Ref. Ares(2024)8514389 - 29/11/2024

EUROPEAN COMMISSION

HORIZON EUROPE PROGRAMME – TOPIC: HORIZON-CL5-2022-D2-01

FASTEST

Fast-track hybrid testing platform for the development of battery systems

Deliverable D6.5: Real-time connection of physical and virtual bench

Primary Author [Dr. Shuchen Liu]

Organization [FEV]

Date: [29.11.2024]

Doc. Version: [V1.0]

Funded by the European Union

UK Research and Innovation

Co-funded by the European Union and UKRI under grant agreements N° 101103755 and 10078013, respectively. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Climate, Infrastructure and Environment Executive Agency (CINEA). Neither the European Union nor CINEA can be held responsible for them

Document Control Information	
Settings	**Value**
Work package:	6 – Development of hybrid testing platform
Deliverable:	Real-time connection of physical and virtual benches
Deliverable Type:	R – Document, report
Dissemination Level:	PU - Public
Due Date:	30.11.2024 (Month 18)
Actual Submission Date:	29.11.2024
Pages:	< 24 >
Doc. Version:	V1.0
GA Number:	101103755
Project Coordinator:	Bruno Rodrigues | ABEE (bruno.rodrigues@abeegroup.com)

Formal **Reviewers**		
Name	Organization	Date
Naqeeb Tahasildar	**BMZ**	10.11.2024
Laura Oca	**MGEP**	25.11.2024

Document History			
Version	Date	Description	Author
0.9	30.10.2024	First Draft	Dr. Shuchen Liu (FEV) Murat Bayraktar (FEV) Bruno Rodrigues (ABEE) Antonio Paolo Passaro (COMAU) Walter Verdonck (FLANDERSMAKE) Ahu Ece Hartavi(Surrey) Mohammad Ghazali(Surrey) Batuhan Cinar(Surrey)
1.0	29.11.2024	Initial Submission	Dr. Shuchen Liu (FEV) Murat Bayraktar (FEV)

			Bruno Rodrigues (ABEE) Antonio Paolo Passaro (COMAU) Walter Verdonck (FLANDERSMAKE) Ahu Ece Hartavi(Surrey) Mohammad Ghazali(Surrey) Batuhan Cinar(Surrey)

Project Abstract

Current methods to evaluate Li-ion batteries safety, performance, reliability and lifetime represent a remarkable resource consumption for the overall battery R&D process. The time or number of tests required, the expensive equipment and a generalized trial-error approach are determining factors, together with a lack of understanding of the complex multiscale and multi-physics phenomena in the battery system. Besides, testing facilities are operated locally, meaning that data management is handled directly in the facility, and that experimentation is done on one test bench.

The FASTEST project aims develop and validate a fast-track testing platform able to deliver a strategy based on Design of Experiments (DoE) and robust testing results, combining multi-scale and multi-physics virtual and physical testing. This will enable an accelerated battery system R&D and more reliable, safer and long-lasting battery system designs. The project's prototype of a fast-track hybrid testing platform aims for a new holistic and interconnected approach. From a global test facility perspective, additional services like smart DoE algorithms, virtualized benches, and digital twin (DT) data are incorporated into the daily facility operation to reach a new level of efficiency.

During the project, FASTEST consortium aims to develop up to TRL 6 the platform and its components: the optimal DoE strategies according to three different use cases (automotive, stationary, and off-road); two different cell chemistries, 3b and 4 solid-state (oxide polymer electrolyte); the development of a complete set of physics-based and data driven models able to substitute physical characterization experiments; and the overarching Digital Twin architecture managing the information flows, and the TRL 6 proven and integrated prototype of the hybrid testing platform.

LIST OF ABBREVIATIONS, ACRONYMS AND DEFINITIONS

Acronym	Name
AKS	Azure Kubernetes Services
ACR	Azure Container Services
POC	Proof Of Concept
LIMS	Laboratory Inventory Management System
MQTT	Message Queuing Telemetry Transport
VPC	Virtual Private Cloud
DT	Digital Twin
DoE	Design of Experiment
UUT	Unit Under Test
UUID	Unit under test ID
SFTP	Secure File Transfer Protocol
IoT	Internet of things
MQTT	Message Queuing Telemetry Transport
UI	User Interface
XiL	X in the loop
IP	Internet Protocol

LIST OF TABLES

LIST OF FIGURES

Table of Contents

1. EXECUTIVE SUMMARY

Deliverable D6.5 focuses on the conceptual work, implementation, stabilisation, and testing of an "as fast as possible" connection between physical and virtual test benches in the cloud. The technical requirements needed to achieve this are analysed and listed in this deliverable in accordance with the proposed cloud solution. This deliverable presents the implementation results of the proof-of-concept (POC) connection with the partners, laying the groundwork for the next steps in integrating both virtual and physical test benches with the developed components.

In the current testing facility landscape, operations are often managed locally, with data management and storage handled directly within the facility. The Laboratory Inventory Management System (LIMS) is introduced as a central component of the FASTEST project to streamline the management of testing facilities. The efficient use of available resources is highlighted as crucial for overall efficiency improvement.

In this deliverable, the communication workflow between LIMS, DoE and DT is described. LIMS serves as the main communication hub that connects both the physical and virtual test benches with DoE and DT. This communication is done throughMessage Queuing Telemetry Transport (MQTT) for its advantages on Internet of things (IoT) use cases. Two of the main advantages are its light weight and high reliability on message delivery. The deployed MQTT broker mainly uses two types of message delivery, one-time single messages for control purposes and continuous real-time messages for test observation purposes. The MQTT broker is deployed on a cluster in Azure Kubernetes services for high availability purposes. The broker is reachable via an external service inside the cluster where partners from DoE and DT can connect to subscribe/publish messages through this services Internet Protocol (IP). The communication loop starts from LIMS user interface (UI) where users choose a test type, preferred test date and targeted battery type. LIMS acts as a publisher and a subscriber to respectively propagate this information and receive the response from DT and DoE accordingly. To facilitate a streamlined communication between the broker from FEV and the broker from DT, broker bridging takes place. This improves the resource management where each broker focuses on specific clients. It also allows cross-network communication where each broker is deployed in its own network and message exchange is required.

With the described implementation, messages can be exchanged securely between LIMS, DT, DoE and the test benches in the form of commands or real-time telemetry.

2. OBJECTIVES

The FASTEST project aims to develop and validate a fast-track testing platform able to deliver a strategy based on Design of Experiments (DoE) and robust testing results, combining multi-scale and multi-physics virtual and physical testing. This will enable an accelerated battery system R&D and more reliable, safer, and long-lasting battery system designs. The project's prototype of a fast-track hybrid testing platform aims for a new holistic and interconnected approach. From a global test facility perspective, additional services like smart DoE algorithms, virtualised benches, and DT data are incorporated into the daily facility operation to reach a new level of efficiency.

During the project, FASTEST consortium aims to develop up to TRL6 the platform and its components: the optimal DoE strategies according to three different use cases (automotive, stationary, and off-road); the development of a complete set of physics-based and data-driven models are able to substitute physical characterisation experiments; the overarching Digital Twin (DT) architecture managing the information flows, and the TRL6 proven and integrated prototype of the hybrid testing platform. The platform, aimed to become a flexible tool for any chemistry and application. One of which is, including the predictive maintenance algorithm, developed in WP3, which aims to estimate the remaining useful life (RUL) of a battery, taking into account various negative test scenarios. These scenarios are categorized into three main sections: mechanical, electrical, and thermal abuse. Specific conditions include battery casing penetration, internal and external short circuits, state-of-charge (SoC) calibration errors leading to rapid degradation from overcharge or over discharge, and thermal system failures resulting in internal and external heat exposure. By simulating these extreme operating conditions, the tests ensure that the algorithm can effectively handle rare battery failures. Based on the outcomes of these negative case tests, the system will determine whether a test should be conducted on the virtual test bench or the physical test bench, as part of WP2 and incorporation with LIMS.

Purpose of this work task T6.5 is the concept work, implementation, stabilization, testing and real-life validation of an "as fast as possible"-connection between physical and virtual test bench in the cloud. Therefore, first matching and feasible technologies must be investigated. It will be determined which possibilities are existing to perform fast continuous data transfer between virtual and real bench. After identification of matching base technology, a respective module design and prototype implementation will follow. Subsequently to a POC connection within a selected example scenario, the final module will be implemented and tested. On some use-cases hard real-time requirements will be necessary. In that case it might be necessary to download the models into the test centre IT infrastructure to overcome the shortcoming of internet communication connection bandwidth and latency (more comparable to an X-in-the-loop (XiL)-like approach). Finally, the concepts and module's limitations need to be carefully investigated and documented transparently.

The objectives of this work task are summarized as follows:

- Choosing the required cloud services to match the technical requirements
- Designing an efficient and secure cloud infrastructure setup
- Development and deployment of LIMS software to serve as the central communication hub between the project’s components
- Establish real-time communication between LIMS and the relative partners from DT, DoE, and test benches

3. INTRODUCTION

The following chapter focuses on laying the foundation of the project from a technical overview. First, the purpose of the test benches is explained, followed up by the problem statement, technical requirements and the proposed solution in the form of a cloud architecture design. Detailed descriptions on the deployment method of each service used in our architecture supports the context mentioned.

3.1 Purpose of conceptualizing virtual and physical tests

The purpose of conceptualizing the virtual and physical tests is to establish an understanding of how these tests interact with the proposed solution’s components. After the design phase comes the implementation phase where the concept is tested and iteratively enhanced.

3.2 Problem description

In this working package, a central communication hub between the test benches (physical/virtual) and the other FASTEST system components (DoE/DT) is to be developed. The software interaction with DT and DoE is summarized as follows. It is meant to fetch the optimized experiment parameters from the DoE and fetch the correct model file from the DT model registry and forward it to the virtual test bench. The communication needs to be in a near real-time basis due to the nature of the tests being run on both physical and virtual benches. Also, the introduced message delay is of an exceptional importance due to the usage of an optimization algorithm that chooses the best suiting timing of each test on each bench. This software is called LIMS and the communication technology it uses is called MQTT.

3.3 Requirements

The described problem requires the followings:

- Maximum communication delay tolerance in the scope of seconds
- Logging communication and test results in a scalable database
- Secure deployment and communication between the database and LIMS
- Scalable MQTT broker deployment, as the expected total clients may increase according to partners, currently total of 5 clients in total are to be expected.
- Portable deployment regardless of the chosen cloud provider
- QoS level 2 to ensure high reliability of receiving messages exactly as planned

4. DESCRIPTION OF LIMS ARCHITECTURE

In Figure 1, a concept of FEV cloud infrastructure is introduced, which is deployed in Microsoft Azure and connected to the digital twin and physical test benches. FEV Azure infrastructure provide the interface for the user to access the FASTEST system and operation platform for many FASTEST components, e.g., LIMS including the scheduling algorithm, DoE and virtual test benches.

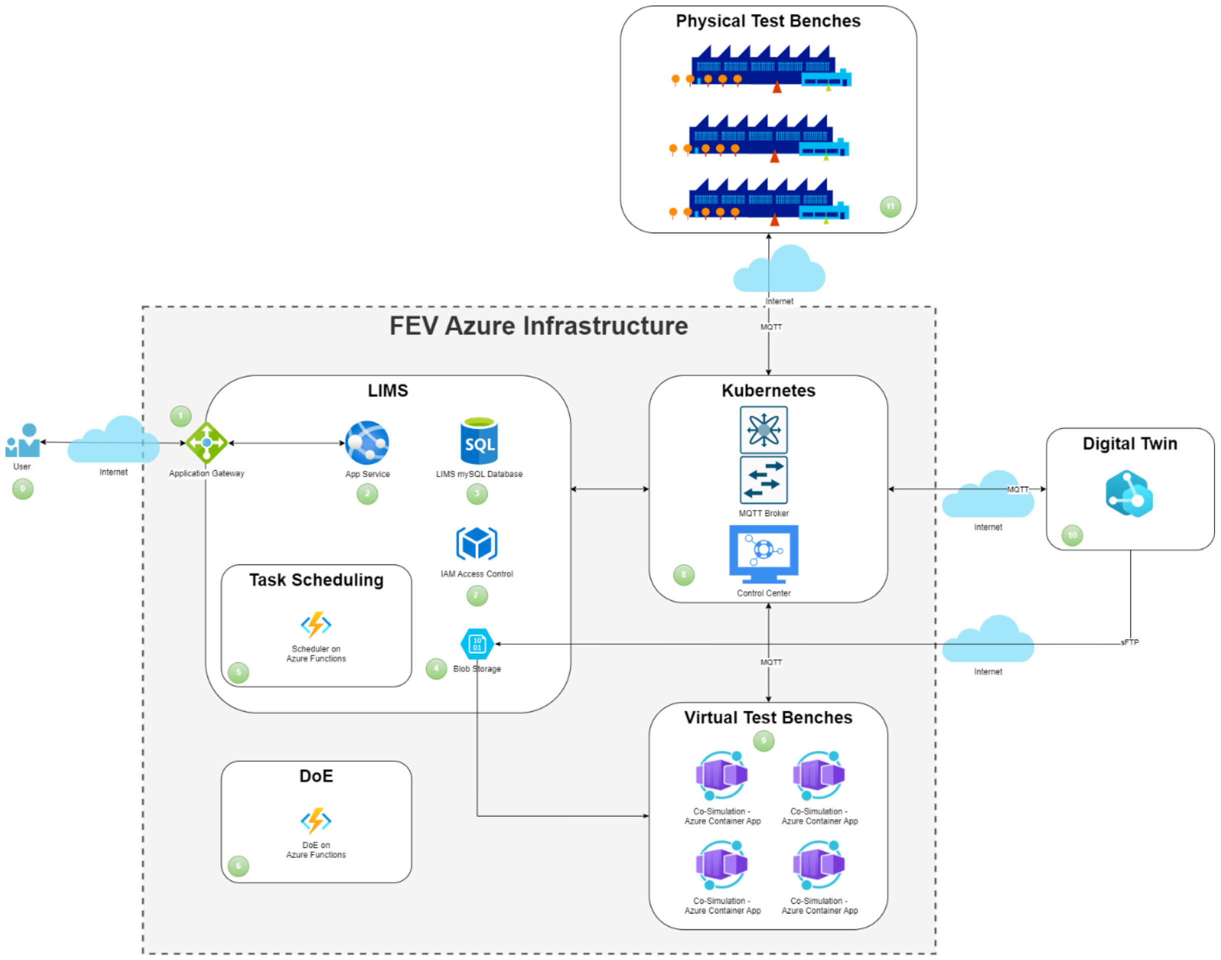


Figure 1 LIMS Azure Infrastructure for connecting virtual and physical test benches, DoE and Digital Twin

Our Virtual Private Cloud (VPC) will be responsible for securely deploying four main applications namely (LIMS / MQTT broker / Scheduling algorithm / virtual test benches). Each of these applications is served using its appropriate Azure service. Additionally, complimentary services like the SQL database and blob storage take place in our architecture to ensure that the project artifacts and data are logged and saved securely.

0. **Users** choosing test type, preferred date and unit under test (UUT)
1. **Application gateway** securing and organizing the user interaction with the LIMS UI
2. **Azure App service** (type: web app – Linux docker container) for hosting LIMS

3. **Azure SQL database** (general purpose, serverless) for saving test related logs
4. **Azure blob storage** for saving artifacts (ex: test models)
5. **Scheduler on Azure Functions** for running the scheduling algorithm (Gurobi)
6. **DoE on Azure Functions** for running the DoE algorithm.
7. **IAM Access Control** for defining role-based access to users/services when needed
8. **Azure Kubernetes Services** for deploying a cluster that hosts the MQTT broker.
9. **Azure container app** for hosting the virtual test bench in the form of a co-simulation docker container.
10. **Digital Twin** service to represent the simulated UUT, hosted and deployed by partners own VPC.
11. **Physical test bench** deployed and maintained by partners to run tests on UUT and publish metrics to LIMS through the MQTT broker.

4.1 Description of Azure infrastructure

Starting with LIMS, we decided to use Azure App Service (Microsoft, App Service - Build and Host Web Pages, 2024) to host it as a web application that is published as a Linux docker container for portability purposes. The accessibility is controlled by an application gateway (Microsoft, Application Gateway, 2024) and optionally whitelisting the public IPs of users when needed. The application is connected to a database instance hosted on Azure SQL Database service (Microsoft, Azure SQL Database, 2024). This database instance allows only specific set of IPs for security purposes.

As for our scheduling algorithm, it is deployed on Azure Functions service for mainly two reasons. Easy integration with LIMS and short runtime duration. Azure Functions service allows LIMS to trigger the scheduling logic behind it using HTTP requests. Also, Azure functions service (Microsoft, Azure Functions, 2024) is best used for running code snippets that require no longer than 10 minutes to finish, which is the case in our scheduler.

Regarding the MQTT broker, we decided to deploy it on Azure Kubernetes Service using HiveMQ's official helm charts. Deploying the broker as a pod inside a Kubernetes cluster adds robustness, scalability and high availability to the service (HiveMQ, 2024). Technical aspects of the responsible pod and the number of the broker replicas can be altered easily. Due to the observed usage, the cluster uses one node. Additionally, it's worth noting that helm charts allow faster deployment because they contain the broker and all its required services in one chart.

Lastly, the virtual test benches are deployed in the form of a docker container running a co-simulation software. This docker container is hosted on Azure Container App/s (Microsoft, Azure Container Apps, 2024). A specific port is setup to allow real-time metrics exchange with clients such as DT and our Azure SQL database. Just before the start of each virtual test, the respective model is fetched from our blob storage which in turn receives these models from our partner in at the DT side. As for the physical test benches, the same metrics will be pushed from it to the respective clients through our broker, but the actual deployment of these test benches remains under the responsibility of the partner's industrial plant.

4.2 Choice of communication protocol

A study regarding the communication protocols in IoT systems was conducted based on the research of (Sidna, Amine, Abdallah, & El Alami, 2020). We took different factors into considerations such as package limit (MB), bandwidth consumption, latency, security. The detailed outcomes are listed in Table 2 in **Appendix A.**

Based on the features of the communication protocols in Table 2, the most suitable ones for the project FASTEST, turn out to be MQTT and CoAP. A further detailed comparison of the two protocols based on the needs of the project gives MQTT as the right protocol to use in this project because:

- MQTT is a messaging protocol that works over TCP, while CoAP is a document transfer protocol that works over UDP. This means that the connections via TCP between the hosts are more reliable and the data transfer is guaranteed.
- MQTT has a slower transmit cycle and lower communication delay than CoAP, but is simpler to implement.
- MQTT is not RESTful, while CoAP is RESTful and can interface with HTTP systems. However, passing the firewalls can be problematic for CoAP because of the use of UDP. On the other hand, this is not an issue for MQTT since it uses WebSockets.
- MQTT works on flexible topic subscriptions and allows easy addition of data consumers and producers, while CoAP has a stable resource discovery mechanism.
- MQTT is best suited for distributing live data to multiple clients, while CoAP is best suited for transferring state information between client and server.

When it comes to exchanging larger content (ex: test models) between connected clients and our infrastructure, we need to use a different communication protocol to accommodate this type of data. After research on the available protocols, we decided to go with the SFTP protocol. SFTP stands for secure file transfer protocol. It offers several advantages particularly for secure and reliable data transfer. Below are the main advantageous aspects of this protocol:

- SFTP provides encryption of both the data in transit and the authentication process, ensuring that sensitive IoT data is not exposed to unauthorized access.
- SFTP operates over a single port (z2) so maintaining the firewall rules on this type of communication is relatively easy.
- It can be used to send large files or batches of data which is helpful in IoT use cases such as ours especially in periodic uploads.

- The protocol provides authentication capabilities using public keys when needed.

During the future phase of integrating model data within our working package, integrating SFTP client/s to Azure blob storage will be implemented seamlessly due to the Azure support to this communication protocol (Microsoft, Enable or disable SSH File Transfer Protocol (SFTP) support in Azure Blob Storage, 2024). By simply enabling SFTP access and authorizing access to the client/s, we can start exchanging data across networks.

4.3 Description of component interactions and data format

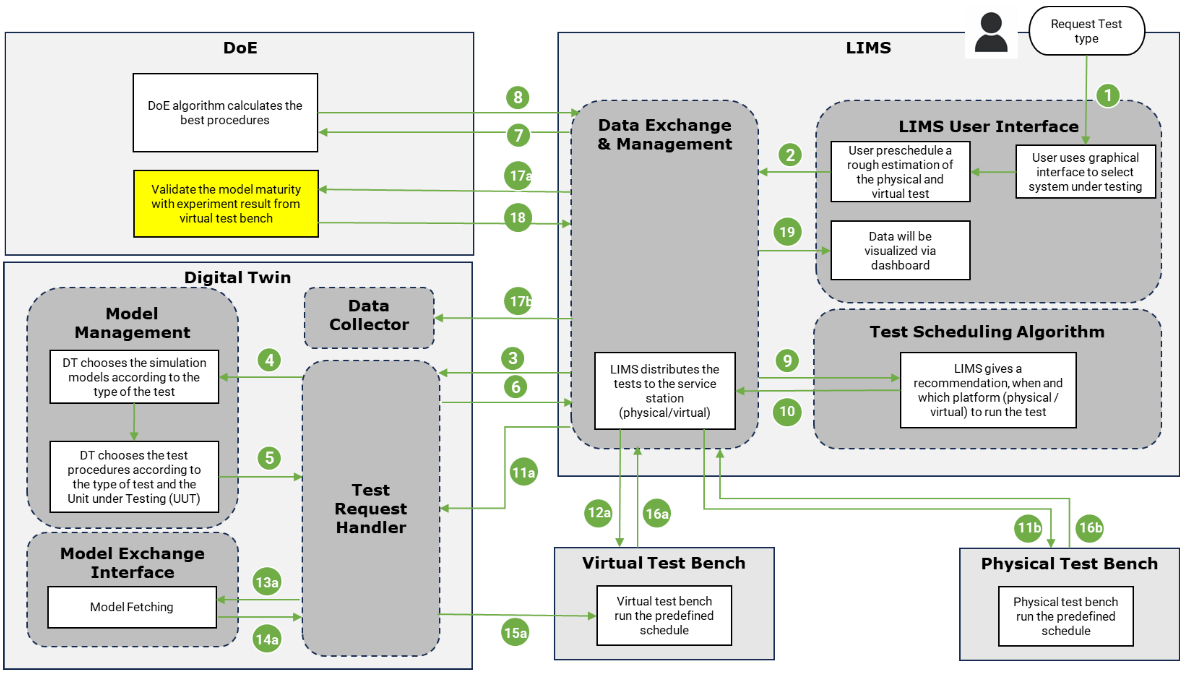

DoE
DoE algorithm calculates the best procedures
Validate the model maturity with experiment result from virtual test bench
LIMS
Request Test type
Data Exchange & Management
LIMS User Interface
User preschedule a rough estimation of the physical and virtual test
User uses graphical interface to select system under testing
Data will be visualized via dashboard
Test Scheduling Algorithm
LIMS gives a recommendation, when and which platform (physical / virtual) to run the test
LIMS distributes the tests to the service station (physical/virtual)
Digital Twin
Model Management
DT chooses the simulation models according to the type of the test
DT chooses the test procedures according to the type of test and the Unit under Testing (UUT)
Data Collector
Test Request Handler
Model Exchange Interface
Model Fetching
Virtual Test Bench
Virtual test bench run the predefined schedule
Physical Test Bench
Physical test bench run the predefined schedule
1
2
3
4
5
6
7
8
9
10
11a
11b
12a
13a
14a
15a
16a
16b
17a
17b
18
19

Figure 2 Communication workflow and interactions among FASTEST components

In Figure 2, a workflow diagram explains the interaction between different services hosted by all the partners that govern the goals of working package 6. The flow starts with the user choosing from the LIMS UI a specific test type, the unit under test and the preferred time of the test to be executed. LIMS will forward this data to the partners at DT that will choose the appropriate simulation models and the required procedures. LIMS will then be responsible for pushing these procedures to DoE that will calculate the best procedures using their own approximation algorithms. LIMS then forwards the results from DoE to the test scheduling algorithm that recommends when and which test bench is to be chosen right before executing the test. Lastly, LIMS will use these results to trigger the tests on both the virtual and physical test benches. As the virtual test benches need a model to run the test, DT will push the previously chosen model to the co-simulation software containing the virtual test bench. After the test runs on both physical and virtual test benches, test metrics are fed back to the DoE in order to determine model maturity by comparing the results from both benches.

Table 1 Workflow communication protocols

Nr.	From	Description of interaction	To	Communication protocol	Data type
1	User	Choose the UUT via		HTTP/S	JSON
2	LIMS – User interface	Send the UUT and preschedule	LIMS-Data management	Database connection	SQL
3	LIMS – Data management	Forward the prescheduled UUT to	DT - Test request handler	MQTT	JSON
4	DT- Test request handler	Forward the prescheduled UUT to	DT – Model management	MQTT	JSON
5	DT – Model management	Send the selected model, procedures and UUT to	DT – Test request handler	MQTT	JSON
6	DT – Test request handler	Forward the selected model, procedures and UUT to	LIMS – Data management	MQTT	JSON
7	LIMS – Data management	Forward the selected model, procedures and UUT to	DoE	Database connection	SQL
8	DoE	Send optimized test procedures to	LIMS – Data management	Database connection	SQL
9	LIMS – Data management	Forward optimized test procedure to	LIMS – Test scheduling	Database connection	SQL
10	LIMS – test scheduling	Send the task scheduling to	LIMS – Data management	Database connection	SQL
11a	LIMS – Data management	distributes the test schedule to corresponding	DT test request handler	MQTT	JSON
11b	LIMS – Data management	distributes the test schedule to corresponding	Physical test bench	MQTT	JSON
12a	LIMS – Data management	distributes the test schedule to	Virtual test bench	MQTT	JSON
13a	DT – Test request handler	Initialize the model fetching in	DT model exchange interface	MQTT	JSON
14a	DT Model exchange interface	Send the fetched model files to	DT Test request handler	SFTP	FMU or others
15a	DT Test request handler	Forward the fetched model files to	Virtual test bench	SFTP	FMU or others
16a	Virtual test bench	Send the real time test result to	LIMS – Data management	MQTT	JSON
16b	Physical test bench	Send the real time result to	LIMS – Data management	MQTT	JSON

17a	LIMS Data management	Forward the result to	DoE	Database connection	SQL
17b	LIMS Data management	Pushes real time test results to	DT data collector	MQTT	JSON
18	DoE	Forward test/model related results after comparison	LIMS – Data management	Database connection	SQL
19	LIMS – Data management	Forward test/model related results	LIMS – UI	Database connection	SQL

4.4 Readiness of Connecting to DoE and Virtual Test Benches

The virtual test bench is going to be hosted on Azure Container App due to the encapsulation of the co-simulation software in a docker container. This allows us to open ports specifically for communication between the co-simulation software and other entities such as the Azure SQL Database through our MQTT broker. Once the real-time metrics are saved under our database, LIMS data management will be able to publish these metrics to partners from DT.

As for the DoE, the WP2 partners will be responsible for preparing the algorithm and specifying the expected input/output formats for handing it over to us for deployment. Due to the expected short runtime cycles and the necessity of being triggered easily by LIMS through HTTP calls, Azure Functions is the chosen medium of deployment for DoE. We will be able to assign the appropriate development environment, variables and libraries inside the function according to the programming language used to develop DoE. Using Azure functions will help us connect the processes between LIMS and DoE with minimal deployment delays, real-time and secure communication within our virtual network.

5. CONNECTING LIMS TO DT, DOE AND PHYSICAL TEST BENCHES

5.1 Connecting DT to LIMS

The Digital Twin platform is hosted on a dedicated Azure cloud environment and is comprised of multiple components. Each component operates as a microservice, running within a Docker container deployed on a Kubernetes cluster.

Communication between the Digital Twin (DT) platform and the Laboratory Information Management System (LIMS) relies on the MQTT protocol. Specifically, this connection is established through an MQTT bridge between the HiveMQ broker (part of the LIMS platform) and the RabbitMQ broker deployed on the Kubernetes cluster within the Digital Twin platform. This bridge facilitates the exchange of MQTT messages on designated topics between components of the LIMS and DT, enabling the transfer of new test requests, model information, and test results.

Bridge configuration is managed on the HiveMQ broker side and requires the HiveMQ Enterprise Bridge Extension. This extension enables HiveMQ to bridge with other MQTT brokers to exchange messages, supporting bi-directional topic mapping and configurable topic filters to direct messages to specific destination topics on other MQTT brokers.

5.2 Connecting Physical Test Benches to LIMS

The connection between LIMS and a physical test bench provided by Flanders Make was established and tested. The dedicated battery testing software from the supplier controls and monitors the physical battery testing hardware. This software implements the different test procedures. All measured data is stored in a (local) SQL database. This way the test bench could be operated standalone. However, we will integrate the setup in a network infrastructure and control the test bench software remotely. Therefore, we will add a test coordinator software which communicates to the test bench software using a TCP interface. This allows changing the configuration of the test bench and control the execution of the tests.

This coordinator software will also feature an MQTT client that will receive the test request from the remote LIMS Data Management and send the real-time data at regular time intervals to the MQTT broker in the LIMS Data Management infrastructure. This coordinator software will be deployed on a virtual PC or docker container.

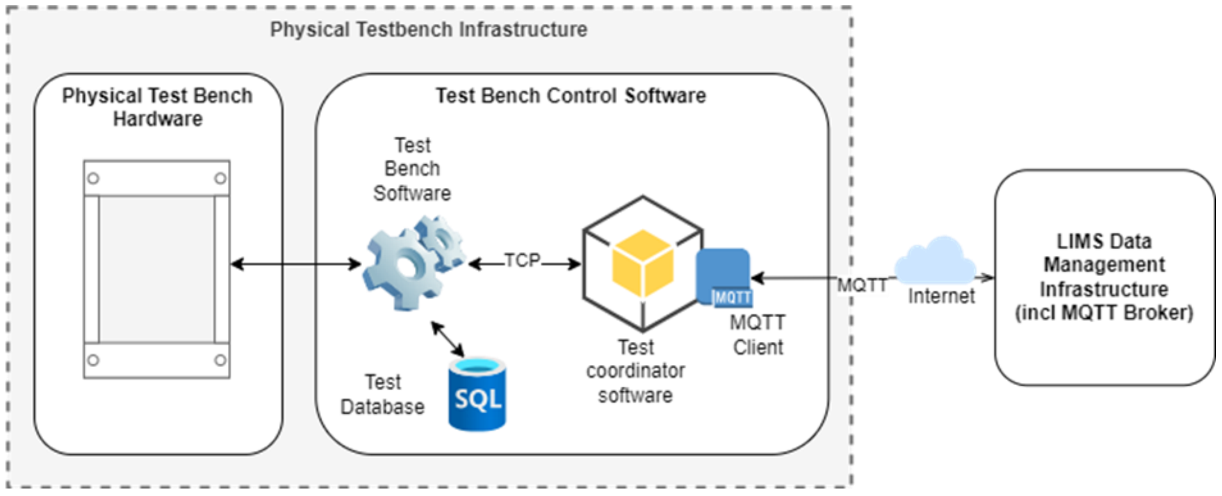

Physical Testbench Infrastructure
Physical Test Bench Hardware
Test Bench Control Software
Test Bench Software
TCP
Test Database
SQL
Test coordinator software
MQTT
MQTT Client
MQTT
Internet
LIMS Data Management Infrastructure (incl MQTT Broker)

Figure 3. Physical Test Bench infrastructure

6. RESULTS

This chapter focuses on representing the results obtained after experimenting the communication protocol (MQTT) between LIMS, DT and the physical test bench. Each test results are organized by a table highlighting the contributed partners, system components, date and status.

6.1 Connecting to DT

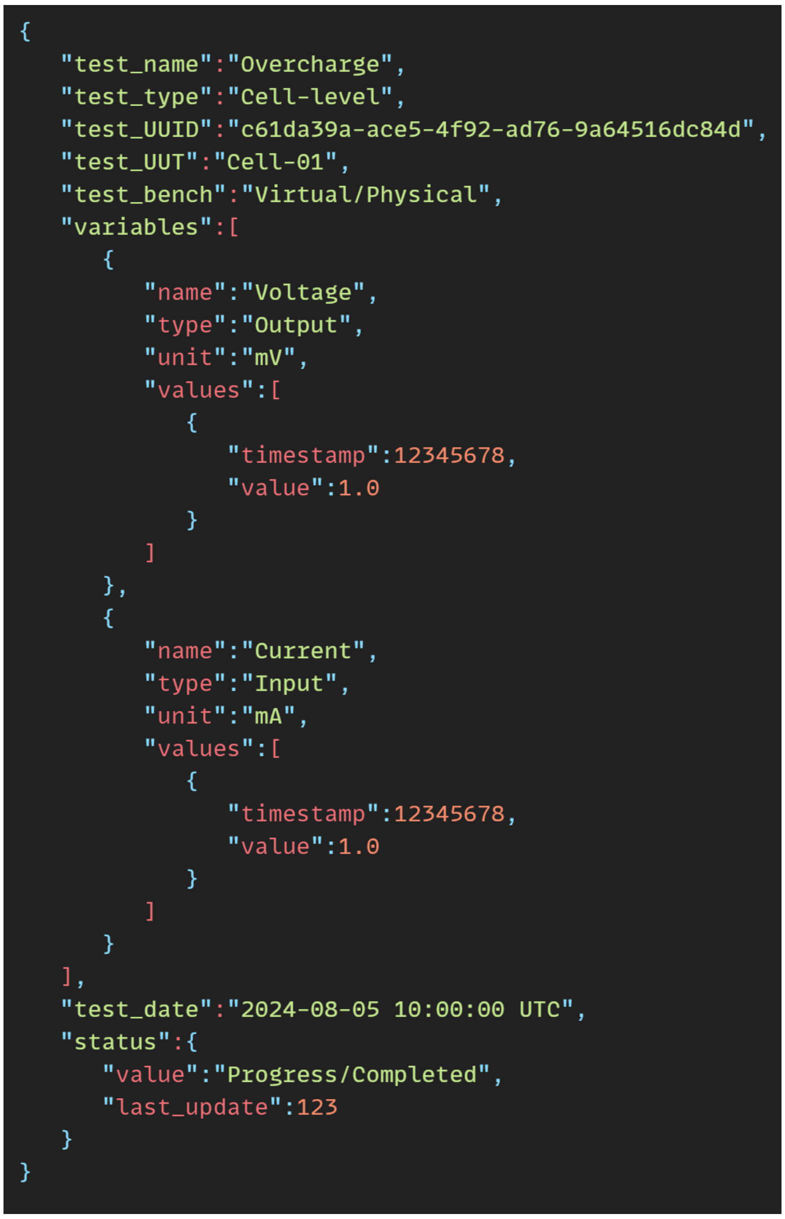
This test was implemented to setup the communication between DT and LIMS as this communication bus is important for successful test cycles. During these cycles, interaction between these two components (DT and LIMS) includes sharing important messages such as the scheduled test type and real-time test metrics.

6.1.1 Test Description

Partners	**System Components**	**Date, Place**	**Status**
FEV, COMAU, INEGI	LIMS, DT, DoE	07.10.2024. Online (Azure)	Passed
Objectives of the Test			
• Test the communication between LIMS, DT and DoE • Assure message type and format.			
Test Description			
Message publishing was tested from LIMS to the Digital twin where the agreed message structure was sent as a JSON. Broker bridging capability was tested where the broker from FEV side acts as a bridge and forwards messages automatically to a preassigned topic towards the broker from the digital twin side. Messages were sent successfully on a real-time basis with no noticeable delays.			
Open Issues			
• None			

6.1.2 Test Results and Evaluation

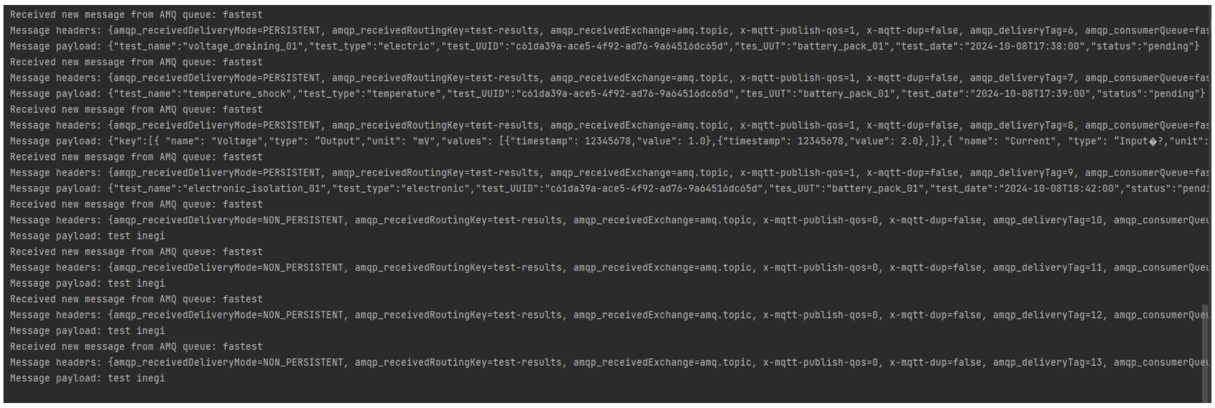
The communication test between DT partners and FEV was established successfully where the MQTT broker (using HiveMQ) at FEV served as a bridge to the MQTT broker (using RabbitMQ) at COMAU (DT partners). The message payload used during the test was constructed of a JSON containing the keys and values shown in Figure 4. The values are experimental, but they reflect actual data that we foresee to be used in the future:



```
{
   "test_name":"Overcharge",
   "test_type":"Cell-level",
   "test_UUID":"c61da39a-ace5-4f92-ad76-9a64516dc84d",
   "test_UUT":"Cell-01",
   "test_bench":"Virtual/Physical",
   "variables":[
      {
         "name":"Voltage",
         "type":"Output",
         "unit":"mV",
         "values":[
            {
               "timestamp":12345678,
               "value":1.0
            }
         ]
      },
      {
         "name":"Current",
         "type":"Input",
         "unit":"mA",
         "values":[
            {
               "timestamp":12345678,
               "value":1.0
            }
         ]
      }
   ],
   "test_date":"2024-08-05 10:00:00 UTC",
   "status":{
      "value":"Progress/Completed",
      "last_update":123
   }
}
```

Figure 4. Proposed real-time test result JSON format

Figure 5 represents a screenshot of the terminal from our partner COMAU taken during our initial communication test. Following the defined workflow, the first step requires the user to select specific parameters: the test type, test name, UUID, UUT and the preferred test date. To simulate this, a user selects this information in the LIMS UI, then the information is automatically combined into a JSON file and sent to DT. During the same test session, we successfully tested transmitting test results in JSON format as well. This step simulates the process by which LIMS receives results from the physical test bench and forwards it to DT for analysis and visualization. The printed timestamps on the terminal confirm that the messages were transmitted to DT without errors, reflecting reliable message exchange. Broker bridging was implemented, allowing any client sending data to FEV broker on a specific MQTT topic to have that data automatically forwarded (bridged) to the DT broker. To enable seamless communication, broker bridging was implemented. In this setup, any client publishing data to the FEV broker on a designated MQTT topic automatically has that data bridged to the DT broker, facilitating synchronized and efficient data flow across systems.



```
Received new message from AMQ queue: fastest
Message headers: {amqp_receivedDeliveryMode=PERSISTENT, amqp_receivedRoutingKey=test-results, amqp_receivedExchange=amq.topic, x-mqtt-publish-qos=1, x-mqtt-dup=false, amqp_deliveryTag=6, amqp_consumerQueue=fas
Message payload: {"test_name":"voltage_draining_01","test_type":"electric","test_UUID":"c61da39a-ace5-4f92-ad76-9a64516dc65d","tes_UUT":"battery_pack_01","test_date":"2024-10-08T17:38:00","status":"pending"}
Received new message from AMQ queue: fastest
Message headers: {amqp_receivedDeliveryMode=PERSISTENT, amqp_receivedRoutingKey=test-results, amqp_receivedExchange=amq.topic, x-mqtt-publish-qos=1, x-mqtt-dup=false, amqp_deliveryTag=7, amqp_consumerQueue=fas
Message payload: {"test_name":"temperature_shock","test_type":"temperature","test_UUID":"c61da39a-ace5-4f92-ad76-9a64516dc65d","tes_UUT":"battery_pack_01","test_date":"2024-10-08T17:39:00","status":"pending"}
Received new message from AMQ queue: fastest
Message headers: {amqp_receivedDeliveryMode=PERSISTENT, amqp_receivedRoutingKey=test-results, amqp_receivedExchange=amq.topic, x-mqtt-publish-qos=1, x-mqtt-dup=false, amqp_deliveryTag=8, amqp_consumerQueue=fas
Message payload: {"key":[{ "name": "Voltage","type": “Output","unit": "mV","values": [{"timestamp": 12345678,"value": 1.0},{"timestamp": 12345678,"value": 2.0},]},{ "name": "Current", "type": “Input�?,"unit":
Received new message from AMQ queue: fastest
Message headers: {amqp_receivedDeliveryMode=PERSISTENT, amqp_receivedRoutingKey=test-results, amqp_receivedExchange=amq.topic, x-mqtt-publish-qos=1, x-mqtt-dup=false, amqp_deliveryTag=9, amqp_consumerQueue=fas
Message payload: {"test_name":"electronic_isolation_01","test_type":"electronic","test_UUID":"c61da39a-ace5-4f92-ad76-9a64516dc65d","tes_UUT":"battery_pack_01","test_date":"2024-10-08T18:42:00","status":"pendi
Received new message from AMQ queue: fastest
Message headers: {amqp_receivedDeliveryMode=NON_PERSISTENT, amqp_receivedRoutingKey=test-results, amqp_receivedExchange=amq.topic, x-mqtt-publish-qos=0, x-mqtt-dup=false, amqp_deliveryTag=10, amqp_consumerQueu
Message payload: test inegi
Received new message from AMQ queue: fastest
Message headers: {amqp_receivedDeliveryMode=NON_PERSISTENT, amqp_receivedRoutingKey=test-results, amqp_receivedExchange=amq.topic, x-mqtt-publish-qos=0, x-mqtt-dup=false, amqp_deliveryTag=11, amqp_consumerQueu
Message payload: test inegi
Received new message from AMQ queue: fastest
Message headers: {amqp_receivedDeliveryMode=NON_PERSISTENT, amqp_receivedRoutingKey=test-results, amqp_receivedExchange=amq.topic, x-mqtt-publish-qos=0, x-mqtt-dup=false, amqp_deliveryTag=12, amqp_consumerQueu
Message payload: test inegi
Received new message from AMQ queue: fastest
Message headers: {amqp_receivedDeliveryMode=NON_PERSISTENT, amqp_receivedRoutingKey=test-results, amqp_receivedExchange=amq.topic, x-mqtt-publish-qos=0, x-mqtt-dup=false, amqp_deliveryTag=13, amqp_consumerQueu
Message payload: test inegi
```

Figure 5. Successfully delivered example MQTT messages to DT

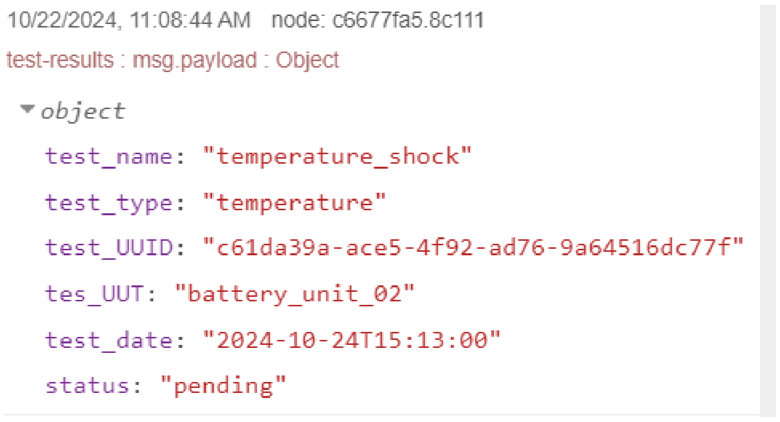
6.2 Connecting to Physical Test Bench

6.2.1 Test Description

Partners	**System Components**	**Date, Place**	**Status**
FEV, Flanders Make	LIMS, Physical test bench	22.10.2024. Online (Azure)	Passed
Objectives of the Test			
• Test the communication between LIMS and the physical test bench • Assure message content and format.			
Test Description			
Publishing test metrics messages between LIMS and the physical test bench was implemented successfully. Both subscribing and publishing worked on real-time basis with no noticeable delays. interval (messages per minute) of the metrics publishing from the physical bench to LIMS is to be decided on a later stage of the project.			
Open Issues			
• None			

6.2.2 Test Results and Evaluation

LIMS successfully implemented a communication test with the partners from the physical test bench department (Flanders Make) by sending/receiving example test related data as shown below. The timestamp of the message indicates successful message receiving. JSON format was the standard used between all partners during the tests as it provided flexibility in defining keys/values and easy serialization/deserialization using the programming languages of all partners (ex: Python, c, c#, etc).



```
10/22/2024, 11:08:44 AM   node: c6677fa5.8c111
test-results : msg.payload : Object
▾object
  test_name: "temperature_shock"
  test_type: "temperature"
  test_UUID: "c61da39a-ace5-4f92-ad76-9a64516dc77f"
  tes_UUT: "battery_unit_02"
  test_date: "2024-10-24T15:13:00"
  status: "pending"
```

Figure 6. Successfully delivered example MQTT messages to the physical test bench

7. CONCLUSION & NEXT STEPS

This deliverable emphasizes the technical aspect of connecting both the physical and virtual test benches and messages exchange between them and DT/DoE through LIMS data management layer. All communication between the respective parties is systematically designed, covered and explained in the form of a workflow that was agreed upon through iterative enhancements to serve as a common ground truth for all partners. Moreover, this work showcases the deployment approaches followed by FEV and the partners (FM, COMAU, INEGI, ABEE) during the development of LIMS & virtual test bench, DoE, physical test bench and DT respectively. The architecture design conducted by FEV to deploy LIMS, the scheduling algorithm, DoE and the MQTT broker was discussed. Then, message exchange capabilities were successfully tested between LIMS, DT and the physical test bench and this resulted in real-time performance with no noticeable delays. At the end, readiness of connecting DoE to the virtual test bench was proven to be true, paving the way to integrate them and prepare for validation in upcoming work packages.

8. BIBLIOGRAPHY

HiveMQ. (2024, 10 30). *www.hivemq.com*. Retrieved from www.hivemq.com: https://www.hivemq.com/solutions/the-best-mqtt-broker-for-azure/

Microsoft. (2024, 10 30). *App Service - Build and Host Web Pages*. Retrieved from Microsoft Azure: https://azure.microsoft.com/en-us/products/app-service

Microsoft. (2024, 10 30). *Application Gateway*. Retrieved from Microsoft Azure: https://azure.microsoft.com/en-us/products/application-gateway/?msockid=333b0623c62160732cc912b7c75961f7

Microsoft. (2024, 10 30). *Azure Container Apps*. Retrieved from Microsoft Azure: https://azure.microsoft.com/de-de/products/container-apps/?msockid=333b0623c62160732cc912b7c75961f7

Microsoft. (2024, 10 30). *Azure Functions*. Retrieved from Microsoft Azure: https://azure.microsoft.com/en-us/products/functions/?msockid=333b0623c62160732cc912b7c75961f7

Microsoft. (2024, 10 30). *Azure SQL Database*. Retrieved from Microsoft Azure cloud: https://azure.microsoft.com/en-us/products/azure-sql/database/?msockid=333b0623c62160732cc912b7c75961f7#Features

Microsoft. (2024, 10 30). *Enable or disable SSH File Transfer Protocol (SFTP) support in Azure Blob Storage*. Retrieved from Microsoft Azure: https://learn.microsoft.com/en-us/azure/storage/blobs/secure-file-transfer-protocol-support-how-to?tabs=azure-portal

Sidna, J., Amine, B., Abdallah, N., & El Alami, H. (2020). Analysis and evaluation of communication Protocols for IoT Applications. *Proceedings of the 13th international conference on intelligent systems: theories and applications*, (pp. 1-6).

APPENDIX A. Comparison of Communication Protocols

Table 2 Comparison of Communication Protocols

Characteristic	MQTT	HTTP	DDS	XMPP	AMQP	coAP
Architecture	Client/broker	Client/server	Brokerless	Client/server	Client/broker or client/server	Request/response or publish/subscribe
QoS (Quality of Service)	QoS(0/1/2)	Limited	23 policies	No support	Settle/unsettle format	Confirmable/non-confirmable
Security	TLS/SSL Has the lowest level	TLS/SSL	TLS/SSL, DTLS	TLS/SSL	TLS/SSL, IPSec, SASL Strongest security	DTLS, IPSec guarantee authentication, integrity and encryption
Latency	MQTT has lower latency than HTTP	involves largest latency, HTTP has highest latency than all others	Low latency	Low latency	AMQP has lower latency than MQTT	CoAP has lower latency than all others
Bandwidth consumption	Consumes high bandwidth	Largest bandwidth consumption	Low	Low	High bandwidth consumption	Lowest bandwidth consumption
Applications	Home automation, Enterprise level applications	Web	Medical Imaging, Military Systems	Instant Messaging, Group chat, Gaming, Vehicle Tracking	Business Messaging, Banking Industry	Smart homes, Smart grid, Building Automation
Package limit (MB)	256 MB/Message	No limit by specification. Limited by server and client. IE: 2GB Firefox: 2GB Chrome: 4 GB Opera: 4 GB	UDPv4 - 64KB (larger messages are fragmented)	No limit by specification. Limited by the server. Suggested minimum 10KB4		Limit of underlying transport - RFC 7252 suggests 1152 bytes for UDP if nothing is known about the target
Data throughput	(256B Messages) MQTT (QoS 1) - 1234 msg/s 3 MQTT (QoS 0) - 18416 msg/s 3					(256B Messages) 8729 msg/s 3
Standards	OASIS, Eclipse Foundations	IETF and W3C	OMG	IETF	OASIS, ISO/IEC	IETF, Eclipse Foundation
Encoding format	Binary	Text	Binary	Text	Binary	Binary
Connectivity	One-to-one, one-to-many, and many-to-many	One-to-One	Peer-to-peer communication, one-to-one, one-to-many, many-to-many, and many-to-one	One-to-One	Point-to-point	One-to-one (natively), many-to-many communications (not natively but with extensions)
Energy consumption	MQTT was more energy efficient	Requires highest power/energy consumed by HTTP was much larger than with MQTT	-	Increased power consumption	requires slightly higher power	CoAP is more efficient in terms of energy
Transport protocol	TCP (MQTT-SN can use UDP) Transport via Websockets possible (broker dependent feature)	TCP	UDP	TCP	TCP, SCTP	UDP