 Layer 4: IoT Gateway

Layer 4 plays a key role in the data pipeline and in the communication where MQTT is most relevant. The functions performed by Layer 4 are therefore considered here. Some overall requirements for digital twins in a complex system also should be noted:

- The design should allow for the addition and removal of physical twins to/from the system, with their associated digital twins, with minimal reconfiguration time and effort, because complex systems are likely to change over time.
- The design should allow for the communication between digital twins, which can include aggregation and/or peer-to-peer interaction, because in complex systems the various digital twins in the system may be developed by different parties and information may be shared on a selective basis.

Layer 4 acts as gateway for information flows between the physical twin and the virtual world, and between its digital twin and other digital twins. This layer's functions are [11]:

- Transform the data received from the physical twin to information as required by the context (e.g. perform unit conversions or convert information to forms more useful for the higher layers).
- Select or transform the information to be transmitted to the cloud repositories to avoid excessive database requirements and bandwidth bottlenecks.
- Direct the information flows to the different data repositories on Layer 5 and other digital twins, taking into account that different users of the digital twin may have access to different subsets of the information.
- Check all information received from the cloud repositories and other digital twins, to ensure safety, consistency and compatibility with the physical twin. Where appropriate, communicate the relevant data to the physical twin.

From the data pipeline perspective, the main functions that this layer performs are:

- Structure the data being gathered into a format that is suitable for messaging.
- Structure and process the data so that it can be used and manipulated in other software applications.
- Interface with the Layers 3 and 5, as well as with other digital twins.

Layer 4 typically will be is a custom software application developed by persons that have detailed knowledge of the associated physical twin. This application can be developed in one of many programming languages, provided that the language has support for the chosen communication means. For MQTT, client libraries are available for C, C++, Java, Node.js, C#/.NET, Python, PHP and more. Each digital twin must have one or more MQTT client and must be able to set up the client(s) according to the chosen authentication and security requirements. Layer 4 may possibly also host an MQTT broker for twin-to-twin communication, as outlined in the next section.

4.3 Secure Asynchronous Communication

In the previous work on SLADTA [11], as outlined in Sect. 2, Layer 3 was named "local data repositories", but included communication functions. Upon closer interrogation of the previous uses of SLADTA, the combination of the repository and communication functions in Layer 3 was found to be incidental, because of the choice of OPC UA as the means of storage and communication. Also, the previous SLADTA research did not identify the means of communication between Layers 4 and 5 as a distinct function.

Redelinghuys [11] divided Layer 3 into Sublayers 3P and 3A, where 3P ("P" for physical) is used to exchange data between the physical twin (Layer 2) and Layer 4 of its digital twin, while 3A ("A" for aggregation) provides information transfer between digital twins. In this paper, the nomenclature is changed to 4P and 4A, because it is more general to allocate the communication functions to Layer 4. Further, 4C ("C" for cloud)

is added, to denote the communication between Layers 4 and 5. The names 4P, 4A and 4C denote all the communication passing through Layer 4, the IoT Gateway.

Although 4P, 4A and 4C can use different communication platforms, their primary functional requirements are similar: they must provide (1) secure, (2) asynchronous communication in a (3) vendor neutral format. Firstly, secure communication is required to protect the proprietary information being exchanged and to prevent malicious interference with the data pipeline. Secondly, asynchronous communication supports the overarching objectives of easy reconfiguration and aggregation. If synchronous communication is used, more knowledge about the communicating partners is required when developing a digital twin's IoT Gateway and unexpected consequences can arise if one communication path unexpectedly blocks another path's communication. In complex systems, such emergent behaviour is best avoided. Finally, vendor neutral communication allows interconnections with physical twin elements, other digital twins and cloud repositories from different vendors. It also supports reconfiguration and aggregation.

MQTT satisfies the above three functional requirements. Using "off-the-shelf" software and libraries often will be preferable because the people developing Layer 4 (a custom application that requires specialist knowledge of the physical twin) usually will not be specialists in developing this level of communication from scratch.

4.4 MQTT Using Google Cloud Platform vs Eclipse Mosquitto

As mentioned in Sect. 3, a MQTT broker is required to use MQTT. In Fig. 2, GCP's IoT Core along with GCP Cloud Pub/Sub provide a MQTT broker for 4C. CGP's services or an alternative MQTT broker that is not cloud based, such as the Eclipse Mosquitto broker, may be considered for 4A, as shown in Fig. 2. This section highlights some key differences between the two MQTT platforms.

Using the MQTT broker of the cloud service chosen for Layer 5 to facilitate 4C allows for easy integration, but GCP does not comply with some MQTT standards (presumably for security and performance reasons). Particularly relevant here are dynamic creation of topics and subscriptions by a client to data published by other clients to the MQTT broker.

In GCP, each MQTT client (and therefore every digital twin) has to be associated with an IoT Core virtual *device* and each virtual device is contained within an associated *registry*. Each registry must be linked to predefined topics in Cloud Pub/Sub, which is the main messaging service within GCP. The MQTT client can then publish to the "events" topic of the virtual counterpart of the device and the virtual device will publish to the registry's linked Pub/Sub topic(s). All IoT Core virtual devices within a registry can publish to the same Pub/Sub topic(s). Dynamic creation of MQTT topics are therefore not possible in the GCP MQTT implementation.

In GCP it is not possible to subscribe directly to the MQTT broker, as in conventional MQTT. For a MQTT client to subscribe to a topic, a Cloud Pub/Sub subscription must first be created and then the cloud Pub/Sub client library must be used for a Cloud Pub/Sub subscription that is linked to the topic of interest.

The publish and subscribe paths in GCP do have advantages, e.g. the IoT Core virtual device that is associated with an MQTT client can be used to check whether an MQTT client is connected, when last it had a heartbeat and had sent telemetry.

Communication using MQTT through GCP is the obvious route for 4C when Layer 5 uses GCP. For 4A, it may also be preferable to use MQTT through GCP due to its security measures and additional services. An alternative is a conventional MQTT broker such as Eclipse's Mosquitto broker, with much more straight forward publish and subscribe operations. With a broker such as the Mosquitto one, the developer has more responsibilities than in GCP to implement security measures. For example, authentication of the client and broker must be configured and there are various options. The easiest is no authentication in which case the broker will use its default configurations. Authentication of the broker using TLS protocol includes the considerations:

- The appropriate connection port must be specified (typically 8883).
- The broker must not allow anonymous connections.
- The file paths for the certificate authority (CA) security certificate, the broker security key and its security certificate must be provided in the configuration file.
- The client must be provided with the CA security certificate.

To authenticate a client, the broker can apply username and password authentication, the TLS protocol or both. For username and password authentication, the mosquito_passwd utility function (provided with the broker) is used to create a text file containing valid username and password pairs. To authenticate the client using the TLS protocol, the client must be provided with a security key and a security certificate in addition to the CA security certificate. The broker must then be configured to request a security certificate from the client during the authentication process. It is also possible to configure the Mosquitto broker to listen on various ports and apply different types of authentication on the different ports.

Authorization to access certain topics can also be specified for certain usernames by creating an access control list (ACL) file and specifying the path of this file in the configuration file of the broker. Mosquitto brokers can further be connected using their bridging function and then selected topics can be shared between brokers as specified in the configuration file of each broker.

Irrespective of the MQTT broker chosen, the digital twin's MQTT clients that connect to the MQTT broker(s) are set up and configured in Layer 4 of the digital twin.

5 Case Study Implementation

A heliostat field was chosen as the case study for this data pipeline because it is a large collection of devices that transmit sensor data and receive configuration settings. The complexity of the pipeline derives from the requirement to structure the large amount of data so that it can be utilized in the cloud. For this case study, an illustrative small implementation is used, with three digital twins that each have a physical twin (referred to further as lower level digital twins) and one aggregate digital twin.

5.1 Layers 1, 2 and 3: Heliostat Sensors, Cluster Controllers and Database

Each digital twin represents a cluster of 24 heliostats. Each heliostat in the cluster (Level 1) sends the following data to one higher-level controller (Level 2): a heliostat ID number,

a battery voltage level, two stepper motor positions, a timestamp, a status value and a target position (as calculated by the Grena algorithm). The higher-level controller is connected to an Ethernet network and sends its data to a PostgreSQL database on Layer 3. The controller for the whole heliostat field interfaces with this database. For the case study evaluation, Layers 1 and 2 were simulated in software that sends the appropriate data to Layer 3. The simulation of Layers 1 and 2 is transparent to the higher layers of the digital twin, but gives the opportunity to easily experiment with, e.g., data rates.

5.2 Layer 4: IoT Gateway

The IoT Gateway is a Python program which includes a configuration file, a setup part, a data processing part, a data source interaction part and an MQTT client part.

The configuration file contains information such as the file path of the RSA private key for authentication purposes. The setup part coordinates the use of the various parts. For the lower level digital twins, this means creating the MQTT clients for 4C and 4A and establishing communication 4P with the PostgreSQL database.

For the lower level digital twins, the data processing part includes converting an SQL query to Python dictionary format, the dictionary format to JSON format or the dictionary format to a string data type. To publish messages to GCP's IoT Core, the data being sent must be a byte object and not a string object, and therefore must be encoded to 8-bit Unicode Transformation Format (UTF-8).

For this case study, the aggregate digital twin's data processing part additionally assesses various heliostat parameters and then publishes the status of a pod (a collection of six heliostats) to the cloud.

The data source interaction part of the IoT Gateway was implemented as follows: For 4P, an SQL database interface using the psycopg2 library in Python is used. For 4A and 4C communication, respective MQTT clients were created using the Eclipse Paho MQTT library. The 4A client subscribes to the Mosquitto broker, but 4C has to use GCP's Cloud Pub/Sub client library to subscribe.

To set up the MQTT client for 4C and 4A, three general steps must be followed: the client authentication and the relevant call-back functions must be specified, and then the client should connect. Thereafter, the client can publish and subscribe as necessary. Authentication of the client with Google Cloud IoT Core's broker (for 4C) has three parts to it: a JSON Web Token (JWT) must be configured, the token must be signed with a Rivest-Shamir-Adleman (RSA) or elliptical-curve (EC) private key, and the TLS certificate must be provided.

For 4A in this case study, the broker was authenticated by the client using the TLS protocol and the client was authenticated by the broker using the TLS protocol together with username and password authentication. The OpenSSL toolkit was used to generate the appropriate security keys and certificates for the TLS communication, as recommended by the Mosquitto documentation. The authentication described above for both brokers was setup using the Paho MQTT client library.

Preliminary results indicate that the Mosquitto broker, running on the local network, has lower round-trip latencies to the aggregate than the GCP broker. These latencies are also influenced by the message frequency and size, but message frequency seems to

have a larger influence. Further tests are being done to determine the number of digital twins that can be aggregated effectively within the case study environment for various scenarios.

6 Conclusion

This paper presents a digital twin data pipeline that is built on SLADTA and uses MQTT. SLADTA provides a framework to implement digital twins for complex systems. MQTT is a communication protocol that is suited to large networks of small devices and is well suited as the protocol for communication with the twin's cloud repository and with other digital twins.

To demonstrate the functionality of SLADTA with MQTT, a case study of a heliostat field was chosen because a heliostat field is a large collection of simple devices. When using the MQTT broker provided within Google's IoT Core service and the accompanying Cloud Pub/Sub service, as well as the Mosquitto broker, MQTT was found to be well suited to the tasks of asynchronous communication and aggregation within SLADTA. In the case study the GCP IoT Core broker was used for communication between the digital twin's IoT Gateway and the cloud, while the Mosquitto broker was used for communication between pairs of digital twins, through their respective IoT Gateways. The choice of broker is situation dependent since both have certain benefits and drawbacks.

The case study demonstrated that MQTT can effectively provide vendor neutral, secure, asynchronous communication that complements SLADTA's use for digital twins in complex systems.

Further research is required to explore the advantages and limitations of the combination of SLADTA and MQTT, such as cases with a large number of digital twins. Additionally, the use of other cloud services on Layers 5 and 6 also need to be researched. A more extensive comparison of the GCP broker vs the Mosquitto broker, particularly regarding security and latency, would be valuable.

Acknowledgements. This work was funded by the Horizon 2020 PREMA project. The project investigates concentrated solar thermal (CST) power, inter alia, for pre-heating of manganese ferroalloys to save energy and reduce CO2 emissions. Industry 4.0 technologies and concepts can play an important role, as it can increase the level of automation and improve the reliability of CST plants.

References

1. Taylor, N., Human, C., Kruger, K., Bekker, A., Basson, A.: Comparison of digital twin development in manufacturing and maritime domains. In: Borangiu, T., Trentesaux, D., Leitao, P., Boggino, A.G., Botti, V. (eds.) Springer Service Oriented, Holonic and Multi-agent Manufacturing Systems for Industry of the Future. Studies in Computational Intelligence, vol 853, pp. 158–170. Springer, Cham (2020)
2. Kritzinger, W., Traar, G., Henjes, J., Sihn, W., Karner, M.: Digital twin in manufacturing: a categorical literature review and classification. In: Proceedings of the 16th IFAC Symposium on Information Control Problems in Manufacturing INCOM 2018. IFAC-PapersOnLine, vol. 51, pp. 1016–1022. Elsevier B.V. (2018)

3. Bao, J., Guo, D., Li, J., Zhang, J.: The modelling and operations for the digital twin in the context of manufacturing. Enterprise Inf. Syst. **13**(4), 534–556 (2018)
4. Grieves, M., Vickers, J.: Digital twin: mitigating unpredictable, undesirable emergent behavior in complex systems. In: Kahlen, F.-J., Flumerfelt, S., Alves, A. (eds.) Trans-Disciplinary Perspectives on Complex Systems: New Findings and Approaches, pp. 85–113. Springer, Cham (2017)
5. Siemens AG: MindSphere the cloud-based, open IoT operating system for digital transformation. https://www.siemens.com/mindsphere https://www.plm.automation.siemens.com/media/global/en/Siemens_MindSphere_Whitepaper_tcm27-9395.pdf. Accessed 16 July 2019
6. Tao, F., Cheng, J., Qi, Q., Zhang, M., Zhang, H., Sui, F.: Digital twin-driven product design, manufacturing and service with big data. Int. J. Adv. Manuf. Technol. **94**, 3563–3576 (2018)
7. Roy, R., Tiwari, A., Stark, R., Lee, J.: Predictive big data analytics and cyber physical systems for TES systems. In: Redding, L., Roy, R., Shaw, A. (eds.) Advances in Through-Life Engineering Services, Decision Engineering. Springer, Cham (2017)
8. Alam, K.M., El Saddik, A.: C2PS: a digital twin architecture reference model for the cloud-based cyber-physical systems. IEEE Access **5**, 2050–2062 (2017)
9. Redelinghuys, A.J.H., Basson, A.H., Kruger, K.: A six-layer architecture for digital twins with aggregation. In: Borangiu, T., Trentesaux, D., Leitao, P., Boggino, A.G., Botti, V. (eds.) Springer Service Oriented, Holonic and Multi-agent Manufacturing Systems for Industry of the Future. Studies in Computational Intelligence, vol. 853, pp. 171–182. Springer, Cham (2020)
10. Naik, N.: Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. In: IEEE International Systems Engineering Symposium (ISSE), Vienna, Austria, pp. 1–7. IEEE (2017)
11. Redelinghuys, A.J.H.: An architecture for the digital twin of a manufacturing cell. Published Doctoral dissertation, Stellenbosch University, Stellenbosch (2019)
12. Redelinghuys, A.J.H., Basson, A.H., Kruger, K.: A six - layer architecture for the digital twin: a manufacturing case study implementation. J. Intell. Manuf. 1–20 (2019)
13. Banks, A., Briggs, E., Borgendale, K., Gupta, R.: MQTT Version 5.0. https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html. Accessed 14 Oct 2019