Acta Montanistica Slovaca
ISSN 1335-1788
actamont.tuke.sk

Digital Twins In The Oil And Gas Industry: The Impact Of Different Data Transfer Protocols On Performance

Sergey ZHIRONKIN[1,2], Fares ABU-ABED[3,4*] and Anastasia BELOVA[5]*

Authors' affiliations and addresses:
[1] Siberian Federal University, 79 Svobodny Av., 660041 Krasnoyarsk, Russia.
E-mail: szhironkin@sfu-kras.ru

[2] T.F. Gorbachev Kuzbass State Technical University, 28 Vesennya St., 650000 Kemerovo, Russia
E-mail: zhironkinsa@kuzstu.ru

[3] Tver State Technical University, 22 Afanasiya Nikitina emb., 170026 Tver, Russia.
E-mail: aafares@mail.ru

[4] Gulf University for Science and Technology, Mishref Campus, Kuwait.
E-mail: aafares@outlook.com

[5] T.F. Gorbachev Kuzbass State Technical University, 28 Vesennya St., 650000 Kemerovo, Russia
E-mail: bel2547@yandex.ru

***Correspondence:**
Sergey Zhironkin, Tver State Technical University, 22 Afanasiya Nikitina emb., 170026 Tver, Russia.
E-mail: aafares@mail.ru

How to cite this article:
Zhironkin, S., Abu-Abed, F. and Belova, A. (2025). Digital Twins In The Oil And Gas Industry: The Impact Of Different Data Transfer Protocols On Performance. *Acta Montanistica Slovaca,* Volume 30 (4), 903-919

DOI:
https://doi.org/10.46544/AMS.v30i4.06

Abstract
As the global oil and gas industry moves towards achieving sustainable development climate goals, the role of hydrocarbon production optimization is increasing, facilitated by Industry 4.0 technologies such as digital twins. The quality of their operations is determined by several factors, including the speed and reliability of data transfer. This makes it relevant to study the influence of the data transfer protocol choice on the efficiency of digital twins in the oil and gas industry. The purpose of the study is to compare three common approaches to data exchange (REST API, MQTT, and WebSocket) in the context of integrating a digital twin with real-time control systems. To achieve this goal, an experiment was conducted to measure the data transfer parameters of a digital twin of an oil and gas well using the REST API, MQTT, and WebSocket protocols, on an experimental stand that emulates the "smart well – cloud digital twin – mobile application" scenario. Technical implementation was carried out using the Java / Kotlin languages for the Android client (digital twin). The stand was used to analyze their efficiency, transmission reliability, and ability to operate in real time. As a result of the comparative experiment, it was found that protocols with a constant connection (MQTT, WebSocket) have significantly lower data delivery delays than the polling REST API, which directly affects the accuracy and relevance of the digital model. Based on this, the article provides recommendations for choosing a protocol depending on the required update efficiency, resistance to network failures, and device limitations.

Keywords
digital twin, oil and gas industry, REST API, MQTT, WebSocket, efficiency, reliability, real-time.

Introduction

Digital twins are becoming an integral tool for improving efficiency and safety in the oil and gas industry (Lukyanenok et al., 2023; Cherevko et al., 2024). A digital twin is a virtual model of an industrial facility or process that operates in parallel with a physical prototype in real time (Mayani et al., 2018), which can also be thought of as an immersive data analysis technology that enables interaction in the human-infrastructure-equipment system in such a way as to ensure contextual development of management decisions (Wanasinghe et al., 2020).

As an integral part of Industry 4.0 (Gasanov et al., 2024; Aleshina, 2023), digital twins integrate advanced technologies such as neural networks, the Internet of Things, machine learning, and predictive analytics (da Silva et al., 2023). In the oil and gas industry, characterized by complex technological processes and high risk, the timely receipt of reliable field data is critical. Here, the role of real physical systems' digital twins is to model scenarios of equipment failures and wear of individual parts, destruction of well walls, negative impact on the environment, etc. (Levina et al., 2023). In this way, it is possible to increase equipment productivity and optimize operating parameters in unconventional gas fields (Alsulaiman et al., 2024). Digital twins are also used to ensure the proper operation of the casing and cementing, and to avoid formation damage through continuous monitoring and comparison of predicted indicators with actual indicators (Bykova et al., 2020). A significant use of digital twins in the oil and gas sector is training employees in safety procedures and emergency response protocols through immersive simulations (Nagornov, 2024), which generally helps extend asset lifespans (Anaba et al., 2024).

In practice, the use of digital twins of offshore platforms yields the following results: an increase in production volume by 10-15%, an optimization of investments and operating costs by 12-15%, and an increase in resource efficiency by 15-17% (Eremin, 2018). In onshore oil and gas drilling, the efficiency of the production system using a digital twin increases by 3% (Shen et al., 2021).

There is an experience in using specialized software by oil and gas companies for applying digital twins for seismic visualization in the exploration of new fields (GeoSigns software from Shell), pipeline design and maintenance (APEX from BP), interactive models of drilling equipment and underground environments (Emerson and Microsoft software packages for Chevron), digital twins of entire drilling platforms (Johan Sverdrup from Equinor) (Dada, 2024).

One of the key aspects of digital twins is that they can be updated in real time, allowing for continuous optimization of the performance of a set of offset wells (Ferrigno et al., 2023) while minimizing predicted shocks and vibrations, reducing the specific mechanical energy while maximizing drilling speed (Gandikota et al., 2024). More mature digital twins that integrate virtual and augmented reality technologies determine the performance of production equipment without human intervention (Mustard et al., 2023) based on controlled variables, allowing for the monitoring of key performance indicators across process flow diagrams (Yusupbekov et al., 2022). Reducing carbon emissions in the oil and gas industry can be achieved by integrating Agile frameworks into the planning phase of digital twin deployments, using methodologies such as Scrum and Kanban, and by leveraging energy system modeling tools such as EnergyPlus and OpenStudio (Bello et al., 2025).

One of the key parameters of the digital twin's functioning is the response to real-time performance, ensuring instant data transfer and visualization. As an example, the digital twin framework of a rack-and-pinion drilling rig, using platforms such as Unity3D (Jiangang et al., 2024) and Open Subsurface Data Universe, is used to eliminate isolated data storage in the digital oil and gas production management ecosystem (Correia et al., 2022). A digital twin of a drilling well must be able to build connections between drilling, well completion, and upstream, including any well workover activities (Titto et al., 2022), working out "what if?" scenarios without the risk of equipment damage (Azieva et al., 2024) and interruptions in the supply chain of oil, gas, and oil and gas products to global markets (Dmitrievsky et al., 2022).

At the same time, despite the lack of an alternative to implementing digital twins in oil and gas production processes, driven by the challenges of increasing productivity in unconventional fields and reducing carbon emissions, a number of problems hinder this process. In particular, a complete digital reconstruction of physical objects can entail incompatibility of innovative technologies and obsolete equipment, which will not allow recouping investments (Pandi, 2023). Therefore, the speed of cloud and edge computing, as well as the speed of data exchange, are critical factors for the implementation of digital twins (Knebel et al., 2023), the structure of which is quite complex and can combine Bayesian inference, Monte Carlo modeling, transfer learning, online learning, a cognitive approach and hyperdimensional model reduction (Rebello et al., 2023).

Definitely, the range of problems accompanying the implementation of digital twins in the oil and gas industry is not limited to the speed of data exchange between physical and cyber systems. It includes the accuracy of data on equipment failure modes, the need to recreate a digital twin when replacing equipment elements, the need to involve a large number of specialists (Singh et al., 2018), and a high level of cybersecurity (Sudhakar et al., 2023). Also, artificial intelligence models used in digital twins of various systems in the oil and gas industry should not introduce even the slightest distortion into the processed data, as this directly affects modeling accuracy (Almatova et al., 2025). This will acquire a special role in the future with the expected replacement of digital twins with digital triplets that combine human cognitive functions, perceptual artificial intelligence with physical objects into full-

fledged virtual domains (Alimam, 2025), which will almost completely eliminate man-made accidents under the influence of the human factor (Abdrakhmanova et al., 2020).

In general, most digital twins in the oil and gas industry rely on real-time data collection, so they are critically dependent on the speed of data processing from various sources, including smart sensors (Meza et al., 2024) coupled with neural networks with long-term and short-term memory (Ren et al., 2021).

In real oil and gas production, data sources can be geographically distributed (wells, pipelines, oil and gas processing plants, etc.), and communication channels are subject to delays and failures. Therefore, the data transfer protocol between the physical object and the digital twin directly affects the quality of the model. It is necessary to ensure minimal delay in information delivery, reliability (so that data arrives without loss), and the ability to work in real time.

Various technologies are used today to exchange data in such systems. The traditional approach is based on the REST API – a web service architecture that uses HTTP requests to transfer data (Matcha et al., 2025). The REST API is widespread and easy to implement; such interfaces are often used to integrate industrial systems and cloud platforms. For example, the REST API plays a key role in ensuring the interactivity and functionality of web platforms. However, the classic REST protocol has a limitation: the server cannot send data to the client on its own – either periodic polling or additional add-ons (long polls, webhooks) are required, which can lead to delays.

An alternative is persistent connection protocols. WebSocket is a technology that allows establishing a long-term two-way TCP connection between a client and a server (Dyuthi, 2024). Unlike HTTP, where each transaction requires a separate request and response, WebSocket supports real-time data exchange: after a handshake, the client and server can freely send messages to each other without additional requests. This makes WebSocket attractive for tasks that require instant transmission of updates (e.g., monitoring systems, stock trading, and online chats). It has already been shown that WebSocket provides significantly lower latency compared to traditional HTTP mechanisms and generally higher scalability with a large number of messages.

For Industrial Internet of Things (IIoT) tasks, the MQTT (Message Queuing Telemetry Transport) protocol (Amirkhanov et al., 2025) has become especially widespread. MQTT was originally developed for devices with limited resources and unreliable networks. It implements a publish-subscribe model: sensor devices publish messages on certain topics, and recipients (for example, a digital twin server) subscribe to them and receive new data automatically. The advantage of MQTT is a very small message header (at least 2 bytes) and a permanent connection through a broker, which minimizes network overhead. MQTT is specifically designed for telemetry and is well-suited for IoT devices. It works efficiently with high latency and limited channel bandwidth, supports delivery confirmation (QoS 0, 1, 2) for reliability, and saves device energy. Thus, when transmitting the same data, MQTT can consume ~20% less energy than REST due to the absence of frequent reconnections.

At the same time, each solution has its limitations. REST API, while robust and simple, suffers from model-execution overhead: it requires opening and closing an HTTP connection for each request, which increases latency and load. MQTT requires a broker and an additional infrastructure layer, and interactions outside the IoT ecosystem (e.g., running directly in a browser) may require running MQTT on top of WebSocket. WebSocket, while easier to integrate into the web environment, does not provide out-of-the-box features such as a message broker/queue or guaranteed delivery of packets during repeated disconnections – these tasks must be addressed at the application layer (or WebSocket must be combined with MQTT). When designing a “physical object – communication channel – digital twin” system, it is important to understand how the choice of protocol will affect the twin’s ability to operate in real time and reliably.

Therefore, the relevance of this study lies in the need to provide digital models of oil and gas equipment with timely, accurate data flow. If a digital twin receives telemetry with significant delay or loss, its predictive value decreases – it no longer reflects the object's current state, and decision-making efficiency is compromised. Modern Industry 4.0 approaches involve the widespread use of IoT sensors and network technologies to connect the physical and virtual worlds. However, there is no single data transfer standard for all cases. In practice, the choice is often between REST APIs (including OPC UA over HTTP in industry), MQTT (via brokers such as Mosquitto, HiveMQ, etc.), and WebSocket (for direct push interaction with applications). The conference proceedings of 2023-2024 note that information system developers are actively exploring the integration of APIs and network protocols to improve the efficiency of process management. Digital twins of oil production facilities can be integrated directly into the control loop and operate in real time. This requires a reliable channel for transmitting operational data from the facility to the model.

The purpose of this article is to conduct a comparative analysis of REST, MQTT, and WebSocket in the context of such requirements and to provide recommendations on which protocol is best used for communication with the digital twin of an oil and gas facility in various situations. The article presents a study of the exchange efficiency, data transfer reliability within these approaches, and the ability of the digital twin to operate in real time.

These characteristics of the digital twin are explored through the following additional research questions:

RQ1. How do the characteristics of the REST API, MQTT, and WebSocket protocols affect the speed of data exchange between elements of the digital twin of an oil and gas well, allowing the implementation of the “smart well – cloud digital twin – mobile application” scenario?

RQ2. How does choosing a protocol with the highest data transfer rate affect the performance of a digital twin?

To provide a reasoned answers to these questions, the article presents an overview of the qualitative and quantitative characteristics of data transfer protocols from the point of view of their application for communication with a digital twin, describes a testing stand that emulates the "smart well - cloud digital twin - mobile application" scenario for analyzing the data transfer rate, and compiles a code for connecting an MQTT client to an Android application. Experimental data are analyzed in the context of reliability and fault tolerance, scalability in wide IoT deployments, ease of integration, user visualization and interaction, compatibility, and adherence to standards.

Materials and methods

Comparative analysis of data transfer protocols: REST API, MQTT, and WebSocket

A digital twin is typically created from data from a variety of sources, including IoT sensors, controllers, manufacturing execution systems, and historical enterprise databases. As Industry 4.0 evolves, the combination of digital twins and Industrial Internet of Things (IIoT) technologies is seen as the key to improving manufacturing efficiency.

The literature describes successful applications of digital twins to optimize oil and gas production. A virtual model of a field that receives well data in real time allows operators to more quickly assess production efficiency and make decisions on regulating equipment operation (Khalkho, 2025). Another example is a hydrodynamic digital twin of a reservoir that, based on well telemetry, predicts changes in flow rate and pressure in the reservoir pressure maintenance (RPM) system. Such systems require a continuous data flow from dozens of sensors (pressure, temperature, flow rate, etc.) with minimal delays (Shypul et al., 2023).

The key aspect emphasized by researchers is the digital twin's efficiency. Some authors present a scheme in which the digital twin is incorporated into an adaptive control loop: control actions are applied simultaneously to the physical object and its model, and the resulting responses are compared (Pishukhin et al., 2024). At the slightest discrepancy, the system can adjust the control. It is clear that this approach requires minimal data exchange delays – seconds and fractions of a second can count, especially when it comes to emergency control. Thus, the data transmission network must operate practically in real time.

Another important requirement is the reliability and stability of the connection. In the field, the connection can be unreliable (wireless channels, satellite channels with high ping, interruptions). The digital twin must receive data even during temporary connection loss, and the protocol must ensure that messages are delivered or stored repeatedly. Here, the advantage of MQTT stands out: thanks to the broker, it can cache messages for recipients who are temporarily unavailable and ensure delivery immediately upon connection restoration (QoS 1 and 2 guarantee at least-once or exactly-once delivery). At the same time, in a basic HTTP/REST implementation, there is no built-in resending mechanism, so the application must implement its own logic for handling repeated requests in case of errors.

The issue of choosing a communication protocol for real-time systems is widely discussed. For web applications, a comparison of WebSocket and REST is common. It is noted that WebSocket reduces the overhead of frequent data exchange by maintaining a constant connection (there is no need to send HTTP headers each time, re-establish a connection, etc.), thereby reducing bandwidth use and system load. On the other hand, the simplicity of REST (essentially HTTP) facilitates the integration of heterogeneous components and debugging. In the IoT environment, much attention is paid to comparing MQTT and HTTP. Thus, the Bevywise blog (2023) describes a device control scenario: when using a purely REST scheme, the server cannot instantly notify the device of a new event – usually, the device polls the server, say, once a minute, which leads to unacceptable delays for interactive actions. MQTT with a persistent connection eliminates this problem - the broker will immediately send a message to the subscriber as soon as it receives it from the publisher, without waiting for an additional request. Moreover, two-way communication (server → device) is supported even if the device is behind a NAT or firewall, since the initiator of the connection is the device itself, and the connection is maintained constantly.

The software used in this research is based on the Arduino IDE for programming ESP32/ESP8266 microcontrollers interacting with DHT11/DHT22 temperature and humidity sensors. The code is based on the Adafruit DHT Sensor Library and the Adafruit Unified Sensor libraries, which simplify reading sensor data via digital pins. The sketch includes initializing the sensor, polling values at a specified interval, and outputting readings to the serial monitor. For the code to operate correctly, the features of ESP boards are taken into account, such as GPIO numbers, interrupt support, and power management, and error handling is added for incorrect measurements.

Table 1 summarizes the main properties of the protocols under consideration for their application to communication with the digital twin.

Table 1. Comparative characteristics of REST API, MQTT, and WebSocket protocols for digital twin systems.

Characteristic	REST API (HTTP)	MQTT	WebSocket
Interaction model	Request-response (client-server).	Publish/subscribe (via broker).	Two-way permanent channel (socket).
Connection	Short-lived, reset for each request.	Permanent TCP connection to the broker (keep-alive).	A persistent TCP connection between two nodes.
Data delivery method	The client requests the data or sends it at the client's initiative. The server cannot initiate sending without a request.	Devices publish data to a broker, which then sends it to subscribers.	Either party can send data at any time after the connection is established.
Delayed updates	Higher due to the need to wait for the request; polling may be delayed.	Low: data is pushed through the broker immediately.	Low: data is pushed immediately over an open channel.
Bandwidth	Effective for infrequent exchanges; frequent requests result in significant HTTP header overhead.	Optimized for frequent small messages; very small headers.	Close to MQTT, but there is overhead of opening WebSocket (masking frames, etc.).
Reliability of delivery	Guaranteed by transport (TCP) for each request. A lost request must be repeated manually.	Built-in QoS 0,1,2 – possibility of guaranteed delivery and queuing for offline clients.	Guaranteed by transport (TCP). When the connection is broken, messages are lost, and manual confirmations/repeats are required.
Difficulty of implementation	Low: widely used, many ready-made RESTful API frameworks.	Medium: MQTT broker required; clients require MQTT libraries.	Medium: supported by web clients (via WebSocket API or libraries), requires server-side socket implementation.
Typical scenarios	On-demand status query: configuration and historical data.	Telemetry stream from sensors; IoT devices requiring traffic and energy savings.	Online monitoring, real-time event exchange between server and application (web client, mobile application).

As shown in Table 1, the REST API is inferior in terms of interaction efficiency. Its periodic polling model does not guarantee immediate receipt of critical updates. MQTT was originally designed for low-latency telemetry and is well-suited for periodic sensor data. WebSocket provides similarly fast bidirectional communication and is easier to integrate into client applications (especially web interfaces for monitoring). However, MQTT wins in reliability (QoS modes) and better performance on unreliable networks, while WebSocket focuses on stable connections (or requires an add-on for session recovery).

To quantify differences in the protocols, a testing stand was developed to emulate the “smart well – cloud digital twin – mobile application” scenario. In this scheme, a physical object (an oil well) is equipped with temperature, pressure, and flow sensors that transmit telemetry to the digital twin server once per second via one of three protocols. The server performs model calculations (e.g., forecasting well flow rate or equipment wear) and sends the results to the client application (dashboard) for display to operators. Thus, communication is bidirectional: an incoming data stream from the field to the digital twin and an outgoing stream from the digital twin to the user interface. It is necessary to ensure acceptable latency (no more than a few seconds) and reliability (no loss of important updates) for both streams.

In general, the operation of the digital twin "Smart Well – Cloud Digital Twin – Mobile Application" is shown in Fig. 1.

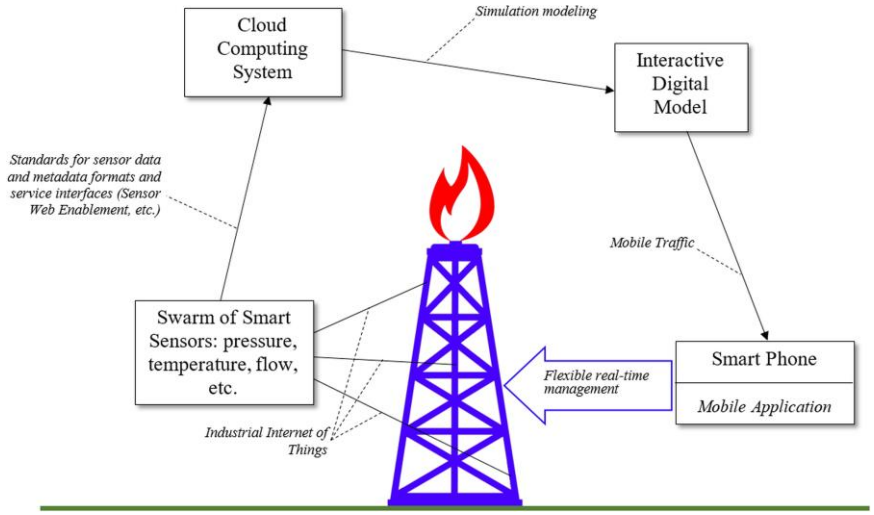

Figure 1. Scenario of operation of the digital twin "Smart Well – Cloud Digital Twin – Mobile Application".

The testing stand consists of the following components: a sensor device, a digital twin server, and a client application. The field node prototype was an ESP32 microcontroller with a connected DHT22 sensor (temperature/humidity measurement). For testing purposes, the microcontroller generated pseudo-random telemetry data (simulating well parameters) and transmitted them via Wi-Fi to the server. The digital twin server was an application running on a local PC or a cloud instance that received data from the device via a specified protocol, recorded the time of receipt, and performed minimal processing (e.g., averaging values) to simulate the twin's computational procedures. The client application was an Android mobile app that displayed updates from the server in real time; it also connected to the digital twin server via the appropriate protocol (REST, MQTT, or WebSocket) to receive processing results. All components are connected by a single data transfer logic: sensors → server (digital model) → operator application.

A separate communication scheme was implemented for each protocol. When using the REST API, the server exposed a RESTful web service with two methods: POST /data to receive data from the device and GET /state to return the estimated state to the client. The sensor device sent an HTTP POST request containing a JSON payload once per second, and the mobile application requested a new state via periodic polling at an interval of 1 s using GET/state. In the case of MQTT, an MQTT broker (Eclipse Mosquitto) was deployed. The device acted as an MQTT client publishing messages with data to a topic (for example, sensors/well_5). The digital twin server subscribed to this topic, instantly received incoming messages, and after processing, published the result to the response topic digital_twin/well_5. The Android client application was an MQTT client subscribed to digital_twin/well_5, due to which new data was displayed immediately upon receipt without polling. For WebSocket, a server-side WebSocket server was launched (e.g., using a Java library) supporting two endpoints: one for receiving data from the device (e.g., ws://server:8080/live) and the second for sending results to the client (ws://server:8080/results). The device establishes a permanent WebSocket connection to the server and sends a JSON-formatted text message containing sensor data once a second. The client application also opens a WebSocket connection to the server (to the address for receiving results) and instantly receives notifications about new calculated data. All WebSocket transmissions are bidirectional and occur without additional requests, based on the push messaging principle. In all three variants, the server part, after receiving new data, ran an update of the digital model (which took <100 ms) and then immediately sent the calculated indicators to the client (or at the next client request in the case of REST).

The tests were conducted sequentially for each protocol under identical conditions, allowing us to compare their impact on the quality of the digital twin's operation. Each run lasted a fixed time interval (e.g., 10 minutes) to accumulate statistics. The testing algorithm in the basic mode was as follows: (1) initialization of the environment (launch of the MQTT broker or WebSocket server, deployment of the REST API); (2) connection of the sensor device and client to the server; (3) transmission of data from the sensor to the server at a frequency of 1 Hz and simultaneous transmission of results from the server to the client; (4) logging of sending and receiving timestamps at each stage, counting successfully delivered messages and traffic volume. The test bench was then used to test the protocols' operation under various conditions. Typical use cases were simulated:

1. Stable connection: the device, server, and client are in the same local network with minimal latency and no losses. In this scenario, the protocols' basic performance was assessed: average delivery delay, delay spread, and throughput under ideal conditions.
2. Limited channel: a degraded network with an additional delay of ~100 ms and packet loss of ~5% was emulated. This was achieved by setting up a network emulator that limits the bandwidth and introduces random delays/losses. The goal was to test the functionality of protocols under conditions of deteriorating connection: increased delays, reduced connection stability, and compromised data integrity.
3. Short-term failures: the connection was intentionally interrupted (e.g., by disabling Wi-Fi on the device for 5 s) and then restored. We analyzed how each protocol handles connection failures: does the transmission continue after restoration, are messages lost during the failure, or does a reconnection require it?

Each of the specified scenarios was played for all three protocols. The data accumulated during the experiment (time stamps, message counters, connection logs) were used to calculate key quality indicators: delivery delays (incoming and outgoing), delay variations (jitter), reliability (percentage of lost messages), and traffic volume. Thus, the developed testbed enabled comparison of MQTT, WebSocket, and REST APIs in a typical "sensor – digital twin – consumer" cycle under controlled conditions, and the resulting metrics directly reflected how the choice of protocol affects the efficiency and accuracy of the digital model.

Each configuration was tested for a fixed time interval (e.g., 10 minutes) to accumulate statistics. The following parameters were measured:

• Average delay in data delivery from the sensor to the server (incoming telemetry) — calculated as the difference between the generation timestamp on the device and the reception time on the server. The server-to-client delay was measured similarly (for the outgoing stream).

• Delay variation (jitter) — statistical deviation in delivery (the standard deviation of the delivery time was calculated for a sample of messages).

• Percentage of lost messages — determined by comparing the counters of sent and received messages at different stages.

• Traffic load — the amount of data (in bytes) transferred over the network during the test period, including protocol headers, was estimated for efficiency analysis.

• Resource usage on the client — Android battery life was subjectively estimated for different protocols (through an indirect indicator of power consumption, since a constant connection can drain the battery more).

To ensure the purity of the experiment, the tests were conducted under the same conditions: the device and the server in the same local network (emulating the low latency of the "ideal" channel), then through artificial bandwidth limitation and increased latency (emulating real cellular/satellite communications). Controlled communication failures were also introduced (Wi-Fi disconnection for 5 seconds, broker reboot, etc.) to test each protocol's behavior.

The technical implementation was carried out using the Java/Kotlin languages. For the Android client, popular libraries were used: OkHttp for working with REST and WebSocket, and the Eclipse Paho MQTT library (via the org.eclipse.paho.android.service wrapper) for MQTT. Below is a simplified fragment of code in Kotlin demonstrating connecting an MQTT client and subscribing to a topic in an Android application:

```
val mqttClient = MqttAndroidClient(this, "tcp://broker.hivemq.com:1883", "ClientID123")
val options = MqttConnectOptions().apply {
    isAutomaticReconnect = true
    isCleanSession = true
}
mqttClient.connect(options, null, object : IMqttActionListener {
    override fun onSuccess(asyncActionToken: IMqttToken?) {
        mqttClient.subscribe("digital_twin/well_5", 0)
        Log.d("MQTT", "Subscribed to topic!")
    }
    override fun onFailure(asyncActionToken: IMqttToken?, exception: Throwable?) {
        Log.e("MQTT", "Connection failed: $exception")
    }
})
```

Similarly, the OkHttp library was used for the WebSocket connection. An example of the WebSocket connection code in Kotlin:

```
val request = Request.Builder().url("ws://server:8080/live").build()
val listener = object : WebSocketListener() {
    override fun onOpen(webSocket: WebSocket, response: Response) {
        webSocket.send("{\"well_id\":5, \"pressure\":101.3, ...}")  // отправка JSON
    }
    override fun onMessage(webSocket: WebSocket, text: String) {
        runOnUiThread { processIncomingData(text) }  // обработка полученных данных
    }
}
val ws = OkHttpClient().newWebSocket(request, listener)
```

Each implementation logged sending and receiving timestamps, as well as connection break/restore events. Additional code snippets demonstrating the processing of received data – parsing JSON, displaying values on the screen, and logging the results are given below.

Example 1: processing an incoming MQTT message. After successfully subscribing to a topic, the MQTT client receives new messages via callback. The code below illustrates parsing JSON data and updating the UI in an Android application when telemetry is received:

```
mqttClient.setCallback(object : MqttCallbackExtended {
    override fun connectComplete(reconnect: Boolean, serverURI: String) { }
    override fun connectionLost(cause: Throwable?) {
        Log.w("MQTT", "Connection lost: $cause")
    }
    override fun messageArrived(topic: String, message: MqttMessage) {
        val payload = message.toString()  // получен JSON в виде строки
        try {
            val json = JSONObject(payload)
            val pressure = json.getDouble("pressure")
            val temp = json.getDouble("temperature")
```

```
            // Отображение значений на экране (TextView)
            runOnUiThread {
                pressureTextView.text = "Давление: $pressure бар"
                tempTextView.text = "Температура: $temp °C"
            }
            Log.d("MQTT", "Data received: P=$pressure, T=$temp")
        } catch (e: JSONException) {
            Log.e("MQTT", "JSON parse error: $e")
        }
    }
    override fun deliveryComplete(token: IMqttDeliveryToken?) { }
})
```

The above snippet uses the JSONObject class to parse a JSON string. The pressure and temperature fields are extracted, and runOnUiThread is used to update interface elements (e.g., TextView) and write a log of the received values. In the event of a connection loss, the connectionLost method will log the event, and the auto-reconnect option will ensure reconnection.

Example 2: WebSocket message handling. In a WebSocket connection, the server sends JSON-formatted model updates to the client. The handling code in Android might look like this – parsing the received string and updating the interface:

```
override fun onMessage(webSocket: WebSocket, text: String) {
    runOnUiThread {
        try {
            val json = JSONObject(text)
            val pressure = json.getDouble("pressure")
            val temp = json.getDouble("temperature")
            pressureTextView.text = "Давление: $pressure бар"
            tempTextView.text = "Температура: $temp °C"
            Log.d("WS", "Update UI with P=$pressure, T=$temp")
        } catch (e: JSONException) {
            Log.e("WS", "JSON parse error: $e")
        }
    }
}
```

Here, when a message is received in onMessage, the code is executed on the UI thread, which extracts the necessary values from JSON and displays them on the screen immediately. A record containing the received values is written to the log. Data processing for the REST API is implemented similarly (receiving JSON in response to an HTTP request, parsing it, and displaying it).

Interface and description of the Android application

Figures 2-5 show an example of the interface of the FacilityMonitor mobile application, designed to monitor oil and gas facilities and visualize data in real time.

1. Home page:
- the user sees the name of the application - FacilityMonitor, accompanied by the description: "Real-time data visualization for oil and gas facilities".
- the "Explore" button invites the user to explore the capabilities of the application.

2. Authorization:
- the user is asked to create an account to start monitoring objects.
- the screen includes two text fields: for entering an e-mail and a password, with corresponding icons for clarity.
- the "Join" button allows you to register or log in to the system.
- a link to go to the settings or authorization section (Access settings or Sign) is available under the button.

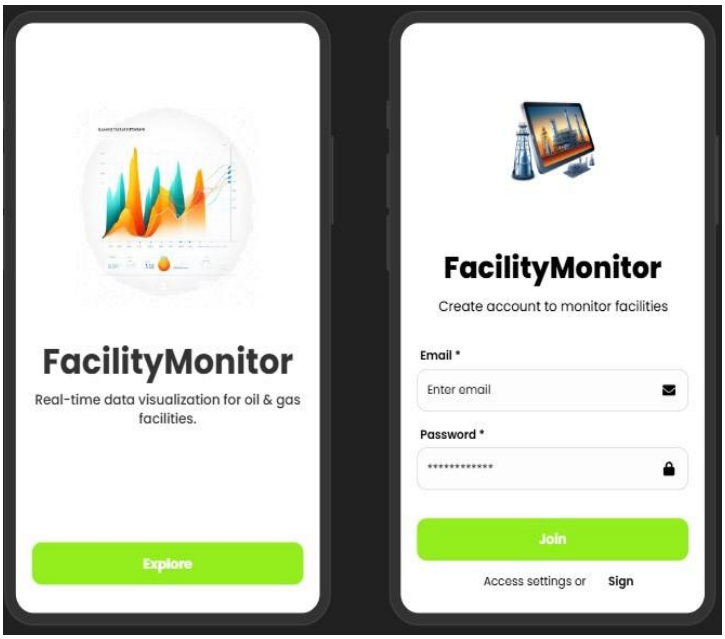

Figure 2. Login page.

3. Home page: the top of the screen displays the current date and an interactive block with real-time data visualization. Below, there are four buttons for quick access to the main functions:

- Charts: access graphical reports on key indicators, such as temperature, pressure, or flow.
- Alerts: view current warnings or critical events related to objects.
- History: archive data on the operation of objects for subsequent analysis.
- Metrics: overview of the performance and status of objects in the form of metrics.

The bottom navigation bar provides quick access to sections such as the home page, notifications, cameras, and others.

4. Notifications (divided by importance):

- Urgent: notification about urgent maintenance of the object.
- Enable monitoring: recommendations for setting up or activating application functions.
- Excess gas usage: warning about exceeding resource usage standards. Less urgent notifications (such as successful changes to settings or special offers) are highlighted with a gray background for visual differentiation. Each notification has an icon indicating its type (attention, settings, information, and offer). Arrows on the right indicate that notifications can be opened for more information.

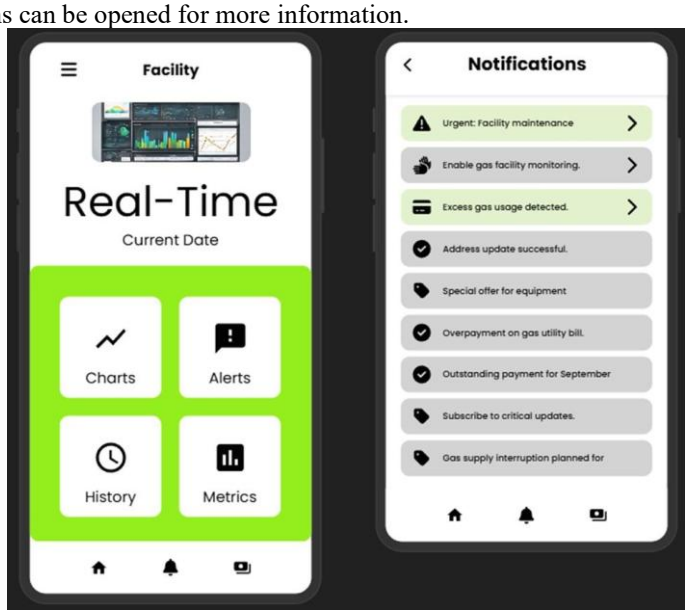

Figure 3. Home page and notification page.

5. Object metrics – this screen provides the user with key performance indicators of the oil and gas facility for the selected time period. Main interface elements are:

- Time range selection: at the top of the screen, there is an element for setting the data analysis period (e.g., "Jan 2023 - Feb 2023").
- "Production rate" graph: displays the current value (3.500) and the change dynamics (+0.75%) as a bar graph.
- "Gas flaring" graph: shows the gas flaring level (120) and its reduction by 1.20%, visualized on the graph.
- Other indicators:

A) Efficiency: (Empty example in the screenshot, but may include equipment efficiency data).

B) Pressure levels: includes several parameters such as pipelines, temperatures, flow rates, and warnings, displayed as scatter plots.

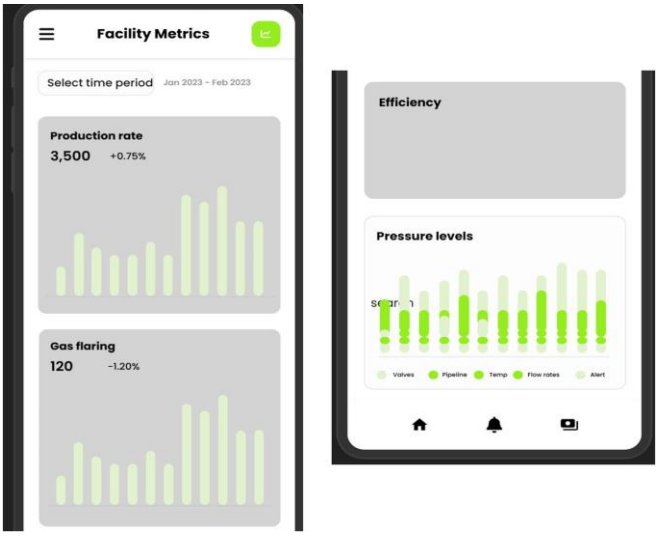

Figure 4. Statistics of productivity, gas combustion, pressure level, and profile.

6. Live Data Monitoring – provides the user with access to real-time visualization of key data on oil and gas equipment operation. Key interface elements are:

- Pressure Data: A graph displays real-time pressure changes.
- Temperature Data: A bar graph displays equipment temperature.
- Flow Rates: A scatter plot for analyzing flow rate changes.
- Data Tables – Displays key data: Pressure (1023 psi), Temperature (350°F), Flow Rate (3000 bbl/day), Humidity (12%).
- Refresh button: Refreshes the data on the screen.

7. Monitoring settings. The user can personalize the application. The main interface elements are:

- Monitoring Preferences:
- Slider for setting the data update frequency.
- Switch for activating notifications.
- Notification Settings.
- User Profile.

8. Statistics – designed for working with historical data on equipment performance. Main interface elements are:

Filters (select time range, object type, and data type):

A) Data Overview: line graph for visualizing changes in performance over the selected period.

B) Production Data:

- table with performance data for different objects;
- Export button: allows you to download data for further analysis.

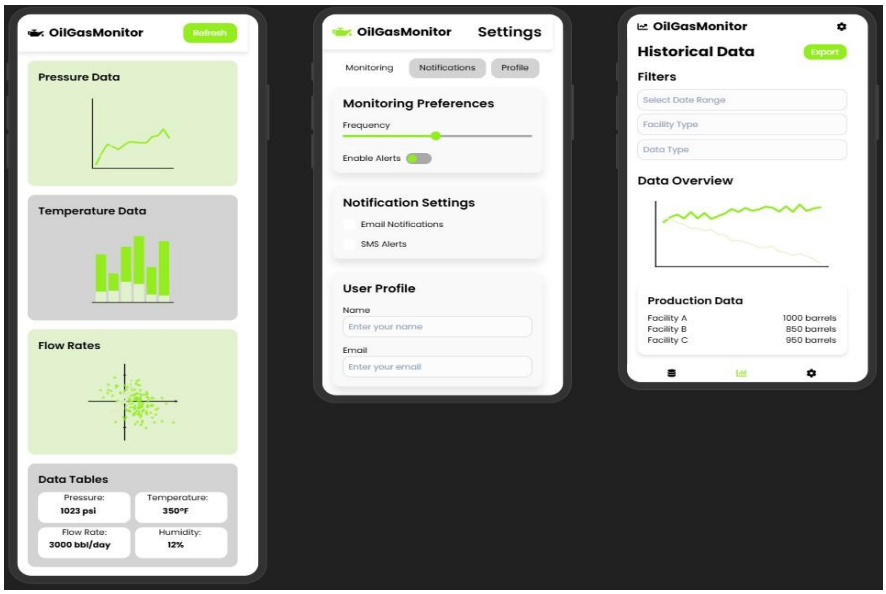

Figure 5. Graphs, settings, and statistics

Results

General assessment of the quality of the digital twin's operation in terms of efficiency, synchronization accuracy, reliability, and network load

To assess the quality of the "smart well - digital twin - application" link, the following parameters were selected: efficiency (data transfer delays), accuracy of model synchronization with the object, reliability of data delivery, and volume of transferred data. Table 2 below shows the average indicators characterizing these parameters at different stages of the system.

Table 2. Evaluation of the performance parameters of the digital twin "smart well – digital twin – mobile application".

Quality parameter	Smart Well (Sensor Transmission)	Cloud Digital Twin (Server)	Mobile application (client)
Data transmission delays	~0.3 sec to cloud (with persistent MQTT/WS connection), up to 0.34 sec with REST	Processing <0.1 s + sending ~0.3 s to the client (MQTT/WS), with REST latency ≥1 s (depending on polling)	~0.3–0.5 sec (MQTT/WebSocket, push updates), 1–5 sec (REST API with a polling interval of 1–5 sec)
Synchronization accuracy	Not applicable (data source)	High at low latencies (the digital twin lags behind the real object by no more than a fraction of a second); deteriorates at high lags (REST)	High for push protocols (displays in near real time), lower for REST (model gets stale between polls)
Reliability of delivery	MQTT: guaranteed (QoS and buffering, 0 lost messages); REST/WebSocket: possible losses on failures (3-5 messages per 5 s of failure)	MQTT: the broker ensures delivery even after the connection is broken; REST/WS: data sent at the time of failure is lost irretrievably	MQTT/WS: auto-receive all messages when connection is restored; REST: skip updates when server is unavailable (not received during failure)
Volume of data transferred	Low for MQTT/WS (message header 2-6B; ~60-70B per dimension considering protocol); High for REST (HTTP request ~500B)	Transferred per minute: MQTT ≈0.06 MB, WS ≈0.07 MB, REST ≈0.53 MB (tens of times more due to HTTP overhead)	Receives similar volume: MQTT/WS – minimal payload, REST – significant portion of traffic is HTTP headers

In the test scenario, the MQTT client did not lose any messages due to internal buffering and resending. WebSocket and REST API do not have a built-in ability to save unsent data: a connection failure resulted in the loss of several (≈4) messages that were not delivered to the server (Fig. 6).

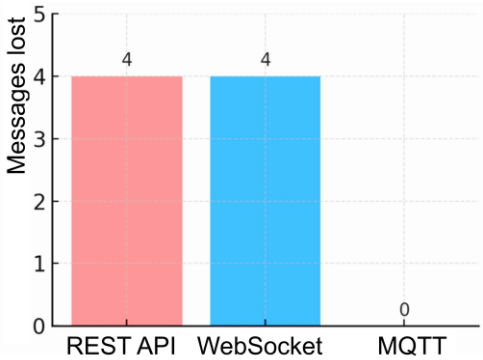

Figure 6. Reliability of message delivery for different protocols

These results demonstrate that MQTT has the highest reliability: even if there are connection interruptions, all messages eventually reach the recipient (albeit with a small delay). In contrast, WebSocket and especially REST API lost data during failures (about 3-5 messages did not arrive). In real-time systems, where the loss of even one critical signal is unacceptable, MQTT's reliability advantage is decisive.

A comparison of data transfer protocols based on their volume per unit of time is shown in Fig. 7.

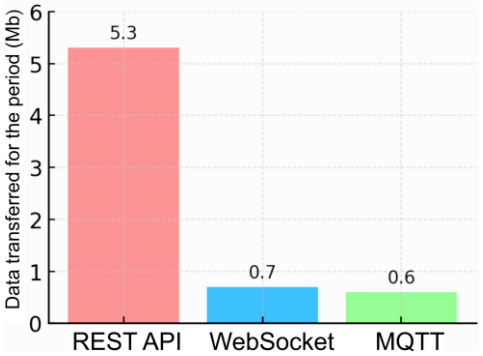

Figure 7. Volume of transferred data for the same period with different protocols

As shown in Fig. 6, MQTT transfers a minimal volume (~0.6 MB), WebSocket transfers a little more (~0.7 MB), and the REST API transfers an order of magnitude more (~5.3 MB). This experiment was conducted at a frequency of 1 message per second for a given period of time. It is evident that when using the REST API, network traffic is significantly higher due to HTTP headers and the overhead of each transaction. In the given example, REST consumed about 5.3 MB during the test, while MQTT only consumed ~0.6 MB, i.e., almost 9 times less. Thus, in conditions of a limited or charged communication channel, protocols with a permanent connection are significantly more efficient. In addition, a smaller amount of transferred data means lower power consumption on the device: according to observations, MQTT, all other things being equal, is ~20% more energy efficient than REST (due to the absence of frequent repeated connections and a smaller transfer volume).

Across all considered aspects (operability, synchronization accuracy, reliability, and network load), MQTT and WebSocket protocols demonstrated superiority over the traditional REST API. MQTT distinguished itself by the lowest delays and variations, as well as the highest data safety, making it the optimal choice for critical IIoT systems that require an uninterrupted telemetry flow. WebSocket also significantly exceeds HTTP/REST in terms of exchange speed and is suitable for web clients, although in the case of an unstable network, it may require additional code for guaranteed delivery.

Comparison of data transmission delay in different protocols

The tests showed a significant difference between the protocols in terms of data delivery speed. Fig. 8 shows a comparative distribution of "sensor → server" delays for the three protocols (under conditions of a virtually ideal network). It is clear that MQTT provides an average delivery time of about 290 ms, while the REST API has an average delay of about 340 ms. WebSocket occupies an intermediate position: the average time is ~310 ms. The spread of values (jitter) is important here: WebSocket had a more stable delay (standard deviation ~193 ms), while the spread of HTTP/REST reached ~308 ms. This means that under load, the REST connection provided very uneven response times – sometimes fast (≤100 ms), but periodically there were delays of up to 1–2 sec due to the overhead of establishing a connection and processing requests on the server. MQTT showed the lowest variance

(std ≈ 150 ms estimated, exact value not reported in this experiment), due to its persistent connection and lightweight protocol.

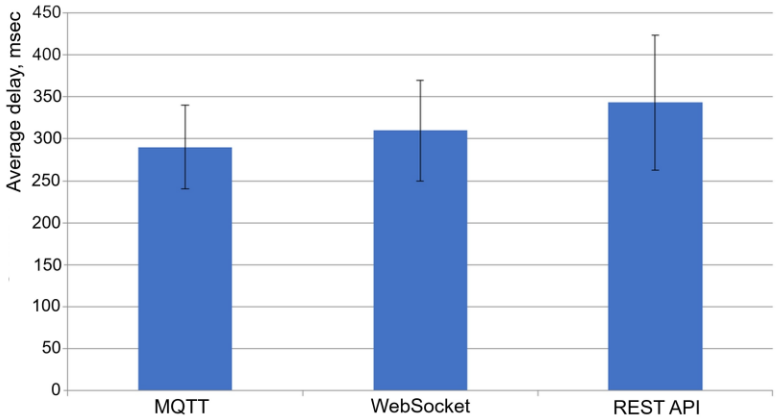

Figure 8. Average delay in data delivery from the sensor to the digital twin (lower is better). MQTT over TCP demonstrates the shortest response time of ~290 ms, WebSocket – about 310 ms, REST API (HTTP) – the longest ~343 ms. Error bars reflect the delay variation (standard deviation).

As network conditions worsened (introducing an additional 100 ms delay and 5% packet loss), all protocols showed an expected increase in time lags. However, relative differences remained: MQTT and WebSocket continued to operate in real time (the delay increased to approximately 400–500 ms), while REST API began to experience significant problems: with a polling interval of 1 s, the actual update delay reached 1.5–2 s, and when the polling interval increased to 5 s, the system completely ceased to meet the real-time criterion. In addition, in the degraded network mode, HTTP connections were more often interrupted and required re-establishment, which added overhead.

In our scenario, the simulation results were sent from the server to the client. For REST, this happened on a client request (polling); for MQTT, by publishing to a response topic; for WebSocket, by pushing via an established connection. The measurements here are similar to the previous ones: MQTT and WebSocket ensured almost-instant delivery (hundreds of milliseconds), while the REST solution resulted in a delay equal to the polling interval. We also tested REST in the long-polling variant, in which the client keeps the HTTP connection open for a period of time, waiting for data. This somewhat reduced the average delay (to ~1 s with a connection break every 30 s), but did not fundamentally solve the problem – without a permanent connection, it is difficult to achieve instant transmission of results from the server to the application.

When simulating short-term connection failures (a 5s network outage), the protocols behaved differently. In the REST API case, the sensor device experienced connection errors during the failure; after the network was restored, it resumed sending, but the data intended for transmission at the time of the failure was irretrievably lost. The WebSocket connection was broken when the network was lost; the device attempted to reconnect every 10 s and successfully restored the connection. However, messages that were not transmitted during the break were also lost (the server did not receive them). In contrast, the MQTT client automatically switched to offline mode when the connection with the broker was lost and stored incoming data in an internal buffer. The Paho library by default configures an auto-retry mechanism when the connection is restored. As a result, after a 5-second failure, the MQTT broker received all messages, albeit with a delay, while REST and WebSocket lost 3-5 messages. Thus, according to the reliability criterion, MQTT is a clear leader, especially important for systems where missing even a single critical message (for example, an emergency signal) is unacceptable.

MQTT messages turned out to be the most compact: one sensor measurement (for example, JSON {pressure:101.3, temp:55.0} of ~40 bytes) packed into an MQTT frame yielded a total of ~60 bytes, including headers. In the case of WebSocket, each message is encapsulated in a frame with a header of ~6 bytes + TCP/IP overhead, for a total of ~70 bytes. And when using REST, each value was transmitted in an HTTP request with headers, which amounted to ~500 bytes per transaction (including the POST start line, Host, Content-Type, Content-Length, etc. headers). Even if several measurements are combined into one request, the presence of HTTP overhead makes the protocol less efficient in terms of traffic for frequent updates. A practical measurement confirmed these calculations: for the same period of time at a frequency of 1 Hz, the device sent ~0.6 MB of data via MQTT, ~0.7 MB via WebSocket, and ~5.3 MB via REST. Thus, all other things being equal, the REST API consumes an order of magnitude more bandwidth than MQTT. For networks with paid traffic or with a narrow channel (for example, satellite), this can become a serious limitation.

It was noted indirectly that Android, in the mode of constant retention of a WebSocket connection or MQTT session, consumes slightly more energy than with rare REST polls. However, the difference was insignificant when updating once per second. Moreover, because MQTT sends less data, it is preferable for battery-powered

devices (e.g., an autonomous sensor on a battery): according to some data, MQTT transfers equivalent data ~20-25 times faster and ~20% more energy-efficient than REST. In our experiment, the ESP32 module worked stably throughout the MQTT test, whereas in the REST scenario, due to the volume of Wi-Fi traffic, the module heated up more. WebSocket has comparable power consumption to MQTT, though it has slightly higher overhead due to TCP session support.

Discussion

The obtained experimental data confirmed the expectations based on the theoretical properties of the protocols. In the context of efficiency (real-time), both MQTT and WebSocket are clearly superior to the REST API. A constantly open connection minimizes delivery delays by eliminating the request-response cycle. In a digital twin, this means that the virtual model will receive updates almost simultaneously with their occurrence on the real object. A difference of tens or hundreds of milliseconds, small at first glance, in total (with many parameters and long operations) leads to a more accurate correspondence between the twin state and the real object. On the contrary, REST, even with a high polling frequency, inevitably introduces delays and can lead to the accumulation of a model lag. For systems that require strict synchronization or instant response (e.g., emergency automation), the REST approach is not suitable; a push mechanism is needed.

MQTT also showed itself to be better in terms of reliability and fault tolerance. Its architecture, with a broker, acts as a buffer between the source and the receiver of data. Even if the digital twin or sensor loses connection with the broker for a short time, it will transmit the accumulated messages upon reconnection. In addition, MQTT provides built-in delivery receipts (PUBACK/PUBREC), which are critical for the guaranteed delivery of important telemetry. WebSocket lacks this level of abstraction: it is a “naked” connection that is either there or not. It is possible to implement an add-on (for example, application message confirmations and resend queues), but this complicates development. REST, in its basic form, relies on TCP reliability for each request, but if the connection between requests is broken, data can be lost. In the oil and gas industry, where facilities are often located in remote regions prone to communication interruptions, the reliability of the protocol is critical. Thus, for unreliable networks, MQTT provides the greatest resilience due to its architecture.

An interesting point is scalability and load. Here, WebSocket and MQTT also have an advantage: the HTTP protocol is redundant for traffic. As the number of devices and message frequency grow, the difference can become critical. For example, a hundred wells sending data every second via REST will create a load that not every server can handle (100 requests/sec, HTTP processing, parsing). An MQTT broker, on the other hand, easily routes thousands of messages because it operates on an event-driven model. Moreover, MQTT supports multipoint delivery: a single message from a sensor can be sent to multiple subscribers simultaneously (for example, to a digital twin, a data storage system, and a control center). In REST/WebSocket, such multicasting would require separate connections for each recipient. WebSocket is closer to MQTT in terms of connection efficiency, but remains more focused on point-to-point interaction (client-server). Thus, for scalability in wide IoT deployments, MQTT is the most suitable option, as confirmed by its popularity in industrial IoT platforms (e.g., Azure IoT, IBM Watson IoT, etc.) that support MQTT as the main protocol.

In terms of ease of integration, it is impossible to clearly identify a leader - the choice depends on the environment. If the project already uses web services with developed REST APIs, then integrating a digital twin via REST can be the fastest to implement (less new code, everything over HTTP). At the same time, modern IIoT solutions increasingly include MQTT brokers as a standard component. In our case, it was necessary to configure an additional broker, but many industrial controllers and SCADA systems have already "learned" to work with MQTT out of the box. As for WebSocket, its main advantage is direct work from the browser and applications without intermediaries. For example, if you need to build a real-time monitoring panel for operators, WebSocket will allow you to directly transfer data from the digital twin to the web interface in HTML/JavaScript. MQTT for a web client will require either MQTT-over-WebSocket (which many brokers support, essentially adding a WebSocket→MQTT layer within the broker) or an MQTT•REST gateway.

Therefore, WebSocket is very convenient for visualization and user interaction tasks. Ideally, you can combine the best sides: the digital twin receives data from the field via MQTT, and outputs it to the web dashboard via WebSocket (the twin itself, being on the server, is subscribed to MQTT and simultaneously acts as a WebSocket server for the frontend).

Finally, there is the issue of compatibility and standards. REST API is essentially an architectural style built on top of HTTP that easily traverses corporate networks, proxies, and firewalls. MQTT has historically used non-standard ports (1883/8883) that can be closed, but many brokers support working over WebSocket (port 80/443), which makes it easier to implement. WebSocket is a standard (RFC 6455) supported by web browsers, and WebSocket traffic is virtually indistinguishable from regular HTTPS once a connection is established, which is also convenient for passing firewalls. For oil and gas companies, where IT infrastructure may impose restrictions on protocol use, all three options are potentially acceptable, but in practice MQTT is increasingly included in IIoT standards. For example, the OPC Foundation consortium (developers of the OPC UA standard, widely used in

industrial automation) added specifications for an MQTT broker as a transport for industrial data. Also, AMQP and Kafka protocols are beginning to be used, but they are beyond the scope of this work.

Digital twins are becoming an integral part of increasing the efficiency of oil and gas facilities. The choice of data transfer protocol between real equipment and the computing module is critical. Three main methods of telemetry exchange are used in practice: REST API, MQTT, and WebSocket. Experiments have shown that REST APIs, which implement the request-response model, incur higher overhead with frequent polling and can introduce delays of 1–2 seconds in unstable networks.

In contrast, MQTT (Message Queuing Telemetry Transport) and WebSocket support persistent connections and reduce average latency to hundreds of milliseconds. MQTT also provides a guaranteed delivery mechanism (QoS) and message buffering at the broker, which is especially important in regions with communication interruptions. According to test results, the average data delivery time decreased from 340 ms (REST API) to 290 ms (MQTT), resulting in a digital twin that better matches the real object.

When developing oil and gas industry systems that require low latency and fault tolerance, it is preferable to use MQTT or WebSocket, with MQTT being optimal when increased delivery reliability is required.

Conclusion

The conducted study showed that the choice of data transfer protocol significantly affects the quality of digital twins in the oil and gas industry. For systems that are critical to delays and operate near real time, solutions based on MQTT or WebSocket are preferable. They provide a continuous data flow with minimal delays, enabling the digital twin to more accurately and promptly reflect the equipment's state. REST API is advisable to use where the requirements for efficiency are not so strict, or the web service infrastructure has already been established - for example, for periodic transfer of summary information, system configuration, and access to historical data. In real-world conditions, where communication can be unstable, MQTT with a broker provides the most reliable exchange: even in the event of failures, data is not lost, but reaches the recipient with a delay. WebSocket is inferior to MQTT in terms of reliability, but surpasses REST, and can be successfully used in combination with backup reconnection mechanisms.

For the practical implementation of a digital twin of an oil and gas facility, the following scheme can be proposed: telemetry from sensors is sent to an MQTT broker, which ensures collection and buffering. The digital twin (mathematical model) located on the server subscribes to the corresponding MQTT topics, constantly receives updates, processes them, and performs calculations. The model's results are transmitted to users. If users use a web interface, it is reasonable to transmit results via WebSocket directly to the browser (for real-time updates). If the results are sent to other systems (databases, analytical panels), they can be provided via the REST API via periodically updated endpoints or via MQTT (in separate topics). Thus, by combining MQTT for machine-to-machine data exchange with WebSocket/REST for interaction with a person, it is possible to build a hybrid architecture that leverages the strengths of each protocol.

Future work includes exploring additional protocols and technologies for digital twins. For example, comparing MQTT with AMQP, using CoAP (Constrained Application Protocol) for very limited devices, or implementing gRPC for high-performance services – all of this can provide additional information on how to optimize the real-object–digital twin connection. Another interesting direction is ensuring the cybersecurity of such a connection: protocols differ in their encryption and authentication capabilities, and the industry often raises data protection requirements. However, it is already clear that choosing an effective data transfer protocol is a cornerstone of successful digital twin deployment in the Oil & Gas industry.

References

Abdrakhmanova, K.N., Fedosov, A.V., Idrisova, K.R., Abdrakhmanov, N.Kh., Valeeva, R.R. (2020). Review of modern software complexes and digital twin concept for forecasting emergency situations in oil and gas industry. IOP Conf. Series: Materials Science and Engineering, 862, 032078. https://doi.org/10.1088/1757-899X/862/3/032078

Aleshina, O.G. (2023). Methodology for studying neo-industrial structural shifts in the economy under conditions of external shocks. Economics and Innovation Management, 4(27), 11-19. https://doi.org/10.26730/2587-5574-2023-4-11-19

Alimam, H. (2025). Digital Triplet Paradigm Based Brain Like Intelligence for Augmenting the Resilience of Intelligent Mechatronics, Towards Mitigating the Complexity of Cognitive Computing in the Oil and Gas Industry 5.0 Context. PhD Thesis, Ancona: Università Politecnica delle Marche, 220 p. https://iris.univpm.it/handle/11566/340575

Almatova B.G., Kupeshova A.S., Balmaganbetova F.T., Dosmagambetova M.B. (2025). Application of Artificial Intelligence and Automation in the Oil and Gas Sector. Oil and Gas, 1(145), 178-187. https://doi.org/10.37878/2708-0080/2025-1.11

Alsulaiman, N., Reddy, K., Odi, U, Rabines, J., Temizel, C. (2024). Opportunities in Utilization of Digital Twins in Unconventional Gas Fields: Enhancing Efficiency and Performance through Virtual Replication. Proceedings of the International Petroleum Technology Conference, Dhahran, Saudi Arabia, February 2024. https://doi.org/10.2523/IPTC-23176-MS

Amirkhanov, B., Amirkhanova, G., Kunelbayev, M., Adilzhanova, S.A. (2025). Evaluating HTTP, MQTT over TCP and MQTT over WEBSOCKET for digital twin applications: A comparative analysis on latency, stability, and integration. International Journal of Innovative Research and Scientific Studies, 8(1), 679-694. https://doi.org/10.53894/ijirss.v8i1.4414

Anaba, D.C., Kess-Momoh, A.J., Ayodeji, S.A. (2024). Digital transformation in oil and gas production: Enhancing efficiency and reducing costs. International Journal of Management & Entrepreneurship Research, 6(7), 2353-2131.

Azieva, R.H., Taymaskhanov, H.E. (2024). The use of digital twins in the oil and gas industry. European Proceedings of Multidisciplinary Sciences, 06(5), 30-38. https://doi.org/10.15405/epms.2024.09.5.

Bello, A.A., Fakeyede, M., Abaneme, O.G., Eshun, V., Akibor, C.J., Owusu, F. (2025). Optimizing agile collaboration frameworks for carbon-efficient digital twin deployment in oil and gas: strategies, tools, and challenges in the planning phase. Global Journal of Engineering and Technology Advances, 22(02), 034-045. https://doi.org/10.30574/gjeta.2025.22.2.0025

Bykova, V.N., Kim, E., Gadzhialiev, M.R., Musienko, V.O., Orudzhev, A.O., Turovskaya, E.A. (2020). Application of a digital twin in the oil and gas industry. Actual problems of oil and gas, 1(28), 1-8. https://doi.org/10.29222/ipng.2078-5712.2020-28.art8

Cherevko, M.A., Lukyanenok, P.P., Ligai, A.A. (2024). Analysis of innovative development of upstream segment enterprises in the context of global industry challenges. Economics and Innovation Management, 3(30), 59-69. https://doi.org/10.26730/2587-5574-2024-3-59-69

Correia, J., Rodrigues, F., Santos, N., Abel, M., Becker, K. (**2022).** Data Management in Digital Twins for the Oil and Gas Industry: beyond the OSDU Data Platform. Journal of Information and Data Management, 13(3), 1-17. https://doi.org/10.5753/jidm.2022.2506

da Silva, T.L., Chagas, U. (2023). How Digital Twins Is Being Used in Industry 4.0. In book: Digital Twin Technology - Fundamentals and Applications. IntechOpen, 1, 1-10. https://doi.org/10.5772/intechopen.113060

Dada, M.A. (2024). Digital Twins in the Upstream Oil and Gas Industry: Trends, Applications, and Challenges. NAPEC 2024, 16 October 2024, Oran. 2024, 1-11. https://doi.org/10.5281/zenodo.14250614

Dmitrievsky, A.N., Eremin, N., Basnieva, I.K. (2022). Digital modernization of oil and gas production in the conditions of reducing carbon footprint. News of Tula State University. Earth Sciences, 1, 467-467. DOI 10.46689/2218-5194-2022-1-1-467-476

Dyuthi, K.S.S. (2024). Configuring Real-Time Event Processing of Api Gateway with Aws and Websocket Api's. Journal of Informatics Education and Research, 4(3), 3324-337. https://doi.org/10.52783/jier.v4i3.1711

Eremin N.A. (2018). Digital Twin in the Oil and Gas Production. Oil. Gas. Innovations, 12, 14-17.

Ferrigno, E., Barsola G.A. (2023). 3D Real Time Digital Twin. Paper presented at the SPE Latin American and Caribbean Petroleum Engineering Conference, Port of Spain, Trinidad and Tobago, June 2023. https://doi.org/10.2118/213115-MS

Gandikota, R.A., Chennoufi, N., Hadad, T. (2024). Digital Twins Revolutionizing Oil and Gas Industry Optimizing Drilling Operations with Physics Informed AI. Paper presented at the ADIPEC, Abu Dhabi, UAE, November 2024. https://doi.org/10.2118/222587-MS

Gasanov, M.A., Gasanov, E.A., Ashvanyan, S.K., Zhavoronok, A.V., Zhironkin, S.A. (2024). Digital structural shift: an approach to analysis in modern economy. Economics and Innovation Management, 2(29), 23-34. https://doi.org/10.26730/2587-5574-2024-2-23-34

Jiangang, W., Lei, S., Ding, F., Jinli L., Lingxia H., Enming M. (2024). A digital twin modeling and application for gear rack drilling rigs lifting system. Sci. Rep., 14, 23711. https://doi.org/10.1038/s41598-024-73954-z

Khalkho, R. (2025). 3D Geological Modelling for Ore Deposit Characterization and Mine Planning. International Journal of Science and Research, 14(2), 707-711. https://www.doi.org/10.21275/SR25204203555

Knebel, F.P., Trevisan, R., do Nascimento, G.S., Abel, M., Wickboldt, J.A. (2023). A study on cloud and edge computing for the implementation of digital twins in the Oil & Gas industries. Computers & Industrial Engineering, 182, 109363. https://doi.org/10.1016/j.cie.2023.109363

Levina, A., Kalyazina, S., Levaniuk, D., Zaitsev, A. (2023). Application of Digital Twins for Digital Transformation of Oil and Gas Enterprises. Lecture Notes on Data Engineering and Communications Technologies, 157, 129-137. https://doi.org/10.1007/978-3-031-24434-6_13

Lukyanenok, P.P., Zhironkina, O.V. (2023). Factors of innovative development of oil and gas industry in transition to Industry 4.0. Economics and Innovation Management, 4(27), 77-85. https://doi.org/10.26730/2587-5574-2023-4-77-85

Matcha, S., Solanki, S. (2025). Best Practices for Building Scalable and Maintainable Web Services. International Journal of All Research Education and Scientific Methods, 13(1), 3620-3642.

Mayani, M., Gholami, S.M., Oedegaard, S.I. (2018). Drilling Digital Twin Success Stories the Last 10 Years. Paper presented at the SPE Norway One Day Seminar, Bergen, Norway, April 2018. https://doi.org/10.2118/191336-MS

Meza, E.B.M., Souza, D.G.B.d., Copetti, A., Sobral, A.P.B., Silva, G.V., Tammela, I., Cardoso, R. (2024). Tools, Technologies and Frameworks for Digital Twins in the Oil and Gas Industry: An In-Depth Analysis. Sensors, 24, 6457. https://doi.org/10.3390/s24196457

Mustard, S., Stray, O. (2023). The Role of the Digital Twin in Oil and Gas Projects and Operations. In Book: Crespi, N., Drobot, A.T., Minerva, R. (eds). The Digital Twin. Springer, Cham, 2023, pp. 703-732. https://doi.org/10.1007/978-3-031-21343-4_25

Nagornov, A.A. (2024). The leverage of digital twins in oil and gas production. In Proceedings of the conference: "Arctic: modern approaches to industrial and environmental safety in the oil and gas sector". Tyumen, 29 November 2023, Tyumen Industrial University, Tyumen, 2024, pp. 476-479.

Pandi, S. (2023). A Study on Building Blocks of Digital Twin for Oil and Gas Industry. International Journal of Engineering Research & Technology (IJERT), 12(11), 1-10.

Pishukhin, A.M., Lomukhin, I.A., Akhmedyanova, G.F. (2024). Digital twins of an oil producing enterprise. In Proceedings of the XIV All-Russian Conference on Management Problems (VSPU-2024). Moscow, June 17-20, 2024. pp. 2617-2621. Available online: https://vspu2024.ipu.ru/proceedings/2617.pdf (last access: 11.05.2025).

Rebello, C.M., Jäschkea, J., Nogueira, I.B.R. (2023). Digital Twin Framework for Optimal and Autonomous Decision-Making in Cyber-Physical Systems: Enhancing Reliability and Adaptability in the Oil and Gas Industry. arXiv, 2311, 12755. https://doi.org/10.48550/arXiv.2311.12755

Ren, S., Shen, F., Li, J. (2021). Revolutionizing Oil and Gas Production State Diagnosis with Digital Twin and Deep Learning Fusion Technology. Research Square, Preprint. https://doi.org/10.21203/rs.3.rs-3167023/v1

Shen, F., Ren, S.S., Zhang, X.Y., Luo, H.W., Feng, C.M. (2021). A Digital Twin-Based Approach for Optimization and Prediction of Oil and Gas Production. Mathematical Problems in Engineering, 3062841. https://doi.org/10.1155/2021/3062841

Shypul, O., Garin, V., Tkachenko, D., Zaklinskyy, S., Tryfonov, O., Plankovskyy, S. (2023). Development of a Digital Twin of Reservoir Filling by Gas Mixture Component. Lecture Notes in Networks and Systems, 667, 85-98. https://doi.org/10.1007/978-3-031-30251-0_7

Singh, S., Shehab, E., Higgins, N., Fowler, K., Tomiyama, T., Fowler, C. (2018). Challenges of Digital Twin in High Value Manufacturing. SAE Technical Papers, 2018-November, 1-10. https://doi.org/10.4271/2018-01-1928

Sudhakar, R. (2023). Robust Digital Twin Approach Towards Operational Excellence and Sustainability Goals. Paper presented at the ADIPEC, Abu Dhabi, UAE, October 2023. https://doi.org/10.2118/216747-MS

Titto, T.P., Ziatdinov, S. (**2022).** Enhancing Drilling and Production Through a Digital Twin Methodology. OnePetro, 1, SPE-211120-MS. https://doi.org/10.2118/211120-MS

Wanasinghe, T.R., Wroblewski, L., Petersen, B.K., Gosine, R.G., James, L.A., De Silva, O. (2020). Digital Twin for the Oil and Gas Industry: Overview, Research Trends, Opportunities, and Challenges. EEE Access, 8, 104175-104197. https://doi.org/10.1109/ACCESS.2020.2998723

Yusupbekov, N., Adilov, F., Ivanyan, A. (2022). Application of Digital Twin Theory for Improvement of Natural Gas Treatment Unit. Lecture Notes in Networks and Systems, 362, 505-513. https://doi.org/10.1007/978-3-030-92127-9_68