IoT

Article

Performance Analysis of OPC UA for Industrial Interoperability towards Industry 4.0

Marc Ladegourdie and Jonathan Kua *

School of Information Technology, Deakin University, Geelong, VIC 3220, Australia
* Correspondence: jonathan.kua@deakin.edu.au

Abstract: Open Platform Communications Unified Architecture (OPC UA) incorporates a wide range of features and covers most of the requirements for a platform-independent interoperability standard which can be used to transmit data and information from the factory production floor to the enterprise and management level. Due to its highly scalable and interoperable architecture, it is well-positioned for future deployment in smart embedded devices towards Industry 4.0, especially in environments where there are heterogeneous communication nodes. In this paper, we aim to evaluate the performance of OPC UA for communication in Industrial Internet of Things (IIoT) environments to better understand the technical implementation of OPC UA and the feasibility of incorporating OPC UA directly to resource-constrained edge devices. We propose an architectural system framework for OPC UA performance evaluation across a wide range of experiments. Our experimental results demonstrated the efficacy of the proposed system and evaluation framework. The OPC UA-based IIoT system architecture and budget-friendly/cost-effective testbed setup can be flexibly adopted for protocol testing, prototyping and educational purposes.

Keywords: OPC UA; TCP; Industrial IoT; Industry 4.0

check for updates

Citation: Ladegourdie, M.; Kua, J. Performance Analysis of OPC UA for Industrial Interoperability towards Industry 4.0. *IoT* **2022**, *3*, 507–525. https://doi.org/10.3390/iot3040027

Academic Editors: Carlo Giannelli and Amiya Nayak

Received: 29 September 2022
Accepted: 15 December 2022
Published: 19 December 2022

1. Introduction

The concept of fourth industrial revolution, i.e., Industry 4.0 is the confluence of industrial systems with advanced computing, sensors and communication systems where technologies such as Industrial Internet of Things (IIoT), Cyber-physical Systems (CPS), Artificial Intelligence (AI), Cloud/Edge computing and so forth are deployed [1–3]. One of the key application domains that Industry 4.0 promises to revolutionise is the manufacturing industry, making it smarter and and more efficient with the use of digital technologies [4,5]. Data exchanges and network communication between industrial devices/nodes in the envisioned smart factory/manufacturing environments present significant challenges [6,7].

Amongst the many implementation and deployment challenges, there is a lack of an interoperable standard for industrial communication [8]. There are only few standards and processes created to facilitate each entity to communicate in a "common language" [9]. New IIoT technologies are changing the traditional hierarchical structures and are providing a more scalable platform where data can seamlessly be exchanged. This process is often called bridging the Information Technology (IT) and Operational Technology (OT) gap [10]. IIoT focuses strongly on machine-to-machine (M2M) communication, abstracts industrial processes into data types, converts devices into data terminals, collects data from all the sources and uses the powerful computing capabilities of cloud and/or edge computing to perform deep data analysis to optimise operations [11,12].

The most common application scenarios of the IIoT include construction, manufacturing, agriculture, logistics, mining and energy production. These applications typically connect enormous amounts of devices to the internet to control mission-critical systems in high-risk industries where the slightest error could be catastrophic. Therefore, latency, reliability, scalability and security are the four fundamental requirements that IIoT must comply with to ensure seamless operations [13]. Open Platform Communications Unified Architecture (OPC UA) incorporates a plethora of features and covers most of the application requirements for a platform-independent interoperability standard which can be used to move data and information from a production floor to the enterprise level. OPC UA has been developed to address the heterogeneity and the incompatibilities of industrial transmission protocols (such as variants of fieldbuses and more IP-based protocols). Due to OPC UA's highly scalable architecture, it is well-positioned for future deployment in smart embedded devices. There are four core characteristics of OPC UA that positions it to become the Industry 4.0 standard of choice. It is internet ready, cross platform, enables complex information modelling and facilitates IT integration [14].

In this paper, we aim to design, implement and evaluate the performance of OPC UA in a self-contained and cost-effective testbed environment. We focus on understanding and evaluating the benefits and implications of OPC UA for industrial communication with multiple heterogenous nodes (as emulated in our testbed). Furthermore, we demonstrated the feasibility of incorporating OPC UA directly into commonly used embedded edge devices (such as Raspberry Pis), and contribute to the emerging body of work in developing a budget-friendly open source platform for experimental reproducibility, protocol design/testing/implementation, and educational purposes.

2. Background and Related Work

In this section, we present some background information on the ISA-95 automation pyramid, industrial IoT connectivity technologies, and related work on performance analysis of OPC UA.

2.1. ISA-95 Automation Pyramid

The ISA-95 automation pyramid represents pictorially the different automation levels in a typical factory and serves as a visual aid to comprehend how each level communicates with the other using different technologies [15]. The pyramid comprises of five distinct levels as shown in Figure 1.

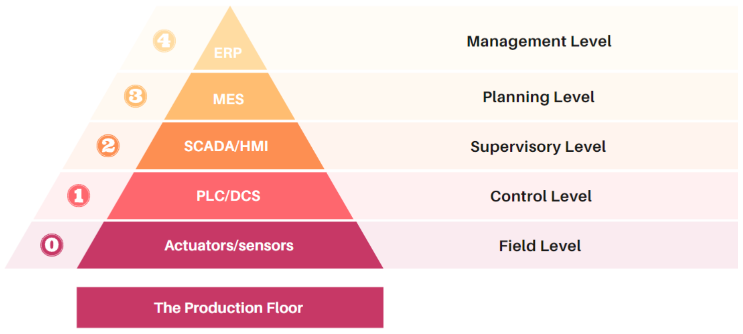


Figure 1. The ISA-95 automation pyramid (adapted from [15]).

- Level 0—Field: This level consists of a variety IIoT sensor devices and actuators on the factory floor [16].
- Level 1—Control: The control level is referred to as the brain behind the production floor. It is generally composed of multiple Programmable Logic Controllers (PLCs). A PLC is an Industrial Control System (ICS) that continuously tracks the state of input devices and controls the state of output devices based upon decisions taken by a custom program [17].
- Level 2—Supervisory: The two most prevalent technologies used at this level are Supervisory Control and Data Acquisition (SCADA) systems and Human Machine Interfaces (HMI). SCADA is a system constituting of both hardware and software components used to monitor and control industrial processes. It allows manufacturers to collect and inspect production data, monitor and manage alarms, and issue automatic

control responses triggered by different events [18]. HMIs are dashboards or screens that translate complex data into understandable information and used to control the machinery [19].

- Level 3—Planning: The next level of the pyramid contains the Management Execution System (MES). An MES is an information system that monitors and tracks the production process of goods on the factory floor. The main goal is to ensure effective execution of the manufacturing operations and improve production output [20].
- Level 4—Management: The management level is built around the Enterprise Resource Planning (ERP) system. An ERP refers to a software that organisations use to manage everyday activities such as manufacturing, supply chain, compliance, finance, procurement, services, and more [21].

2.2. Industrial IoT Connectivity

Communication is a key enabler of IIoT. Different wired and/or wireless technologies are adopted depending on which combination of backbone and edge architecture is required. Neither the five-layer Internet model nor the seven-layer Open System Interconnect (OSI) model are appropriate to convey the distributed nature of sensors, controllers, actuators, gateways, and the rest of the components that form the IIoT. Therefore, it requires a different layered model. The lower levels of the stack must comply with the requirements of being flexible and scalable and the upper levels must enable smart devices to transport data whilst ensuring that the semantic rules are correctly understood at the destinations. In view of this, three macro layers can be established. The networking layer, manages the packets and frames; whereas the connectivity layer dealing with messages and the information layer handling the end-user data structures [3,22–24].

Similar to the wider computer data networks, Internet Protocol (IP)-based protocols (such as IPv4 and IPv6) are also the common enabler for applications and communication protocols in IIoT. The majority of current industrial applications rely on fieldbuses (which connect and control a network of input and output devices in real-time), each having their own ecosystem, consequently contributing to poor interoperability [25]. There are different variants of Fieldbus solutions, i.e., Modbus, Profibus etc. Ethernet and IP protocols are now natively supported by the latest technologies which makes it easier to provide technical interoperability as it is built on the IP suite. For instance, EtherNet/IP is gaining traction arising from the development and improvement of associated protocols. In addition to having TCP/UDP at the transport layer, HTTP can access higher-tier functionalities [3].

The upper layers of the IIoT stack aim to enable and facilitate interoperability, i.e., the ability to use a set of rules and a common data structure to exchange information. The Industrial Internet Consortium (IIC) separates the upper layer protocols into two different levels. The transport layer is the lower level responsible for exchanging variable length messages between the concerned applications and the framework layer is the upper level that manages the exchange of structured data. The transport layer is loosely mapped to that of the OSI model (where TCP and UDP are implemented).

Recent solutions for interoperability implement horizontal integration leverages the use of messaging protocols that supports the publisher/subscriber model. Typically, both sides of the data exchange are not connected. The application publishing the message data needs to connect to a message queue broker to place the message in a queue and the subscribers on the other hand automatically receive the message. The advantage of using this technology is that it ensures scalability since the applications do not have to know each other to communicate. The prevailing messaging protocol up till date is Message Queue Telemetry Transport (MQTT) [26]. A different method of implementing the functionalities previously described is using a protocol referred to as the Constrained Application Protocol (CoAP) [27] CoAP relies on a request/response model where synchronous or asynchronous data exchanges are allowed. Scenarios involving synchronous data exchanges requires that the requestor wait for replies before issuing a request. The reply is returned to the requestor randomly in asynchronous data exchanges.

The Open Platform Communications Unified Architecture (OPC UA) protocol standard has been recently developed as an important facilitator for industrial interoperability, as discussed in detailed in a recent Informational Internet Draft by the Internet Engineering Task Force [28]. OPC UA is a Service Oriented Architecture (SOA) which relies on a client/server architecture in which the server models the data, information, processes and systems as objects that are then relayed to the clients in combination with all the available and usable services [3]. Recent innovations also investigated extending and implementing OPC UA with Pub/Sub extension over TSN (or similar) networks for real-time industrial communication [29].

2.3. OPC UA and Related Protocol Standards

OPC UA was proposed and designed to create a replacement for all existing Component Object Model (COM)-based specifications without losing any characteristics or performance. It gathers all the necessary and successful features of OPC Classic, fixes the known issues and packages it into a more flexible version that provides standardisation for more additional use cases. OPC UA incorporates more features and covers all the requirements for a platform-independent interoperability standard which can be used to move data and information from the production floor to the enterprise. Due to its highly scalable architecture, it is well-positioned for future deployment in smart embedded devices and offers simple migration strategies to integrate legacy OPC products based on the previous OPC standards. There are four core characteristics of OPC UA that differentiates it from OPC Classic and positions it to become the Industry 4.0 standard of choice [14].

- Internet ready and cross platform: OPC UA is no longer dependent on COM and DCOM which implies that any OPC UA application can easily be deployed across multiple computing platforms such as sensors PLCs, embedded controllers, gateways etc. Additionally, it is internet ready and firewall friendly because of the utilization of protocols such as HTTP. These protocols are actively used and do not require any additional ports to be opened on the firewall. This ensures that the production equipment are embedded with ability to exchange information securely and seamlessly over the internet [14].
- Complex Information Model: OPC UA comes equipped with rich and extensible data modeling capacities allowing developers to expose machine or sensor information in a substantially more complex format than what was prior impossible with OPC Classic. For example, OPC Classic allowed automation data to be expressed in its purest form i.e., the temperature value from a sensor. OPC UA allows the developers to expose the units of measurement, the temperature set-points, the type of temperature sensors, the instrument configuration parameters, the position in the hierarchy of machines and define all other components of that machine and how they are related to the temperature sensor in order to give the OPC UA client a holistic view of the machine or plant information. In other words, it is a complete digital description of an underlying physical asset. Despite the complex information modeling capabilities, OPC UA is still able to support simple data models such as those found in OPC Classic [14].
- Service Oriented Architecture (SOA): M2M Legacy protocols such as those mentioned previously, Fieldbus, Profibus and Modbus rely on data exchange based on the bits and bytes that are transmitted back and forth. In order to request information from a device or machine a specific bit or byte sequence has to be sent. This is not a user friendly way of building communication systems. In contrast an OPC UA Server exposes services in the form of methods that a client can use to request information from a server. These methods can be used for performing administrative tasks such as the FindServer() method and additional methods to access automation data, for example the ReadTag() and WriteTag() method. Using an SOA simplifies the engineering effort in terms of development and maintenance of communication systems as methods can easily be read by humans and programmed into machines for consuming automation data [14].

The most popular Industrial Ethernet protocols are Ethernet/IP, Modbus TCP, Profinet and Ethercat. These protocols must natively support the fieldbus protocols and the TCP/IP protocols that are present in IT. In this context, OPC UA is more than just a simple communication standard for real-time communication in automation. It is an SOA compatible with both IPv4 and IPv6. Combined with all the other advantages listed above, OPC UA represents an All-in-One solution [30].

- IT Integration Facilitated: Devices or sensors that are found at the field level of the ISA-95 automation pyramid using fieldbus as the communication protocol, are not able to communicate directly with a high level application such as an ERP system. The information needs to be collected by PLCs then transmitted to SCADA systems and then finally pushed up the stack to IT networks. OPC UA is an information carrier that is common throughout all the levels of the automation pyramid. As a result, it makes the automation pyramid obsolete since devices can deliver data directly to higher levels. In addition, OPC UA enables the use of non-cryptical process logic from the cloud [14]. Therefore, a different approach to digital transformation is possible leveraging OPC UA as shown in Figure 2.

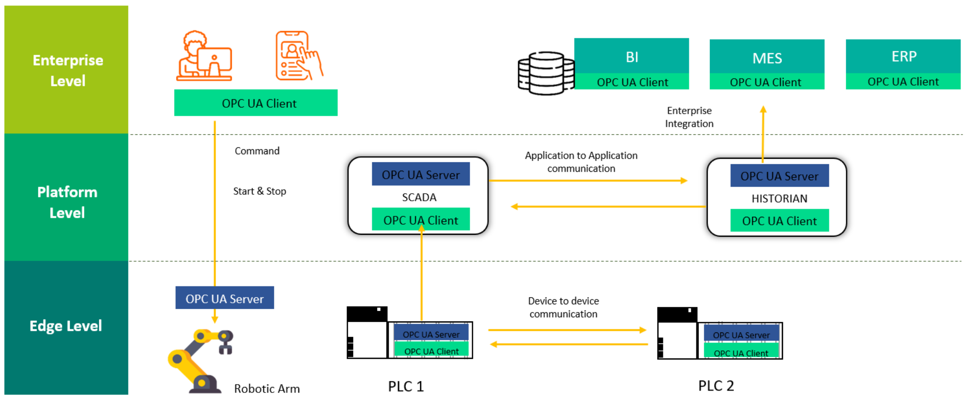


Figure 2. Digital transformation leveraging OPC UA.

As illustrated in Figure 2, OPC UA is used at the different levels of the three tier architecture. It is common to have a system that contains both an OPC UA Server for exposing system information and an OPC UA Client for accessing other servers. Furthermore, OPC UA and MQTT can be implemented together when digitally transforming a business since OPC UA is efficient at communicating horizontally with PLCs, SCADA systems and edge gateways or controllers and MQTT is performant at handling the platform and enterprise level.

The two transport protocols defined by the OPC UA standard are Unified Architecture Transmission Control Protocol (UA TCP) and Simple Object Access Protocol (SOAP)/Hypertext Transfer Protocol (HTTP). They are both used to establish a connection between an OPC UA Server and an OPC UA Client at network level [14].

Using UA TCP for the transport layer results in a fast and simple network communication [31]. There are multiple requirements contributing to the design decision to define a lightweight protocol on top of TCP. It is necessary to negotiate buffer sizes for sending and receiving data that are configured at the application level. Different OPC UA server endpoints should be enabled to share a single IP-Address and port. The structure of an UA TCP message chunk is generally composed of a message header and a message body. The information about the type and the length of the message is contained in the message header. The message body comprises of either UA TCP specific connection messages used to establish a socket connection or exchange connection error information and the secured and encoded service messages that are forwarded to the upper layer.

There are three types of UA TCP messages [32]. The "Hello" message is sent by the OPC Client to the OPC Server when a socket connection to a specific endpoint provided by the server is established. Consequently, it requests a particular buffer size to send and receive data from the server as well as maximum chunk and total message lengths. An acknowledge message is sent by the OPC UA Server to the Client in response to the hello message, confirming or revising the requested buffer sizes in combination with the chunk and message lengths [14].

From a security and reliability perspective, an agreement on these numbers is crucial so that Clients and Servers know what to expect from each other and can prevent certain attacks such as Denial-of-Service (DoS) or buffer overflows. The last message type is the error message which provides error information to other applications. These messages are sent around whenever a connection problem arises. UA TCP provides a specific error recovery mechanism to enable OPC UA sessions to survive any potential network interruptions. When the client loses the socket connection, a new socket is generated and assigned to the existing secure channel which first requires re-authentication by the server. All the pending requests from the client and the responses from the server are buffered until the generated socket is available. On the other hand, if the error recovery fails an error message will be sent around which allows the application layer to react to it [14].

SOAP/HTTP (i.e., SOAP over HTTP) is a widely accepted communication format in the web service environments due to its simplicity and firewall-friendliness [33,34]. No additional port needs to be opened in the firewall because the standard ports for HTTP is already used for transportation. This signifies that OPC UA applications can securely communicate with one another through Internet in a similar fashion that web browsers communicate with web servers. SOAP is a network protocol used to call remote procedures and exchange data between systems. It relies on various standards for example XML to correctly represent the data and HTTP or TCP for the transportation of data. A SOAP message is structured in the header and in the body. The headers are composed of the address and routing information whereas the body encapsulates the required payload to be transported. The secured and encoded service message is then enclosed by the body of the SOAP message. Despite the SOAP message being an XML-based data structure, it is still capable of transporting both an XML encoded service messages and UA Binary encoded messages. As pointed out above, HTTP is used for the transportation of SOAP messages. The procedure involves the transmission of the OPC UA service request embedded in SOAP requests in the body of an HTTP POST request. HTTPS can be used to secure the exchanged data. However, if one of the secure channels is already in use, it results in an unnecessary overhead [14].

2.4. Related Works

This section reviews the papers most relevant to our work, more specifically those evaluating the performance of OPC UA under various settings.

Profanter et al. [35] performed a comprehensive evaluation and performance comparison of OPC UA, Robot Operating System (ROS), Data Distribution Service (DDS), and Message Queuing Telemetry Transport (MQTT) in IIoT environments. The authors compared the protocol features, their performance, and the implications of protocol overheads on the performance. The authors conducted experiments using open-source protocol implementations under different network configuration and settings. They measured the Round Trip Time (RTT) of the messages in three different system states (idle, high CPU load, and high network load) in a network with up to 500 nodes on the same host. The same authors [36] later also proposed a hardware-agnostic industrial system architecture (called 'Plug & Produce' system) that leverages OPC UA to semantically linked the robots, factory tools, and other system components in the workcells (which are manufactured by various manufacturers which relied on their specific control interfaces). Their evaluation demonstrates that OPC UA (specificially OPC UA discovery) is well suited for the Plug & Produce concept [37], which resulted in the improved efficiency and reduction in changeover times.

Cavalieri et al. [38] investigates the performance evaluation of the standard but focuses strictly on the data exchange mechanisms that influence client/server communication.

The study is carried out in a simulation environment of an OPC UA client/server data exchange model, built using OMNeT++ framework. Furthermore, the performance of the security mechanisms of the OPC UA standard is investigated in [39,40]. Rocha et al. [41] compares the performance of OPC UA against MQTT for data exchange and focuses on packet/payload length, loop back and response times. These evaluation lack the implementation of a real hardware architecture that incorporates the use of factory devices along with the corresponding software development environment to perform a hands on analysis of OPC UA performance.

Fangjian et al. [42] built a resource-constrained evaluation platform for OPC UA with STM32 and FreeRTOS. They implemented OPC UA servers on the platform and investigated the impact of limited memory space on OPC UA's performance. They analysed the performance of OPC UA in such environment and provided recommendations on how to use the proposed platform with OPC UA more widely to support a variety of industrial manufacturing applications.

To address the cost and implementation challenges of OPC UA in a large factory, Cho et al. [43] investigated the use of an OPC UA gateway that conforms to the OPC UA specification with an ARM processor. The authors analysed OPC UA performance through the gateway using indicators such as publication and sampling intervals, subscription limits, and security/encryption guidelines. Their experimental results demonstrated a two-thirds reduction in operating costs. They also tested the gateway with a Cloud solution with Microsoft Azure, as a way to enable cost reduction in older/legacy systems.

The work in [44] aims to evaluate the performance of OPC UA using a test platform to perform the experiments. The author briefly describes that Microsoft Windows 7 based PLCs are used. However, the work does not describe the platform architecture nor does it convey any information on how to replicate a similar testbed. In [45], the development of an IoT testbed leveraging OPC UA is proposed. However, the test platform makes use of expensive industrial equipment that connects to a central OPC UA server hosted on a Raspberry Pi 3B. Despite the true representation of an industrial environment and the efficiency of the suggested prototype, it is hard to replicate the setup in an educational context due to financial constraints. Part of the analysis and results presented have already been published in [46]. Other works investigated the use of low-cost and resource-constrained Internet of Things devices and edge computing paradigms as an architecture to enable cost-effective testing of communication protocols [47–49].

In this paper, we aim to address the gaps in the literature by focusing primarily on the standard when integrated to edge level devices. We also aim to provide recommendations on the feasibility of incorporating OPC UA on low power/performance devices. Furthermore, this paper contributes to existing studies by proposing an open source test platform that is budget-friendly/cost-effective and easily replicated for testing, prototyping and educational purposes.

3. Experimental Methodology

To evaluate the performance of the standard, we have built a hardware-based experimental testbed. The testbed architecture had to resemble that of a modern manufacturing plant and the artefact was required to produce a substantial load on the network imposed by multiple PLCs to replicate a scenario that would typically occur at the edge level.

Note that in this work, we are using the OPC UA client/server communication model, which are currently well-defined and matured in implementations such as Prosys (https://www.prosysopc.com/, accessed on 29 September 2022), Free OPC-UA (https://github.com/FreeOpcUa, accessed on 29 September 2022), and open62541 (https://www.open62541.org/, accessed on 29 September 2022). The OPC UA pub/sub model is emerging and currently undergoing active developments.

3.1. *Architectural System Design*

Our experimental platform comprised of five PLCs each having their own OPC UA server. A variety of nodes, including a sensor was connected to each PLC. An OPC UA client mimicking a SCADA system, communicated via OPC UA TCP/IP to each PLC to retrieve the corresponding data generated by the nodes as demonstrated in Figure 3. It is worth noting that the client could similarly have been an HMI, ERP or MES system.

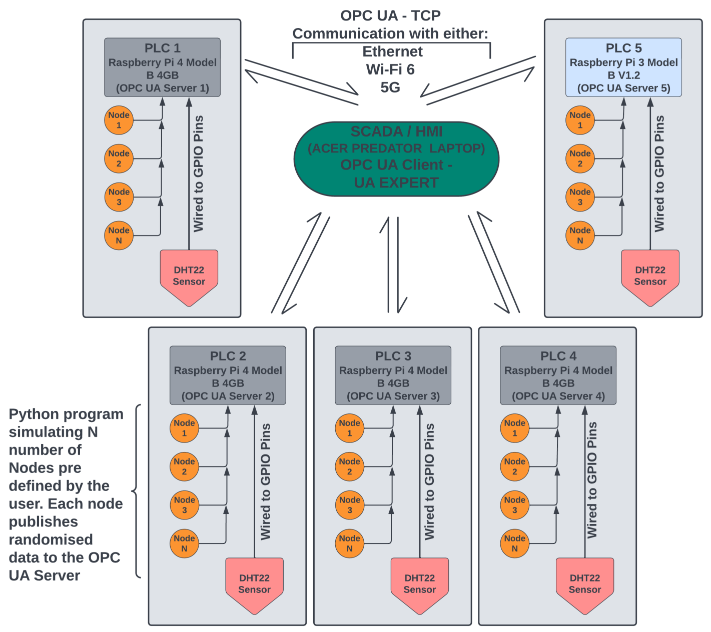

OPC UA - TCP
Communication with either:
Ethernet
Wi-Fi 6
5G
PLC 1
Raspberry Pi 4 Model B 4GB
(OPC UA Server 1)
PLC 5
Raspberry Pi 3 Model B V1.2
(OPC UA Server 5)
Node 1
Node 2
Node 3
Node N
Wired to GPIO Pins
DHT22 Sensor
SCADA / HMI
(ACER PREDATOR LAPTOP)
OPC UA Client -
UA EXPERT
PLC 2
Raspberry Pi 4 Model B 4GB
(OPC UA Server 2)
PLC 3
Raspberry Pi 4 Model B 4GB
(OPC UA Server 3)
PLC 4
Raspberry Pi 4 Model B 4GB
(OPC UA Server 4)
Python program simulating N number of Nodes pre defined by the user. Each node publishes randomised data to the OPC UA Server

Figure 3. System Design Architecture.

3.1.1. Programmable Logic Controller (PLC)

Actual/real-world PLCs can be costly (e.g., Siemens S7-1200 is a commonly used PLC and can very costly while scaling to build a large testbed). For prototyping and educational purposes, building an IIoT environment with actual PLCs is not realistic, hence an alternative computerised control system had to be sourced. A microprocessor such as the Raspberry Pi proved to be the best alternative considering the price-performance ratio, its ability to emulate the functionalities of a PLC and the fact that it provided both onboard Ethernet and Wi-Fi connectivity.

To construct the platform, four Raspberry Pi 4B 4 GB and one Raspberry Pi 3B V1.2 were chosen. Table 1 summarizes the differences between both. It was important to include a Pi with lower specifications to monitor the performance difference between both models.

Table 1. Technical specifications of the Raspberry Pis implemented in our testbed.

Specifications	Raspberry Pi 4B 4 GB	Raspberry Pi 3 B V1.2
CPU	Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5 GHz	Broadcom BCM2837, Quad Core Cortex-A52 (ARM v8) 64-bit SoC @ 1.2 GHz
RAM	4 GB LPDDR4-3200 SDRAM	1 GB LPDDR2-1400 SRAM
Wi-Fi	2.4 GHz and 5.0 GHz IEEE 802.11ac wireless	2.4 GHz IEEE 802.11b/g/n wireless

3.1.2. OPC UA Server

A wide range of OPC UA servers are available, for instance, the Prosys OPC UA simulation server [50] or the Umati sample server [51]. However, none of them possessed the appropriate tools to measure performance, thus, did not satisfy the context of the experiment. As a substitute, an OPC UA server had to be created from scratch using an open source library. Amongst the usable libraries, namely, open62541 [52], NodeOPC UA [53], freeopcua–opcua python [54] and freeopcua–asyncua [55], the hindmost was favored. It boasted a high performance benefit in comparison to the others as it is built on top of the asyncio python library which offers concurrency coding utilising the async/await syntax. Moreover, the asynchronous framework was referenced as being an ideal fit for input and output bound applications [56].

3.1.3. Communication Nodes

There were two approaches to generate sufficient data to impose load on the network. Practically, a large number of sensors, between 50–60 could have been connected to feed more data to the Pi. However, this would have been a costly method that would have introduced data redundancy to the equation. In addition, a total of 60 variables per Pi would not have been a true representation of a factory scenario. Instead, a script was programmed in python to generate a user defined number of nodes each containing 10 sensor simulated values.

3.1.4. Sensors

On each PLC, a real sensor had to be connected to serve as a control for the simulated sensors. The DHT22 digital temperature and humidity sensor was selected as the most desirable sensor to replicate the ones found in a standard factory because of its interesting properties. It was inexpensive, compact, user-friendly and could sense two physical properties at once by using a thermistor to measure the air temperature along with a capacitive humidity sensor to output a digital signal to a data pin [57].

3.1.5. SCADA

In practice, a powerful computer is required to run a SCADA software (OPC UA client). Therefore, an Acer Predator Helios 300 laptop equipped with an Intel Core i7-7700HQ, 16 GB of DDR4 RAM and an install of Windows 10 64 bit Pro version was utilised as the SCADA system. A custom made OPC UA client could have been built using the asyncua python library. However, this technique was found to be a tedious and time consuming process considering that readily available OPC UA clients had useful features such as the automatic detection of nodes and variables. Consequently, the Prosys OPC UA Browser software was chosen from the list of all clients on the market, for example, UA Expert, Kepware's OPC UA Client, OPC UA Commander among others [58]. This software was particularly chosen for its unique ability to graphically visualise, in real time, the data changes of any selected variable [59].

3.2. Experimental Testbed Setup

In this section, we describe how we build our experimental testbed, which includes the hardware setup and also the software configuration of OPC UA servers and the communication nodes.

3.2.1. OPC UA Server Configuration

The OPC UA servers are built using Python version 3.10.4. As illustrated in Figure 4, both the asyncio and the asyncua libraries were imported. The main asynchronous function was defined, containing several instructions to setup the server. The server was initialised using the "server.init" function and the endpoint URL was passed as a parameter to the "server.set_endpoint" function in the form of "opc.tcp://0.0.0.0:4840", where "opc.tcp://" represents the protocol identifier, "0.0.0.0" defines the Raspberry Pi's IPV4 address and 4840, a unique and unutilised TCP port number. The namespace was registered using the "register_namespace" function and passing the OPC UA server ID was as a parameter.

Lastly, the server was asynchronously started. It is to be noted that the default encoding, OPC UA Binary was chosen for its efficiency of fast encoding and decoding of the data.

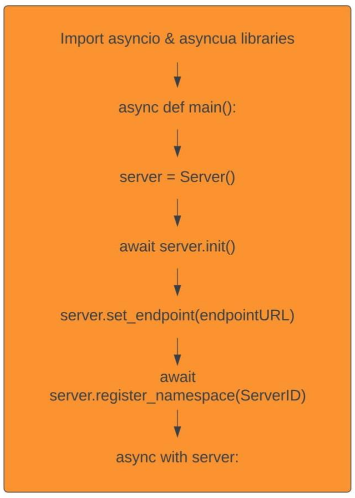

Import asyncio & asyncua libraries
async def main():
server = Server()
await server.init()
server.set_endpoint(endpointURL)
await server.register_namespace(ServerID)
async with server:

Figure 4. Configuration process of the OPC UA Server.

3.2.2. Nodes Configuration

The OPC UA Specification defines the Address Space as a hierarchical structure composed primarily of object nodes, followed by variables and lastly their associated data value as showcased in Figure 5. Each hierarchy level contains specific references allowing them to be linked to one another [60].

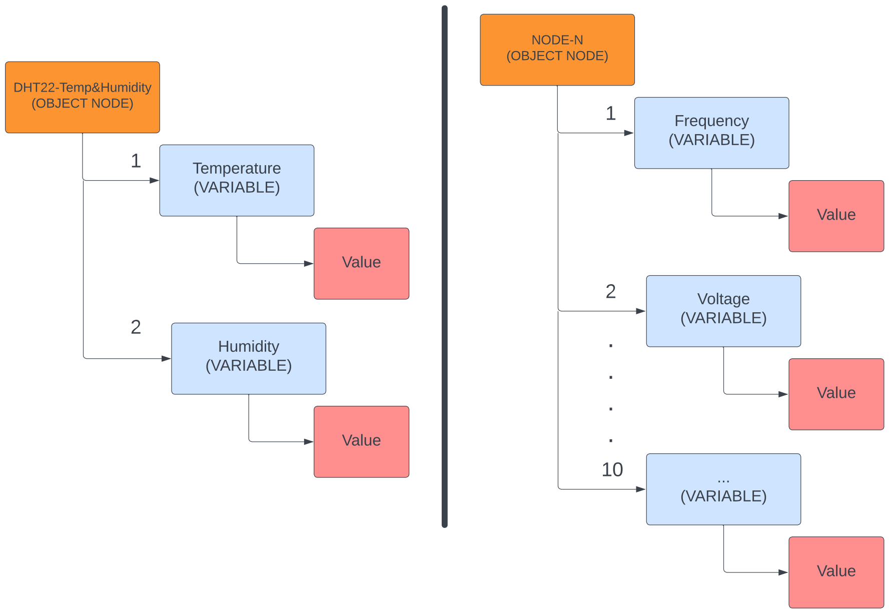


Figure 5. Data model of the Address Space.

3.2.3. DHT22 Sensors

For a practical implementation of the DHT22 sensor, a breadboard was used to make the connections. The first step was to place a 10K resistor between the first pin referred to as the Voltage Common Collector (VCC) and the second pin termed the data pin. The resistor served as a pullup from the data pin to the VCC pin. The VCC pin was connected to pin 1 on the Pi, the data pin to pin 7 and the last pin on the sensor to any of the ground pins on the Pi, for example pin 6 as shown in Figure 6.

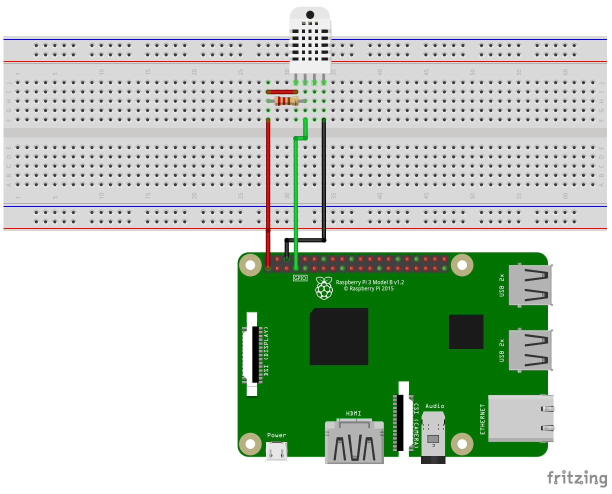


Figure 6. DHT22 Sensor connection diagram.

As illustrated in Figure 7, the sensor object node was added to the OPC UA server using the "server.nodes.objects.add_object" function from the asyncua library and passing in the namespace and the name identifier as parameters. Moreover, two variables, specifically temperature and humidity were added to the sensor object node using the "add_variable" function and passing in the namespace, name identifier, value and the data type. The variables were then set to writable using "set_writable". Finally, two asynchronous functions were defined to read and write the sensor data to the corresponding variables. The temperature value was acquired employing the ".temperature" function from the adafruit library and written to the variable using the "write_value" from asyncua. A similar approach was adopted for the asynchronous humidity function making use of the ".humidity" instead.

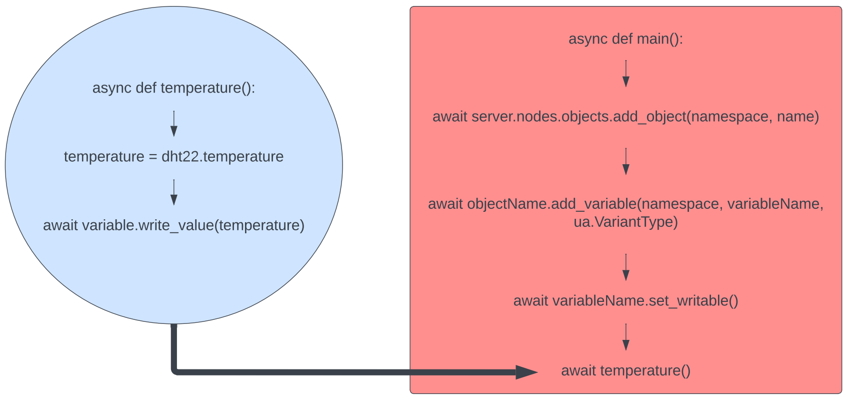

async def temperature():
temperature = dht22.temperature
await variable.write_value(temperature)
async def main():
await server.nodes.objects.add_object(namespace, name)
await objectName.add_variable(namespace, variableName, ua.VariantType)
await variableName.set_writable()
await temperature()

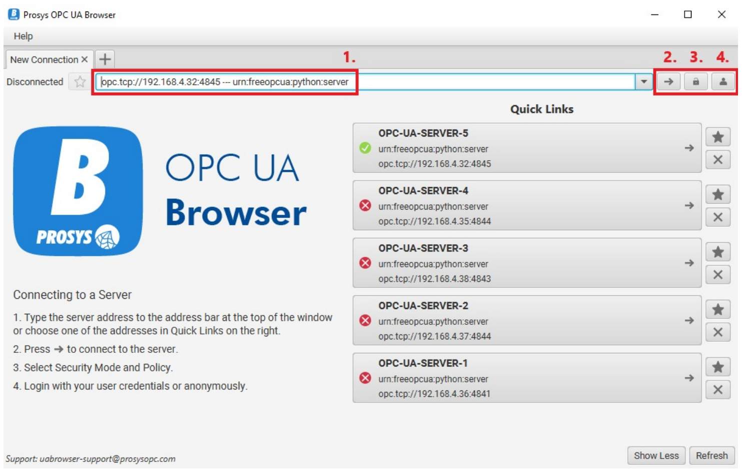
Figure 7. Configuration process of the DHT22 node.

3.2.4. Simulated Nodes

In the main function, the user was prompted to input the number of nodes that he wished to generate. The object nodes were then created and added to the server using the approach previously described. The program then iterated through each node and generated 10 individual variables acting as sensors with names randomly chosen from a list of 50 measurement units. All the created variables were then set to writable and next added to a list. Ultimately, an asynchronous function was defined to loop through each variable previously created and write random values possessing random data types.

3.3. SCADA Configuration

The Prosys OPC UA Browser for Windows-x64 was downloaded from the website (https://downloads.prosysopc.com/opc-ua-browser-downloads.php, accessed on 29 September 2022) and installed on the Windows laptop. OPC UA Browser supports multiple server connections that can be managed by means of tabbed pages. In that perspective, a connection was established to each of the OPC UA servers on the Pi, achieved by following the four steps as illustrated in Figure 8.



opc.tcp://192.168.4.32:4845 --- urn:freeopcua:python:server

Quick Links

OPC-UA-SERVER-5
urn:freeopcua:python:server
opc.tcp://192.168.4.32:4845

OPC-UA-SERVER-4
urn:freeopcua:python:server
opc.tcp://192.168.4.35:4844

OPC-UA-SERVER-3
urn:freeopcua:python:server
opc.tcp://192.168.4.38:4843

OPC-UA-SERVER-2
urn:freeopcua:python:server
opc.tcp://192.168.4.37:4844

OPC-UA-SERVER-1
urn:freeopcua:python:server
opc.tcp://192.168.4.36:4841

Connecting to a Server

1. Type the server address to the address bar at the top of the window or choose one of the addresses in Quick Links on the right.
2. Press → to connect to the server.
3. Select Security Mode and Policy.
4. Login with your user credentials or anonymously.

Support: uabrowser-support@prosysopc.com

Figure 8. OPC UA browser connection setup.

The endpoint URL of each server previously configured was individually inputted to the address bar and the connect button was pressed. The connection was established anonymously with no security features as this connection parameter was not relevant to the scope of the evaluation. Once connected to the servers, the access to the Address Space was granted and could be observed on the left pane of the application window. The Address Space contained three main folders, namely, Objects, Types and Views. The object nodes and variables created in the OPC UA server could be retrieved from the objects folder. To monitor the variable, the variable ID was right clicked and the monitor data option was selected. This created a different panel called the Data View panel enabling real time monitoring and visualisation of the data changes in the OPC UA server [61].

3.4. Data Logging and Collection

The evaluation of the standard relied on the analysis of different performance metrics. The three most important metrics were the CPU usage, amount of RAM used and the network traffic bandwidth. The metric data were obtained when increasing the number of nodes connected to the Raspberry Pi. A total of four test scenarios were conducted on each Pi as outlined in Table 2.

Table 2. Test scenarios in our experiments.

Test Scenario	No. of Connected Nodes	No. of Unique Variables and Data Values
1	1	10
2	10	100
3	20	200
4	30	300

The first two metric data were harvested every second for a period of 10 min in each of test scenarios using asynchronous functions defined in the OPC UA server code. These used the "check_output" function from the subprocess library that enabled "cat" commands to return the corresponding data [62]. The data were then converted to float values and rounded off to two decimal places before outputting them to a file containing the respective time stamp and the number of nodes tested. The network traffic was obtained using an open source command line program called Nethogs [63] which details the traffic going to and from a machine for a particular process.

4. Evaluation Results and Analysis

In this section, we present our evaluation results and analysis over a wide range of network settings. We evaluate the CPU, RAM usage and the throughput measured by the communicating Raspberry Pi nodes. These are a subset of the performance evaluation metrics commonly used when evaluating IIoT protocols, such as Robot Operating System (ROS), Data Distribution Service (DDS), Message Queuing Telemetry Transport (MQTT) and OPC UA [35].

4.1. CPU Usage of the Raspberry Pi 4B & 3B

Figure 9 presents the CPU usage of the Raspberry Pi 4B during each of the four test scenarios. From the observations made, there is a general increase of approximately 0.07% in CPU consumption between each test scenario, starting from 1.48% in scenario 1 up to 1.71% in scenario 4. This signifies that when a greater amount of data are sent from the OPC UA server to the OPC UA client, the CPU usage increases. The calculation of the maximum rise in CPU consumption reveals that there is an approximate 0.25% usage trade off when 300 data values are sent in test scenario 4 in comparison to 1 data value in test scenario 1. This can be considered as a marginal increase for the ARMx64 Quad core CPU found in the Raspberry Pi 4B.

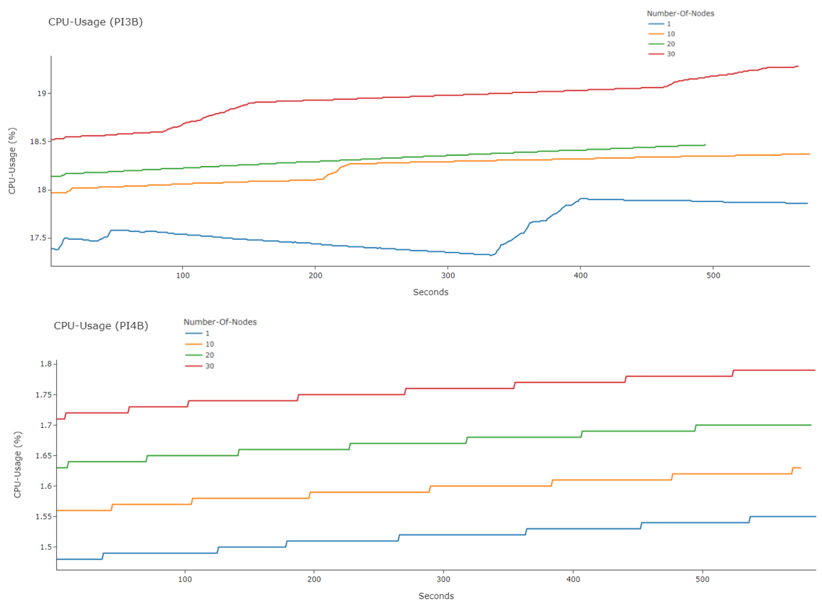


Figure 9. CPU usage on the Raspberry Pis 4B and 3B with each test scenario.

It is observed in Figure 9 that the CPU usage for the same test scenario is much greater on the Raspberry Pi 3B starting at 17.4% in test 1 up to 18.5% in test 4. In addition, the usage trade off equates to 1.42% when the number of data values sent increases from 1 to 300. This demonstrates that for a device equipped with lower specifications, OPC UA has a more prominent impact on CPU performance.

The gradual increase in CPU usage resembling a staircase-like structure on Figure 9 is suspected to be caused by a type of recursion that is likely at fault in the server code. This could potentially be asynchronous loops being created instead of a single asynchronous call.

4.2. RAM Usage of the Raspberry Pi 4B & 3B

As illustrated in Figure 10, the amount of RAM used at the start of each test, peaks to an approximate value of 10.15%, then drops back down to an average value of 9.7% until the end of the experiment. The high peaks on the graph are suspected be linked to the successful establishment of a socket connection between the OPC UA client and server which can either happen at the start of the test or during an unexpected disconnection from the server. Therefore, all the pending requests from the client and the responses from the server are buffered until a new socket is established. It can be concluded from the graph that there is minimal RAM usage trade off when 300 data values are sent in test scenario 4 in comparison to 1 data value in test scenario 1.

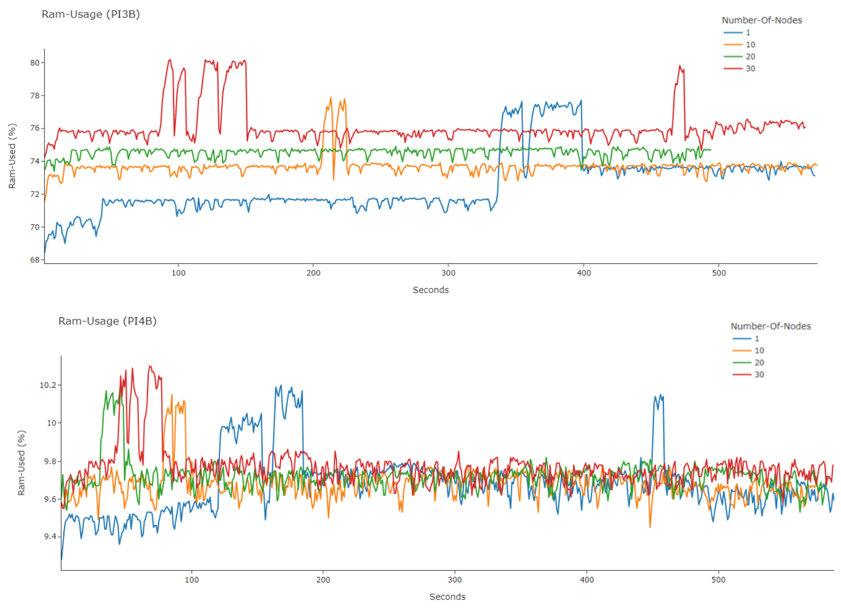


Figure 10. RAM usage on the Raspberry Pis 4B and 3B with each test scenario.

On the other hand, Figure 10 reveals that the RAM usage for the same test scenario is significantly higher on the Raspberry Pi 3B starting at 68.4% in test 1 up to 74.3% in test 4. Moreover, the RAM usage trade off when the number of data values sent, increases from 1 to 300 equates to 2.9%. This demonstrates once again that OPC UA has a substantial impact on devices equipped with lower specifications for example 1 GB of RAM on the Raspberry Pi 3B.

4.3. Network Traffic of the Raspberry Pi 4B & 3B

Figures 11 and 12 showcase the network traffic on the OPC UA Servers for each of the test scenarios. A similar trend can be observed on each graph. At the start of the experiment, the bitrate peaks to approximately 60 Kilobyte per second (Kbps) then stabilises itself until the end of the experiment. When each test scenario is started, a socket connection to the endpoint URL is attempted. A "Hello message" is sent from the OPC UA client to the server which is observed as the peaks at the start of the experiment on Figure 12. Consequently, the client requests a particular buffer size to send and retrieve data from the OPC UA server. Therefore, the OPC UA server sends an acknowledgement message to the client containing the buffer size. This can be viewed as the peaks at the start of the tests in Figure 11. From the same graphs, it can be perceived that sending 300 distinct data values to the client only takes up an average of 8.4 Kbps. This confirms that using OPC UA TCP as a transport protocol for data exchange does not congest the network.

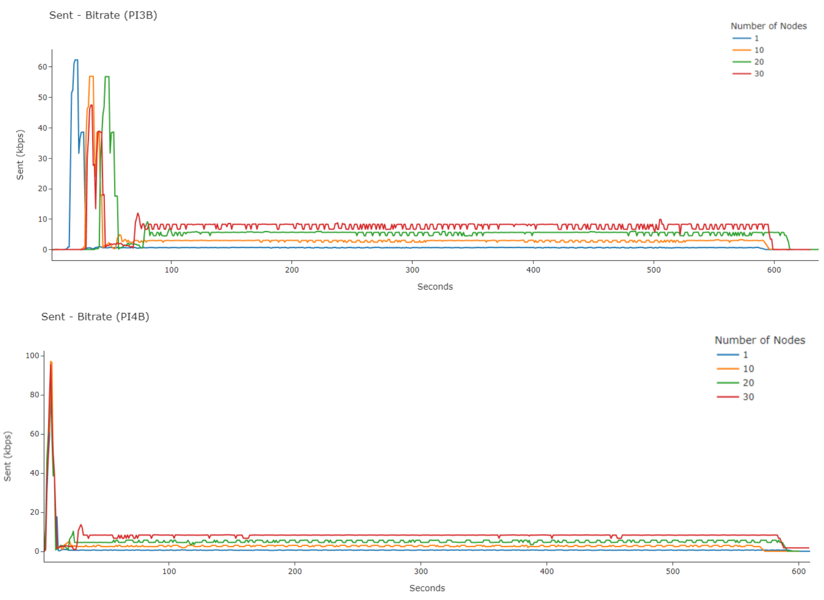


Figure 11. Sent bitrate on the Raspberry Pis 4B and 3B with each test scenario.

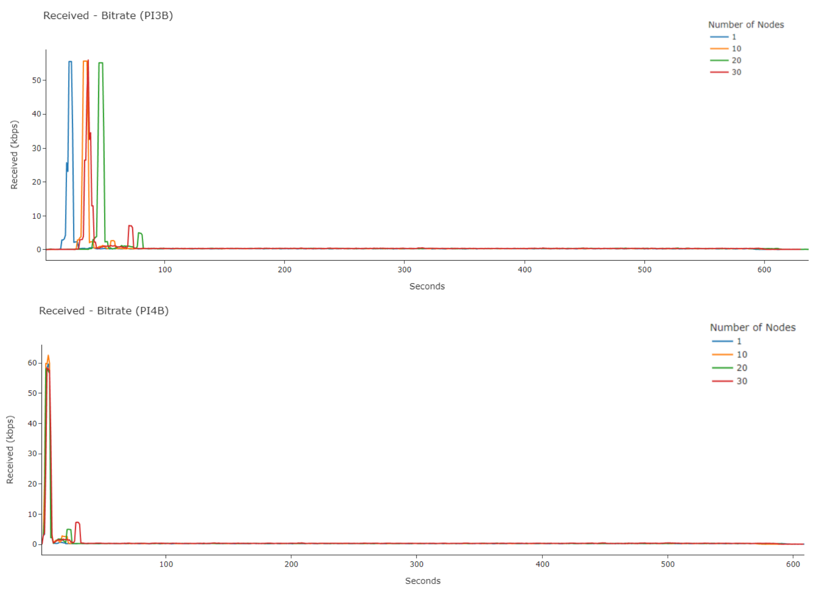


Figure 12. Received bitrate on the Raspberry Pis 4B and 3B with each test scenario.

5. Conclusions and Future Work

OPC UA can relatively resource-intensive to deploy in IIoT environments. Integrating OPC UA servers/clients in resource-constrained edge devices is an ongoing challenge. In this paper, we have proposed, designed, and implemented a system architecture for evaluating and testing the OPC UA as an industrial interoperability standard in a cost-effective environment. We demonstrated that a Raspberry Pi-based testbed can be used for cost-effective testing and evaluation of the OPC UA standard. Preliminary performance evaulation results and analysis (using standard performance metrics such as CPU, RAM, network throughput) demonstrate the feasibility and efficacy of our proposed system and experimental testbed. Our testbed can be used for small-scale protocol testing and evaluation, and also for educational purposes demonstrating how OPC UA operates and differs from other communication protocols.

Research into OPC UA as an industrial interoperability standard is advancing at a rapid pace due to increasing heterogenity of communicating nodes in IIoT environments.

Future work will incorporate real-world sensor data for OPC UA evaluation in our testbed, and developing a system for automated experiments [64,65]. Data and network traces collected from real-world industrial environments can be fed in our proposed testbed environment, and also the potential integration of network resource management techniques [66–69]. Furthermore, future work will include extending the experimental testbed to incorporate emerging wired/wireless industrial communication protocols, such as Time Sensitive Networks (TSNs) for Digital Twin Networks [70,71], Wi-Fi 6/6E [72], 5G's Ultra-reliable Low Latency Communication (URLLC) [73,74], etc. Future research will also include exploring a broader range of industrial network settings that deploy OPC UA, the interactions between OPC UA and other industrial communication protocols, and also optimising OPC UA to further support emerging/new industrial applications towards an increasingly converged ecosystem as envisioned by Industry 4.0.

Author Contributions: Conceptualization, M.L. and J.K.; methodology, M.L.; software, M.L.; validation, M.L.; formal analysis, M.L.; investigation, M.L.; resources, M.L.; data curation, M.L.; writing—original draft preparation, M.L.; writing—review and editing, M.L. and J.K.; visualization, M.L.; supervision, J.K.; project administration, M.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Boyes, H.; Hallaq, B.; Cunningham, J.; Watson, T. The industrial internet of things (IIoT): An analysis framework. *Comput. Ind.* **2018**, *101*, 1–12. [CrossRef]
2. Dai, W.; Nishi, H.; Vyatkin, V.; Huang, V.; Shi, Y.; Guan, X. Industrial edge computing: Enabling embedded intelligence. *IEEE Ind. Electron. Mag.* **2019**, *13*, 48–56. [CrossRef]
3. Sisinni, E.; Saifullah, A.; Han, S.; Jennehag, U.; Gidlund, M. Industrial Internet of Things: Challenges, Opportunities, and Directions. *IEEE Trans. Ind. Inform.* **2018**, *14*, 4724–4734. [CrossRef]
4. Lasi, H.; Fettke, P.; Kemper, H.G.; Feld, T.; Hoffmann, M. Industry 4.0. *Bus. Inf. Syst. Eng.* **2014**, *6*, 239–242. [CrossRef]
5. Ghobakhloo, M. Industry 4.0, digitization, and opportunities for sustainability. *J. Clean. Prod.* **2020**, *252*, 119869. [CrossRef]
6. Wollschlaeger, M.; Sauter, T.; Jasperneite, J. The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0. *IEEE Ind. Electron. Mag.* **2017**, *11*, 17–27. [CrossRef]
7. Marcon, P.; Zezulka, F.; Vesely, I.; Szabo, Z.; Roubal, Z.; Sajdl, O.; Gescheidtova, E.; Dohnal, P. Communication technology for industry 4.0. In Proceedings of the 2017 Progress in Electromagnetics Research Symposium, St. Petersburg, Russia, 22–25 May 2017; pp. 1694–1697.
8. Dalenogare, L.S.; Benitez, G.B.; Ayala, N.F.; Frank, A.G. The expected contribution of Industry 4.0 technologies for industrial performance. *Int. J. Prod. Econ.* **2018**, *204*, 383–394. [CrossRef]
9. Thames, L.; Schaefer, D. Industry 4.0: An Overview of Key Benefits, Technologies, and Challenges. In *Cybersecurity for Industry 4.0*; Springer: Cham, Switzerland, 2017. [CrossRef]
10. Hanstein, B. IT and IT Infrastructure in the Context of Industry 4.0. Available online: https://info.rittal.us/it-industry-4-lp (accessed on 29 September 2022).
11. Qiu, T.; Chi, J.; Zhou, X.; Ning, Z.; Atiquzzaman, M.; Wu, D.O. Edge Computing in Industrial Internet of Things: Architecture, Advances and Challenges. *IEEE Commun. Surv. Tutor.* **2020**, *22*, 2462–2488. [CrossRef]
12. Sodhro, A.H.; Pirbhulal, S.; De Albuquerque, V.H.C. Artificial intelligence-driven mechanism for edge computing-based industrial applications. *IEEE Trans. Ind. Inform.* **2019**, *15*, 4235–4243. [CrossRef]
13. Mumtaz, S.; Alsohaily, A.; Pang, Z.; Rayes, A.; Tsang, K.; Rodriguez, J. Massive Internet of Things for Industrial Applications: Addressing Wireless IIoT Connectivity Challenges and Ecosystem Fragmentation. *IEEE Ind. Electron. Mag.* **2017**, *11*, 28–33. [CrossRef]
14. Mahnke, W.; Leitner, S.; Damm, M. *OPC Unified Architecture*; Springer: Berlin/Heidelberg, Germany, 2009.
15. Tunkkari, J. Mapping Modbus to OPC Unified Architecture. Available online: https://aaltodoc.aalto.fi/bitstream/handle/123456789/30553/master_Tunkkari_Jesper_2018.pdf?isAllowed=y&sequence=1 (accessed on 29 September 2022).

16. Mantle, J. The 5 Layers of the Automation Pyramid and Manufacturing Operations Management. Available online: https://www.syspro.com/blog/erp-for-manufacturing/the-5-layers-of-the-automation-pyramid-and-manufacturing-operations-management/ (accessed on 29 September 2022).
17. Amci.com. What Is a PLC? Available online: https://www.amci.com/industrial-automation-resources/plc-automation-tutorials/what-plc/ (accessed on 29 September 2022).
18. ProcessSolutions.com. What Is Scada and How Its Used in Manufacturing? Available online: https://processsolutions.com/understanding-scada-and-what-it-can-do-for-you/ (accessed on 29 September 2022).
19. Aveva.com. What Is Human Machine Interface, or HMI? Available online: https://www.aveva.com/en/solutions/operations/hmi/ (accessed on 29 September 2022).
20. Workwisellc.com. What Is a Manufacturing Execution System (MES)? Available online: https://www.workwisellc.com/erp-software/what-is-mes/ (accessed on 29 September 2022).
21. Sap.com. What Is ERP? Available online: https://insights.sap.com/what-is-erp/ (accessed on 29 September 2022).
22. Kua, J.; Armitage, G.; Branch, P. A Survey of Rate Adaptation Techniques for Dynamic Adaptive Streaming Over HTTP. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 1842–1866. [CrossRef]
23. Kua, J.; Nguyen, S.H.; Armitage, G.; Branch, P. Using Active Queue Management to Assist IoT Application Flows in Home Broadband Networks. *IEEE Internet Things J.* **2017**, *4*, 1399–1407. [CrossRef]
24. Kua, J.; Armitage, G.; Branch, P.; But, J. Adaptive Chunklets and AQM for Higher-Performance Content Streaming. *ACM Trans. Multimedia Comput. Commun. Appl.* **2019**, *15*, 1–24. [CrossRef]
25. Thomesse, J.P. Fieldbus technology in industrial automation. *Proc. IEEE* **2005**, *93*, 1073–1101. [CrossRef]
26. MQTT Specifications. Available online: https://mqtt.org (accessed on 29 September 2022).
27. CoAP Specifications. Available online: https://datatracker.ietf.org/doc/html/rfc7252 (accessed on 29 September 2022).
28. IPv6 and 5G based Architecture for IIoT, Internet Draft (Informational). Available online: https://www.ietf.org/archive/id/draft-tang-iiot-architecture-00.html (accessed on 29 September 2022).
29. Pfrommer, J.; Ebner, A.; Ravikumar, S.; Karunakaran, B. Open source OPC UA PubSub over TSN for realtime industrial communication. In Proceedings of the 2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA), Torino, Italy, 4–7 September 2018; Volume 1, pp. 1087–1090.
30. Schweichhart, K. Why OPC UA Will Revolutionize Industrial Automation in the Coming Years. Available online: https://blog.paessler.com/why-opc-ua-will-revolutionize-industrial-automation-in-the-coming-years (accessed on 29 September 2022).
31. Imtiaz, J.; Jasperneite, J. Scalability of OPC-UA down to the chip level enables "Internet of Things". In Proceedings of the 2013 11th IEEE International Conference on Industrial Informatics (INDIN), Bochum, Germany, 29–31 July 2013; pp. 500–505.
32. Haskamp, H.; Meyer, M.; Möllmann, R.; Orth, F.; Colombo, A.W. Benchmarking of existing OPC UA implementations for Industrie 4.0-compliant digitalization solutions. In Proceedings of the 2017 IEEE 15th International Conference on Industrial Informatics (INDIN), Emden, Germany, 24–26 July 2017; pp. 589–594.
33. Gudgin, M.; Hadley, M.; Mendelsohn, N.; Moreau, J.J.; Nielsen, H.F.; Karmarkar, A.; Lafon, Y. *SOAP*; Version 1.2; World Wide Web Consortium: Cambridge, MA, USA, 2003.
34. Mumbaikar, S.; Padiya, P. Web services based on soap and rest principles. *Int. J. Sci. Res. Publ.* **2013**, *3*, 1–4.
35. Profanter, S.; Tekat, A.; Dorofeev, K.; Rickert, M.; Knoll, A. OPC UA versus ROS, DDS, and MQTT: Performance evaluation of industry 4.0 protocols. In Proceedings of the 2019 IEEE International Conference on Industrial Technology (ICIT), Melbourne, Australia, 13–15 February 2019; pp. 955–962.
36. Profanter, S.; Perzylo, A.; Rickert, M.; Knoll, A. A generic plug & produce system composed of semantic opc ua skills. *IEEE Open J. Ind. Electron. Soc.* **2021**, *2*, 128–141.
37. Madiwalar, B.; Schneider, B.; Profanter, S. Plug and Produce for Industry 4.0 using Software-defined Networking and OPC UA. In Proceedings of the 2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Zaragoza, Spain, 10–13 September 2019; pp. 126–133.
38. Cavalieri, S.; Chiacchio, F. Analysis of OPC UA performances. *Comput. Stand. Interfaces* **2013**, *36*, 165–177. [CrossRef]
39. Post, O.; Seppälä, J.; Koivisto, H. The Performance of OPC-UA Security Model at Field Device Level. In Proceedings of the 6th International Conference on Informatics in Control, Automation and Robotics, Volume Robotics and Automation, Milan, Italy, 2–5 July 2009; Volume 2, pp. 337–341.
40. Braune, A.; Hennig, S.; Hegler, S. Evaluation of OPC UA secure communication in web browser applications. In Proceedings of the 2008 6th IEEE International Conference on Industrial Informatics, Daejeon, Korea, 13–16 July 2008; pp. 1660–1665. [CrossRef]
41. Rocha, M.; Sestito, G.; Dias, A.; Turcato, A.; Brandao, D. Performance Comparison Between OPC UA and MQTT for Data Exchange. In Proceedings of the 2018 Workshop on Metrology for Industry 4.0 and IoT, Brescia, Italy, 16–18 April 2018; pp. 175–179. [CrossRef]
42. Fangjian, L.; Yanlin, Z.; Zhen, W. Research and application of OPC UA server based on resource constrained platform STM32. In Proceedings of the 2021 IEEE 3rd International Conference on Civil Aviation Safety and Information Technology (ICCASIT), Changsha, China, 20–22 October 2021; pp. 961–964.

43. Cho, H.; Jeong, J. Implementation and performance analysis of power and cost-reduced OPC UA gateway for industrial IoT platforms. In Proceedings of the 2018 28th International Telecommunication Networks and Applications Conference (ITNAC), Sydney, Australia, 21–23 November 2018; pp. 1–3.
44. González Vázquez, F. Test Platform for the Performance Evaluation of OPC-UA Servers for Fast Data Transfer between Intelligent Equipment. In Proceedings of the ThinkMind—INTELLI 2015, the Fourth International Conference on Intelligent Systems and Applications, St. Julians, Malta, 11–16 October 2015; pp. 179–182.
45. Okuda, M.; Mizuya, T.; Nagao, T. Development of IoT testbed using OPC UA and database on cloud. In Proceedings of the 2017 56th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE), Kanazawa, Japan, 19–22 September 2017; pp. 607–610. [CrossRef]
46. Silva, D.; Carvalho, L.I.; Soares, J.; Sofia, R.C. A Performance Analysis of Internet of Things Networking Protocols: Evaluating MQTT, CoAP, OPC UA. *Appl. Sci.* **2021**, *11*, 4879. [CrossRef]
47. Jan, O.R.; Jo, H.S.; Jo, R.S.; Kua, J. Real-Time Flood Monitoring with Computer Vision through Edge Computing-Based Internet of Things. *Future Internet* **2022**, *14*, 308. [CrossRef]
48. Olorunnife, K.; Lee, K.; Kua, J. Automatic Failure Recovery for Container-Based IoT Edge Applications. *Electronics* **2021**, *10*, 3047. [CrossRef]
49. Welgama, H.; Lee, K.; Kua, J. A Framework for Seamless Offloading in IoT Applications using Edge and Cloud Computing. In Proceedings of the IoTBDS Internet of Things, Big Data and Security, International Conference, Prague, Czech Republic, 22–24 April 2022.
50. PROSYS. Prosys OPC UA Simulation Server—Prosys OPC. Available online: https://www.prosysopc.com/products/opc-ua-simulation-server/ (accessed on 29 September 2022).
51. Umati. GitHub—Umati/Sample-Server. Available online: https://github.com/umati/Sample-Server (accessed on 29 September 2022).
52. open62541. open62541. Available online: https://www.open62541.org/ (accessed on 29 September 2022).
53. Rossignon, E. Node-OPCUA. Available online: https://node-opcua.github.io/ (accessed on 29 September 2022).
54. FreeOpcUa. Python-Opcua: LGPL Pure Python OPC-UA Client and Server. Available online: https://github.com/FreeOpcUa/python-opcua (accessed on 29 September 2022).
55. FreeOpcUaGroup. OPCUA-Asyncio:OPC UA Library for Python >= 3.7. Available online: https://github.com/FreeOpcUa/opcua-asyncio (accessed on 29 September 2022).
56. PythonSoftwareFoundation. Asyncio—Asynchronous I/O—Python 3.10.5 Documentation. Available online: https://docs.python.org/3/library/asyncio.html#module-asyncio (accessed on 29 September 2022).
57. Littlebirdelectronics. DHT22 Temperature-Humidity Sensor. Available online: https://littlebirdelectronics.com.au/products/dht22-temperature-humidity-sensor-extras (accessed on 29 September 2022).
58. Cacamille3. OPC-UA-Clients: List of Commercial and Open Source OPC UA Clients. Available online: https://github.com/cacamille3/OPC-UA-Clients (accessed on 29 September 2022).
59. Prosysopcltd. Prosys OPC UA Browser—Prosys OPC. Available online: https://www.prosysopc.com/products/opc-ua-browser/ (accessed on 29 September 2022).
60. OPCFoundation. OPC Factory Server. Available online: https://opcfoundation.org/developer-tools/specifications-unified-architecture (accessed on 29 September 2022).
61. Prosysopc. Prosys OPC UA Browser UserManual. Available online: https://downloads.prosysopc.com/opcua/apps/UaBrowser/dist/4.2.0-33/Prosys_OPC_UA_Browser_UserManual.pdf (accessed on 29 September 2022).
62. Python Software Foundation. Subprocess—Subprocess Management. Available online: https://docs.python.org/3/library/subprocess.html#subprocess.check_output (accessed on 29 September 2022).
63. Raboof. Nethogs: Linux 'Net Top' Tool. Available online: https://github.com/raboof/nethogs (accessed on 29 September 2022).
64. Kua, J.; Al-Saadi, R.; Armitage, G. Using Dummynet AQM-FreeBSD's CoDel, PIE, FQ-CoDel and FQ-PIE with TEACUP v1. 0 testbed. *CAIA Tech. Rep.* **2016**, *160708*, 8.
65. Kua, J.; Armitage, G. Generating Dynamic Adaptive Streaming over HTTP Traffic Flows with TEACUP Testbed. *CAIA Tech. Rep.* **2016**, *161216*, 16.
66. Kua, J.; Armitage, G.; Branch, P. The Impact of Active Queue Management on DASH-Based Content Delivery. In Proceedings of the 2016 IEEE 41st Conference on Local Computer Networks (LCN), Dubai, United Arab Emirates, 7–10 November 2016; pp. 121–128. [CrossRef]
67. Kua, J.; Armitage, G. Optimising DASH over AQM-Enabled Gateways Using Intra-Chunk Parallel Retrieval (Chunklets). In Proceedings of the 2017 26th International Conference on Computer Communication and Networks (ICCCN), Vancouver, BC, Canada, 31 July–3 August 2017; pp. 1–9. [CrossRef]
68. Kua, J.; Branch, P.; Armitage, G. Detecting Bottleneck Use of PIE or FQ-CoDel Active Queue Management During DASH-like Content Streaming. In Proceedings of the 2020 IEEE 45th Conference on Local Computer Networks (LCN), Sydney, Australia, 16–19 November 2020; pp. 445–448. [CrossRef]
69. Kua, J. Understanding the Achieved Rate Multiplication Effect in FlowQueue-based AQM Bottleneck. In Proceedings of the 2021 IEEE 46th Conference on Local Computer Networks (LCN), Edmonton, AB, Canada, 4–7 October 2021; pp. 439–442.
70. Wu, Y.; Zhang, K.; Zhang, Y. Digital twin networks: A survey. *IEEE Internet Things J.* **2021**, *8*, 13789–13804. [CrossRef]

71. Wang, S.; Kua, J.; Jin, J.; Kulkarni, A.; Jayaraman, P.P.; Cao, X. Optimal graph partitioning for time-sensitive flow scheduling towards digital twin networks. In Proceedings of the 1st Workshop on Digital Twin & Edge AI for Industrial IoT, Sydney, Australia, 17 October 2022; pp. 7–12.
72. Chung, M.A.; Chang, W.H. Low-cost, low-profile and miniaturized single-plane antenna design for an Internet of Thing device applications operating in 5G, 4G, V2X, DSRC, WiFi 6 band, WLAN, and WiMAX communication systems. *Microw. Opt. Technol. Lett.* **2020**, *62*, 1765–1773. [CrossRef]
73. Popovski, P.; Trillingsgaard, K.F.; Simeone, O.; Durisi, G. 5G wireless network slicing for eMBB, URLLC, and mMTC: A communication-theoretic view. *IEEE Access* **2018**, *6*, 55765–55779. [CrossRef]
74. Kua, J.; Loke, S.W.; Arora, C.; Fernando, N.; Ranaweera, C. Internet of things in space: A review of opportunities and challenges from satellite-aided computing to digitally-enhanced space living. *Sensors* **2021**, *21*, 8117. [CrossRef]