UNIVERSITY OF TURKU

Performance Evaluation of Polling-Based OPC UA Client-Server Communication in Industrial PLC Systems

An Experimental Load Study on OPC UA Performance Using Siemens S7-1500 PLC

Faculty of Technology
Master's thesis

Author:
Reko Vuorinen

9.4.2026

Turku

Master's thesis

Subject: Mechanical Engineering
Author: Reko Vuorinen
Title: Performance Evaluation of Polling-Based OPC UA Client-Server Communication in Industrial PLC Systems
Supervisor(s): University Lecturer Jani Heikkinen
Number of pages: 74 pages
Date: 9.4.2026

As the adoption of Industrial Internet of Things (IIoT) technologies is becoming more common, it has increased the use of standardized communication protocols for integrating industrial devices and information system in a production environment. Open Platform Communications Unified Architecture (OPC UA) is widely used technology for making interoperable communication possible in industrial plants. However, when OPC UA experiences increased communication load, the OPC UA client-server solutions may show performance limitations which makes the system less reliable.

This thesis investigates the performance behaviour of polling-based OPC UA client-server communication in a pharmaceutical production environment. Thesis utilizes programmable logic controller (PLC) as the OPC UA client. The study is conducted in collaboration with a pharmaceutical manufacturing company. In their production, communication anomalies had been observed under increased load conditions. To research this a quantitative research method was used in order to evaluate the effects of OPC UA on given communication performance. This was done by varying the number of nodes per read request, payload size, polling interval, and parallel read jobs in given communication setup.

The results of the study show that the number of nodes per request has the greatest influence on client-side processing delay. In the used testbed setup, this leads to increased polling drift under high communication loads. The amount of total communication data which is determined by payload size and number of nodes together, was found to have the most affect for round-trip (RTT) time and jitter measured in given setup. Parallel read jobs had only a small direct impact on performance within the given hardware limits. Observed anomalies coming from duplicates and missed values, were believed to be primarily caused by timing mismatches between client polling cycles and server update cycles.

Overall, this study gives insight of the performance characteristics of polling-based OPC UA communication. It introduces a testbed that enables systematic analysing of the communication and optimization of polling configuration timings. OPC UA subscription model is also shown as a potentially more robust event-based alternative for the studied industrial use case.

Key words: OPC UA, IIoT, performance evaluation, client-server communication, industrial communication, polling-based communication.

Generative AI is used to improve grammar and clarity of the thesis.

Diplomityö

Oppiaine: Konetekniikka
Tekijä(t): Reko Vuorinen
Otsikko: Lukupyyntö-pohjaisen OPC UA asiaka-palvelin kommunikaation suorityskyvyn testaus teollisissa PLC-järjestelmissä
Ohjaaja(t): Yliopistonlehtori Jani Heikkinen
Sivumäärä: 74 sivua
Päivämäärä: 9.4.2026

Teollisen internetin (Industrial Internet of Things, IIoT) teknologioiden yleistyminen on lisännyt standardoitujen tiedonsiirtoprotokollien käyttöä teollisissa laitteissa ja järjestelmissä. Open Platform Communications Unified Architecture (OPC UA) on yksi yleisimmin käytetyistä teknologioista yhtenäistetyn tiedonsiirron mahdollistamiseksi teollisissa ympäristöissä. Suurilla tiedonsiirtomäärillä, OPC UA asiakas-palvelin kommunikaatiossa on havaittu esiintyvän rajoitteita ja vikakäyttäytymistä, jotka vaikuttavat järjestelmän luotettavuuteen.

Tässä työssä tutkitaan OPC-asiakkaan lukupyyntöihin pohjautuvaan OPC UA asiakas-palvelin kommunikaation suorituskykyä teollisessa ympäristössä hyödyntäen ohjelmoitavaa logiikkaa (PLC) OPC UA asiakkaana. Tutkimus toteutettiin yhteistyössä lääketeollisuudessa toimivan tuotantoyrityksen kanssa, jonka tiedonsiirrossa oli havaittu poikkeamia suuren tiedonsiirtomäärien yhteydessä. Tutkimuksessa hyödynnettiin kvantitatiivista kokeellista tutkimusasetelmaa, jossa kommunikaation suorituskykyä arvioitiin muuttamalla järjestelmällisesti seuraavia parametreja: lukupyynnön elementtien määrää, viestin kokoa (payload), lukupyyntöjen pyyntötiheyttä sekä rinnakkaisten lukupyyntöjen määrää.

Tulokset osoittavat, että lukupyynnön elementtien määrä vaikuttaa merkittävimmin OPC-asiakkaan käsittelyviiveeseen, mikä johtaa lukyöyyntöjen viivästymiseen suurilla tiedonsiirtokuormilla testatussa järjestelmässä. Kokonaiskommunikaatiokuorman havaittiin vaikuttavan merkittävästi tiedonsiirron edestakaiseen vasteaikaan (round-trip time, RTT) ja sen vaihteluun (jitter). Tämä kokonaiskuorma määräytyy viestin koon ja elementtien yhteenlasketun kuorman perusteella. Rinnakkaisilla lukupyynnöillä oli laitevalmistajan asettamien rajoitusten puitteissa vain vähäinen vaikutus kommunikaation viiveeseen ja suorituskykyyn. Lisäksi havaitut poikkeamat, kuten duplikaatit ja puuttuvat arvot, todettiin johtuivat pääasiassa ajallisista epäsynkronoinneista OPC-asiakkaan lukupyyntösyklien ja OPC-palvelimen päivityssyklien välillä.

Tutkimus tuottaa empiiristä tietoa teollisessa käytössä olevan OPC UA -kommunikaation suorituskykyominaisuuksista ja esittelee testausympäristön, joka mahdollistaa kommunikaation systemaattisen analysoinnin sekä mahdollistaa lukupyyntöjen optimoinnin. Tulokset korostavat myös OPC UA subscription mallin potentiaalia luotettavampana vaihtoehtona teolliseen kommunikaatioon tarkastellussa käyttötapauksessa.

Avainsanat: OPC UA, teollinen internet, suorituskyvyn mittaus, asiakas-palvelin-kommunikaatio, teollinen tiedonsiirto, kyselypohjainen kommunikaatio.

Generatiivista tekoälyä käytettiin parantamaan työn kielioppia ja selkeyttä.

Preface

This thesis was carried out in collaboration with the partner company, whose support and expertise made the completion of this work possible. First, I would like to thank my university supervisor, Jani Heikkinen, for his valuable feedback, and support throughout the thesis process. I would also like to thank the supervisors from the collaboration company, Niko Laihinen and Juha Huttunen, for giving the thesis topic and providing valuable advice, technical insights, and support during this work. I want to also thank the entire OT team in the partner company. Their efforts in building and maintaining the testbed environments made the experimental work possible and they helped throughout this thesis. Finally, I would like to thank Heikki Huttunen from the partner company for providing the opportunity to carry out this thesis.

Table of contents

List of Abbreviations

IIoT	Industrial Internet of Things
IoT	Internet of Things
OPC UA	Open Platform Communications Unified Architecture
PLC	Programmable Logic Controller
RTT	Round-Trip Time
PubSub	Publish-Subscribe
CPS	Cyber-Physical Systems
IT	Information Technology
OT	Operational Technology
SCADA	Supervisory Control and Data Acquisition
HMI	Human-Machine Interface
MES	Manufacturing Execution System
ERP	Enterprise Resource Planning
OPC	Open Platform Communications
RAMI 4.0	Reference Model Industrie 4.0
SOA	Service-Oriented Architecture
UNS	Unified Namespace
MQTT	Message Queuing Telemetry Transport
M2M	Machine-to-Machine
OSI	Open System Interconnection
ISO	International Standards Organization
TCP/IP	Transmission Control Protocol / Internet Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
IP	Internet Protocol
IPv4	IP version 4
IPv6	IP version 6
MAC	Median Access Control
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
RTE	Real-Time Ethernet
OLE	Object Linkin and Embedding
COM	Component Object Model
DCOM	Distributed Component Object Model
UADP	UA Datagram Protocol

FLC	Field Level Communications
UAFX	Field eXchange
TSN	Time-Sensitive Networking
PTP	Precision Time Protocol
NTP	Network Time Protocol
QoS	Quality of Service
AMQP	Advanced Message Queuing Protocol
TTC	Time-to-Completion
TIA	Totally Integrated Automation
NGIL	NamespaceGetIndexList
NGHL	NodeGetHandleList
IQR	Interquartile Range

1 Introduction

The Industrial Internet of Things (IIoT) systems are often used to connect smart industrial devices for example, sensors, controllers, and other machines with centralized platforms. This enables that data can be gathered in real time helping with analysis. With IIoT the ability for better decision making and transparency in production can be increased. Due to IIoT operating in industrial environment there are specific requirements related to its performance. Industrial processes often have machines and other assets which are valuable and often critical operations where system malfunctions are likely to cause significant safety hazards and economic losses. The malfunctions can also result in unreliable data. Therefore, IIoT systems must provide more robustness than regular Internet of Things (IoT) applications [1], [2]. On top of this, IIoT systems often also gather large amount of data using multiple sensors and other measurement systems which continuously monitor and control the production processes [2].

As the adoption of IIoT increases in industrial processes, interoperability has become one of the primary challenges related to it. Machines and sensors are often from different vendors and operate over various communication protocols and models. Resulting in fragmented systems that cannot easily exchange information [2]. This has raised the need for standardized communication framework allowing easy integration of these fragmented systems. One of the key technologies in this area is Open Platform Communications Unified Architecture (OPC UA), which is defined by the IEC 62541 standard [2].

OPC UA makes communication possible in networks using devices from different vendors, which helps creating interoperable IIoT systems. However, many industrial devices such as programmable logic controllers (PLCs) operate with limited computational resources. Using the relatively complex OPC UA as a communication method on those devices may lead to decrease in communication performance. Possibly affecting communication latency and therefore also real-time responsiveness of the system [2]. This can further cause unreliable data, which is considered problematic in many industrial sectors. Therefore, research about OPC UA performance in IIoT systems is needed tackle the previously stated challenges.

1.1 Problem Context

The thesis is conducted in collaboration with a pharmaceutical manufacturing company that utilizes OPC UA client-server communication in several of its production systems. Under increased load,

anomalies have been observed in their communication data. This raises concerns about data reliability and potential system malfunctions.

To identify the root causes of these anomalies the performance of polling-based OPC UA client-server communication has been chosen to be evaluated. This analysing is trying to represent the production environment in use, therefore the following research questions were chosen with RQ1 being the overall goal and others specifying it:

RQ1: How does the performance of OPC UA client-server communication behave under increased load in an industrial PLC environment?
RQ2: How does the number nodes per OPC UA Read request affect measured performance metrics?
RQ3: How does the polling interval influence the performance of OPC UA communication?
RQ4: What is the impact of parallel OPC UA Read requests on reliability and performance of the client?
RQ5: How do different payload sizes influence the OPC UA performance?
RQ6: Do communication anomalies occur when OPC UA communication experiences large data loads?

1.2 Research Approach

The study is conducted using quantitative research method with experimental procedures. The method is used to evaluate the performance and behaviour of a polling-based OPC UA communication configuration under varied load conditions. Using a quantitative approach enables the measurement and analysis of OPC UA performance characteristics with actual measured values within a built testbed setup.

One expected outcome of the study is to reproduce of the anomalies observed in the industrial context case. Using the quantitative data, the study tries to determine whether these issues originate from limitations in the OPC UA communication itself, from client logic implementation with possible timing mismatches, or from some other factors within the communication stack. Quantitative data is therefore gathered to analyse how data transfer with chosen metrics under varying load conditions.

With the available resources from the collaborating company, the testbed was built to replicate the essential characteristics of the production architecture. This was done while allowing controlled manipulation of the affecting functions and load parameters. By systematically varying one of these parameters at a time, the influence of each function specified in research questions can be separately studied.

The testbed configuration is well documented and explained in later sections to ensure repeatability. Open-source tools are also used where possible to make the testbed easy to reproduce, this is with exception of the hardware PLC and software.

With the investigation of the backup-file of the production system and the planning for the testbed, few working assumptions were created to verify the testbed setup against. One of these is that duplicate values observed in the SCADA system are not necessarily communication faults caused by packet loss as first thought. But may instead result from the timing behaviour of the polling. If the smart sensor updates its values at a slower rate than the PLC polls them, the PLC may read identical values across the same update cycle of the sensor, resulting in duplicate reads. Because the smart sensor update cycle and the PLC polling cycle operate independently from each other, anomaly free communication in polling configuration would either need buffers which are not allowed in this setup according to the collaborating company or careful synchronization of the timings. Even with timing alignment under increased client load, PLC cycle latency and jitter may increase, potentially still causing misalignment between update and polling cycles.

While a slightly faster PLC polling compared to the smart sensor update cycle may also produce duplicate values. It would most logically be the cause of misconfiguration of the PLC polling timer. Which was not mentioned by the collaborating company, under higher communication load, increased client latency and jitter may cause client polling timing shift so that certain sensor updates are not observed at all. Which would lead to missed values on the historian. Since the historian archives only the values retrieved by the PLC, without additional features such as sequence number, missed updates are not visible in archived data. This is yet to confirm whether missed smart sensor values occur in production.

The working assumption for the occasional invalid values remains difficult to explain at this stage. The specific anomaly will be further discussed based on the results. These findings are then further discussed in the Discussion chapter.

1.3 Scope and Delimitations

As mentioned, the scope of this thesis is to evaluate the performance of polling-based OPC UA client-server communication in an industrial PLC environment. The study focuses on measuring how different communication parameters influence system performance under varying load conditions. The study is conducted within a specific industrial production environment in collaboration with a pharmaceutical manufacturing company. Therefore, the experiments are made to reflect similar operational conditions to the company's production systems.

Several delimitations are applied in order to keep the scope of the study focused in the practical requirements required as an output of this study. First, the study itself focuses only on polling-based OPC UA client-server communication. Meaning that for example that the OPC UA client-server subscription model is not included. This decision was made because the systems used in the production environment rely primarily on polling-based communication.

Second, the research focuses on the performance of a single vendor PLC platform. As a result, the findings reflect the behaviour of OPC UA communication within this specific hardware and software environment. Performance comparisons between different PLC vendors or software platforms are therefore outside of the scope of this study.

Third, other OPC UA communication models, such as OPC UA Publish-Subscribe (PubSub) and other alternative industrial communication protocols, are not included in the study. These communication architectures are not currently causing anomalies in the production system and therefore fall outside of the scope of the study. However, some of them are introduced and discussed at a theoretical level in the literature review. This is to provide context to understand the used communication system and to present alternative communication approaches also used in industrial communication.

1.4 Structure of the Thesis

The thesis starts with and introduction to the thesis. Following with the research problem, its industrial context forming the research questions and the chosen research method. Then the scope and delimitations of the study are presented. The theoretical foundation of the work is shown after this with background of industrial automation architectures and then analysing relevant literature found related to OPC UA performance testing.

Following, the thesis presents the fundamental concepts of industrial communication. Including basics of networking focusing on the used communication protocol, the evolution of industrial communication technologies leading to OPC UA, and lastly explaining the architecture of OPC UA. This is followed by introduction to different OPC UA communication models to present other possibilities to the used OPC UA method. Also, alternative industrial communication frameworks are introduced. Lastly the relevant performance metrics are presented, which some are later used to evaluate communication performance and behaviour of the OPC UA communication.

The methodology part of the thesis describes the conducted experimental study. This includes the research approach, the industrial case context, and the architecture of the developed testbed for the experiments. The implementation of how the metrics are gathered from the three separate parts of the

system are shown. The system is separated into three parts including OPC UA server, PLC client, and the network itself. Later the experimental procedures describing the full systematic method of conducting the experiments are explained in detail.

Finally, the results are presented. This is followed by a discussion of the results. Both are done in relation to the research questions and findings from previous studies are also taken into an account. The thesis concludes by explaining the main conclusions of the work and showing the potential directions for future research.

2 Background

The purpose of this section is to provide background information to help the reader understand different industrial communication architectures. First, the foundational layers of common industrial automation systems are introduced and explained. This is followed by an overview of more modern industrial communication architectures, providing perspective on the current state-of-the-art in industrial automation. Also, it is explained how communication topologies could be modified to even improve the whole automation systems performance and scalability.

2.1 Layers of Industrial Automation

Industrial systems have become more connected through IIoT technologies. To better understand how the modern industrial automation communication works, it is worth discussing the basics of its different levels, and their hierarchical structure. The main ideology in Industry 4.0 is often referred as smart manufacturing [3]. Smart manufacturing and also smart factories utilize cyber-physical systems (CPS) where everything is digitally connected. This is primarily done as a way of gaining productivity and modernizing manufacturing [3]. This structure using CPS in complex systems of modern smart factories are usually built on two components, the upper information technology (IT) and lower operational technology (OT) [4]. IT is often considered as the cyber domain of the structure which includes the software side of the modern factories in industry 4.0, while OT forms the physical domain within the factory field level including machines, actuators and sensors for example [4], [5].

Next the foundational reference architectures behind the smart manufacturing is presented. This is done to better understand the different levels of automation systems in smart manufacturing. To be noted, the following architecture is more for purpose of understanding the foundations and working principles of industrial communication. Other more evolved architectures are also in use, from which some are presented later in this section.

2.1.1 ISA-95 Automation Pyramid

ANSI/ISA 95 standard defines the so called automation pyramid, this represents the most known reference architecture for smart manufacturing, providing clear hierarchical approach to automation [3], [6]. The automation pyramid separates the system into five separate layers, assigning clear tasks and communication methods between each layer [3], [7]. Illustration of an automation pyramid with discussed IT/OT separation is presented in Figure 1.

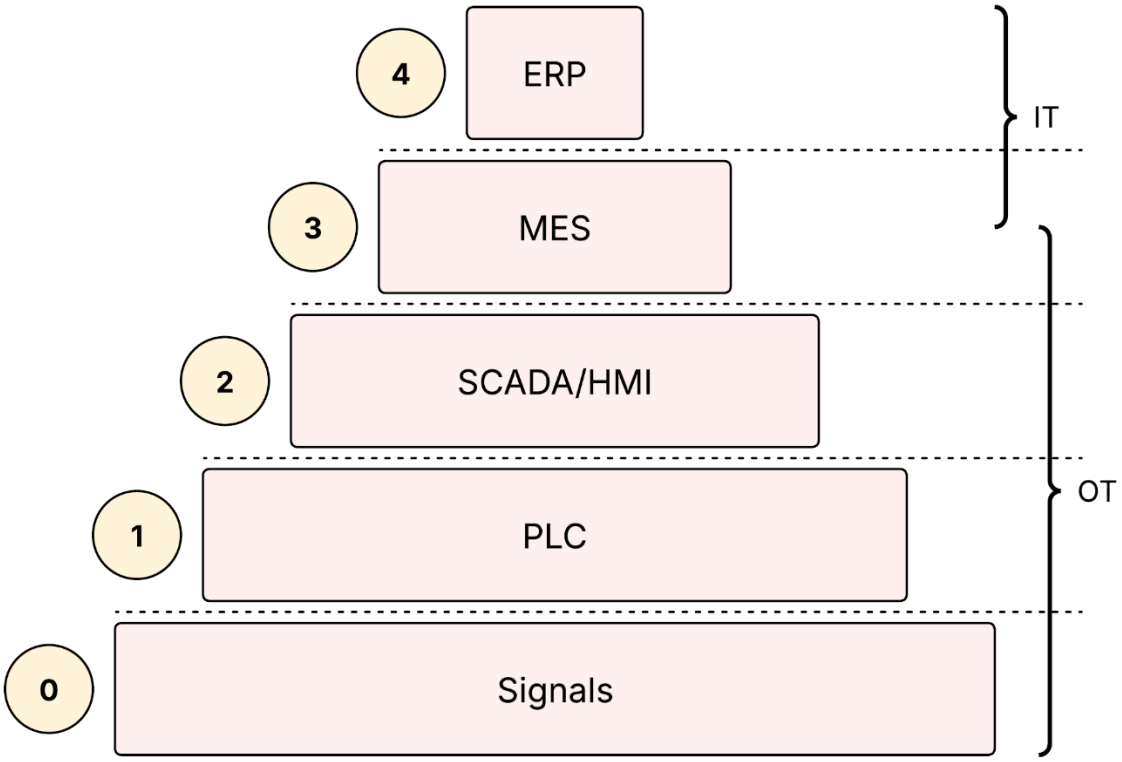

Figure 1. ISA-95 automation pyramid with illustration of the IT/OT gap.

Each of these levels (0-4) represents a specific function within the production and management hierarchy, going from production process control on the shop floor to the enterprise business planning systems [3], [4], [7].

Level 0 represents the actual production itself where IIoT devices interact directly with manufacturing equipment. These devices collect real-time data such as position of actuators, pressure of the process, or temperature from sensors. This typically happens at millisecond intervals. Actuators respond to control signals to perform different actions. These include for example opening valves, adjusting speeds, or moving other mechanical components. The 0 level in the automation pyramid forms the foundation for industrial automation by linking the physical environment to the digital control system above them. [3], [4]

Level 1 consists of the control system coordinating the real-time process management. The key component at this level are PLCs. PLC typically receive data from sensors, processes it according to programmed logic, and then send commands to actuators to maintain desired operational state. PLCs are designed to operate under strict real-time constraints to ensure process safety and reliability. [3], [7]

Level 2 known as the supervisory layer. This layer has two main systems. Supervisory Control and Data Acquisition (SCADA) and Human-Machine Interfaces (HMIs). SCADA oversees and coordinates multiple PLCs. It collects and aggregates data from the control systems to provide operators with real-time overview of production. While SCADA systems enable remote monitoring and control, it can also manage alarms of the system and even record performance data. HMIs on the other hand mainly visualize system states and allow operators interact with the equipment. [3], [4]

The first three layers (0-2) together form the OT domain. These layers are directly involved with the physical processes of manufacturing and often operate under real-time conditions. The focus of these OT systems is the reliable control and data acquisition of machinery and processes. [4]

Level 3, often called as manufacturing operations and management bridges the gap between OT and IT domains [4]. The core system in this level is Manufacturing Execution System (MES). The MES transforms production orders from higher-level enterprise systems into executable instructions to the shop floor. It coordinates scheduling, resource allocation, and production tracking. Making sure that operations align with strategic objectives. MES level also combines performance data from lower levels and provides feedback for business analysis to the upper level. [3], [7]

Level 4 is the enterprise and business management layer. It is the only layer which is entirely within the IT domain. It is typically centred on Enterprise Resource Planning (ERP) systems. These handle long-term planning, logistics, procurement, finance, and supply chain coordination. ERP systems process aggregated data from the MES to support decision-making. This is while business plans, orders, and production targets flow downward toward operational layers. These systems often run on standard IT infrastructure and often use IP-based communication technologies. [3], [4], [7]

The ISA-95 automation pyramid provides a structured view of how industrial systems are organized. This includes all the different levels from floor, where the production happens to the enterprise systems managing the whole smart factory and its stakeholders [3], [7]. In general information typically flows upward from sensors and controllers to management systems for monitoring and optimization, while production plans flow downward [3], [7]. The separation between OT and IT domains makes clear functional separation while also introducing communication challenge [4]. In practice, the communication between supervisory systems such as SCADA and higher-level applications like MES and ERP is often done using Open Platform Communications (OPC) servers providing standardized interface for data exchange across different systems and vendors [3], [8]. As industries increasingly move toward fragmented IIoT environments, bridging these domains between OT and IT has become important.

2.1.2 Modern Industrial Architecture Frameworks

While ISA-95 automation hierarchy gives good foundation for understanding modern industrial automation. Newer frameworks have been developed to address the demands of Industry 4.0 and the IIoT. For the scope of the thesis, these models are presented to give view of the trends in industry and alternatives for the more traditional system architectures. One example of these newer models is Reference Architectural Model Industrie 4.0 (RAMI 4.0), which was designed specifically to guide the new era of connected and data-driven production [3].

RAMI 4.0 introduces a Service-Oriented Architecture (SOA) approach which makes possible more modular, scalable and interoperable automation systems [9]. The basic structure of RAMI 4.0 is similar to ISA-95, but the main point in RAMI 4.0 is that skipping layers is made possible in order to provide flexibility in the system [3]. Rather than a pyramid structure which is in ISA-95 automation pyramid, RAMI 4.0 is typically presented as a three-dimensional cube. These three axes of RAMI 4.0 model is shown in Figure 2 which illustrates typical structure of RAMI 4.0.

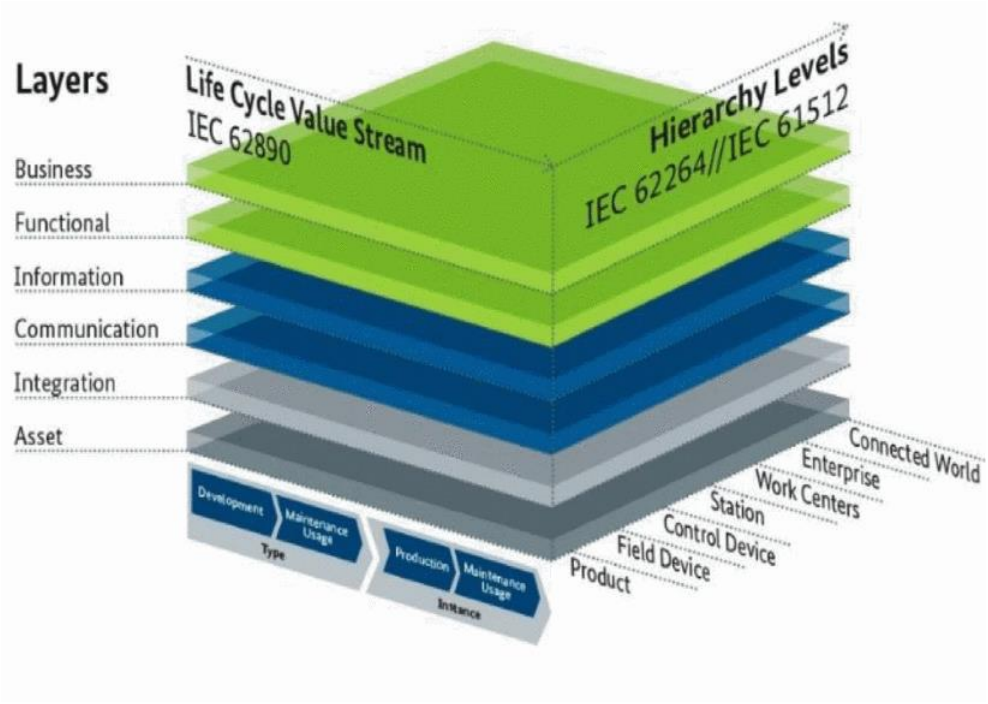

Figure 2. RAMI 4.0 architecture where the three-dimensional cube structure is illustrated. [3]

The three axes of RAMI 4.0 introduced briefly:

1. Hierarchy Levels: represents the vertical integration of the different systems in factory. Extending the ISA-95 concept so that every part of the production system is connected to each other. This allows more flexible communication between devices. [3], [9]

2. Life Cycle Value Stream: describes all different stages of an asset's existence. This ranges from initial design and development to operation, maintenance and decommissioning. [3], [9]
3. Architecture layers: is the primary factor in defining interoperability layers. These layers clarify each component's role and abilities. The third axis also defines clear inputs and outputs for each layer. This feature makes changes easier to implement on a specific layer therefore increasing interoperability. [3], [9]

In many industrial environments, system integration still relies mostly on numerous end-to-end connections between components. Even when factories adopt standardized communications protocols such as OPC UA, each subsystem typically requires its own set of custom interfaces to exchange data with other component. Making it so that every new functionality has to be integrated one way or another into the existing architecture. Possibly introducing additional interface development new or even new maintenance plans. Ultimately increasing system complexity and lifecycle cost. This challenge mentioned has created alternative integration approaches, Unified Namespace (UNS) being one of these. [3]

Different approach to end-to-end integrations used in RAMI4.0, where each system requires individual interfaces to exchange data. The UNS makes so that all information and data is stored in a single data structure that serves as the real-time "*single source of truth*" [10] for operational and business systems. In UNS, every change observed in the production environment is treated as an event. These changes can be for example counter increment, machine status update, or temperature variation in cooling system. Each device that produces data publishes its to specific event broker, while any system or application interested in those events can subscribe to the broker and receive updates when they occur. The function of UNS event-based architecture compared to other end-to-end architectures such as RAMI4.0 is illustrated in Figure 3. The key enabler for this kind of event-based architecture is the PubSub communication model presented later in this thesis. [3]

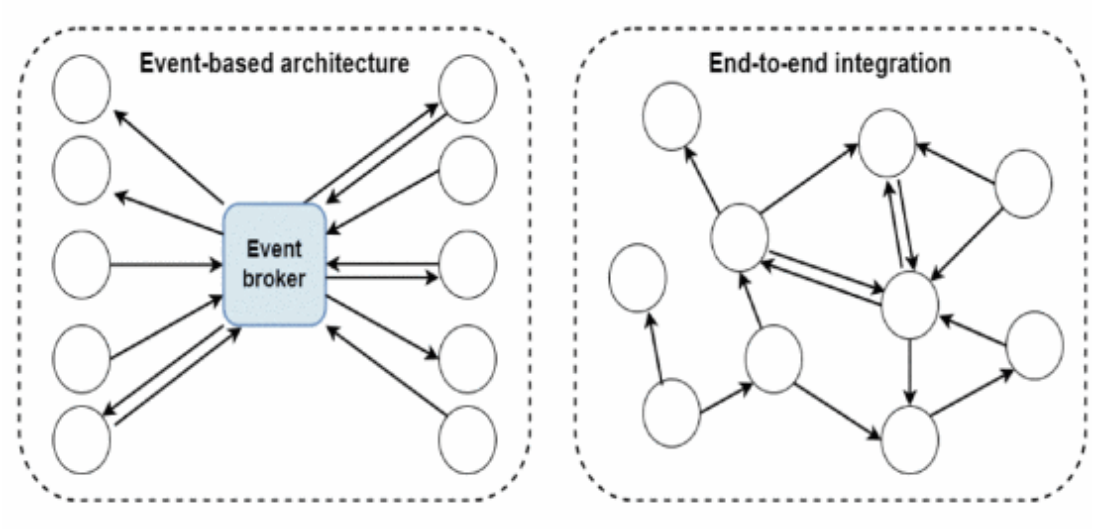

Figure 3. Comparison of event-based architecture and end-to-end integration. [3]

The structure of UNS usually follows a hierarchical naming inspired by the ISA-95 model. Which is typically organized as Enterprise, Plant, Segment, Line, and Cell. The specific way of naming makes possible that all participants in the system can easily locate relevant data in the system and understand its meaning. Each level of this naming hierarchy also represents a logical boundary, much like in ISA-95. For example, the Enterprise identifies the company, Plant stands for the production site, and Cell represents the actual machine in manufacturing. [3]

The UNS can be implemented using different communication protocols. There are still requirements for the used protocols, which are that the protocols should support scalable data exchange, security features and have real-time capabilities [3]. But in practice, Message Queuing Telemetry Transport (MQTT) and OPC UA are most used protocols with UNS. Benefits of MQTT used with UNS are that it offers simple protocol with high scalability and low bandwidth usage together with its communication structure based on topics. To achieve this, MQTT topics and their hierarchies are freely defined by the user, which also makes so that MQTT lacks standardized structure. OPC UA on the other hand provides defined namespace model where data is represented in nodes. This makes OPC UA particularly suitable for UNS implementations, although it introduces additional complexity and resource demands compare to MQTT. The working principles of OPC UA and MQTT are presented more detail in later sections. [10]

3 Related Work

This section is intended to show what other OPC UA performance related studies are done. Purpose is to find similar studies with industrial PLC as done in the study of this thesis. The found and chosen studies include one testbed also utilizing industrial PLCs. The study is the best comparable to the configuration used in the performance measurements of this thesis. Due to the limited number of other studies found involving PLC-based setups, research using other lightweight embedded systems such as Raspberry Pi as a part of OPC UA communication has been also reviewed. Software-only studies are also discussed due to the mentioned lack finding closely related studies.

A study from L. Freitas et al. [9], utilized Omron NX1P2 PLC which controls different sensors and actuators used in the machine. The PLC was then connected to an IoT device via Ethernet I/P sending the data to a separate PC using OPC UA. The performance of OPC UA with both client-server and PubSub model was analysed and compared with MQTT being a lighter communication protocol. This setup was the closest to a testbed used in this thesis which the authors could find. Another research still rather similar to the implementation of the one in this thesis was discovered. The study from S. Profanter et al. [11] did not utilize PLC in its setup. It instead used testbed with separate client machine connected to a Linux-based server working as a remote host for latency and throughput measurements. The connection between the devices was done using a network switch. The conducted study compared OPC UA with wider range of different communication protocols, which fall out of the scope of this thesis.

Several studies investigating OPC UA performance have been implemented using various versions of the Raspberry Pi single-board computers [2], [7], [12], [13]. These platforms were configured in multiple roles withing the used testbeds. They functioned as OPC UA servers, clients, monitoring devices, or even emulating sensors generating process data. The Raspberry Pi's were often connected to personal computers in multiple different ways. Either via a network switch, a broker in setups utilizing PubSub model, or in some cases even wirelessly. In many of these setups, a PC operated as the primary server responsible for publishing data. Or alternatively for monitoring the communication to record performance metrics. The metrics most commonly evaluated in these studies were communication latency, CPU and memory utilization. While the study in this thesis was conducted with industrial PLC, these studies provided good ideas for testbed implementation such as emulated sensor, separate monitoring device for data communication or the metrics for performance measurements.

In the work by S. Cavalieri and F. Chiacchio [14], the OMNeT++ framework was used to model the behaviour of physical hardware and analyse OPC UA communication performance. This was done in

the study within a simulated network environment. The setup was configured using the tools available in the used OMNeT++ framework. Performance metrics such as RTT and transmission delay were measured with respect to different points of data exchange. Which shows an example of a performance study using already available dedicated performance measuring tools. Yet Another study from X. Wang et al. [15] conducted the performance evaluation in totally virtualized environment. They utilized Docker containers, which were configured as different parts of the system. Including servers, clients, and brokers. The used configuration is commonly referred to as a “Docker Swarm” [15]. The Docker Swarm approach enabled the research to easily simulate different distributed communication with less physical hardware. As in many other studies, the OPC UA communication stack was implemented using open-source tools. The primary performance indicators analysed in this research included latency, packet loss ratio, and CPU load.

4 Industrial Communication

The following chapter reviews industrial communication relevant to the analysis of OPC UA performance study later in this thesis. Establishing the technical foundation and introduces concepts for understanding how data is often exchanged in industrial settings.

First, the fundamental principles of industrial networking are introduced. This includes describing how modern systems apply IP-based communication to enable reliable and scalable communication between multiple devices. Then the evolution of earlier Machine-to-Machine (M2M) and fieldbus protocols are discussed to provide context for the shift toward Ethernet- and Internet-based communication architectures. After these fundamentals and background, the chapter goes thorough the topic of OPC UA. As this is the main communication method used in the study, its purpose and advantage, together with its main communication models are discussed. OPC UA integration with TSN in form of UAFX is also presented. Other common alternative communication frameworks are addressed. These include protocols MQTT and AMQP which both work also with OPC UA as the standard on top of them. These specific protocols were chosen to be discussed due to their appeal as an alternative communication framework for the collaborating company. The chapter concludes with a review of used communication metrics in performance research which are partly utilized in the study also.

4.1 Network Foundations for Industrial Communication

To understand the transport mechanisms underlying OPC UA, a brief overview of the conceptual foundation for layered network communication. Which is known as OSI (Open System Interconnection) model.

The OSI model was developed by the International Standards Organization (ISO). OSI is often considered as the reference model for structure and function of data communication models. While OSI model is primarily a theoretical reference, its terms are well understood in the field of data exchange in general. The framework of OSI model is often used for mapping real-world protocols to their conceptual roles. OSI reference model consists of seven abstract layers illustrated with short description of each layer in Figure 4. This type of models is often referred to as a stack or protocol stack. Naming simply comes from the model looking like it has multiple building blocks stacked on top of each other. The stack structure gives the benefit for the layers not having to have the information of what is done for the data in next layer. It only needs to know it is passed to down, making the network structure to be easily modifiable. It is important to mention that each layer of the stack does not define a single protocol but rather a single function of the communication. [16]

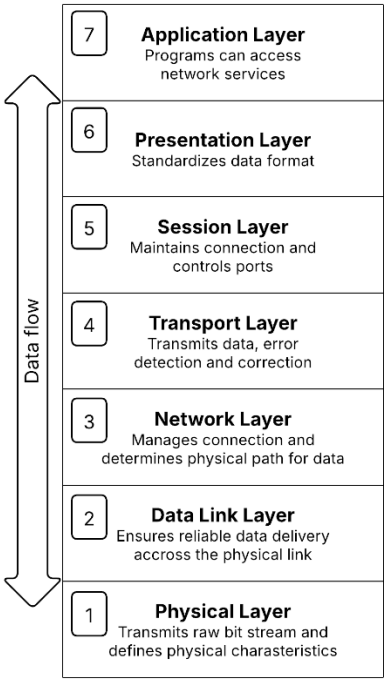

Figure 4. OSI reference model structure, basic functions are explained for each layer. [17]

In industry, a simplified model with only four layers called TCP/IP (Transmission Control Protocol / Internet Protocol) is often used instead of OSI model. The naming of the model simply comes from the two most relevant protocols used in it, TCP and IP. TCP/IP model was built to support communication across heterogeneous physical networks. The structure of TCP/IP model is compared to the OSI model in Table 1, showing how each 4 layers relate to the larger seven-layer OSI model. [17], [18]

Table 1. TCP/IP stack and OSI-model comparison with corresponding layers illustrated.

OSI model	TCP/IP model
Application Layer	Application Layer
Presentation Layer	
Session Layer	
Transport Layer	Transport Layer
Network Layer	Network Layer
Data Link Layer	Physical Layer
Physical Layer	

Many industrial communication frameworks such as OPC UA are built on top of this simpler TCP/IP stack instead of using the directly the OSI model. While OPC UA utilizes TCP/IP as its transport backbone, it implements its own security features and data modelling layers in it. Using the unified security and data modelling helps with interoperability across complex industrial environments, while still benefitting from the reliability and scalability of standard IP-based networking. [17], [18]

Next, each layer of the TCP/IP model are explained with describing their functions, typical protocols, and role communication within industrial networks. This information is useful for the later study of the OPC UA performance, as it helps identify which layer may affect the observed anomalies.

4.1.1 Application Layer

The application layer is the topmost layer of the TCP/IP stack. It combines the functions of the top three layers of the OSI model, which is also clearly shown in the comparison table 1. Application layers works as the interface between software and underlying transport mechanisms of the network, including all communication protocols that utilize transport layer services to deliver data to end users or applications [16]. In general internet, application layer provides the essential network services, these are for example file transfer, remote login, email exchange, web browsing, and name service [19], [20].

Application layer not only defines the rules for data exchange but also ensures that transmitted data can be properly interpreted by the receiving application, since it is the last layer of the communication in both ways. It typically manages functions such as data encoding, encryption, and session control. These are key features in ensuring consistency and interoperability between communicating systems. [18], [20], [21]

Applications operating at this layer interact with the network through ports and sockets. These serve as logical endpoints for communication. The use of ports and sockets enable multiple applications to use same network resources simultaneously, while maintaining separate communication sessions. [17]

In industrial communication, the application layer works as the host for protocols designed for industrial data exchange. Beyond conventional internet applications, industrial communication frameworks such as OPC UA, MQTT, and AMQP operate on this layer. For understanding the relevance in the study, OPC UA mainly functions as an application-layer protocol within the TCP/IP stack. The feature of OPC UA operating on application layer allows it to utilize the underlying transport technologies such as TCP or UDP to achieve standardized data modelling and information exchange. [18], [22], [23]

4.1.2 Transport Layer

As discussed in background section, in many frameworks, applications or devices need to have end-to-end connection between each other to enable communication. The transport layer responsible for this end-to-end connection. Situated second. between application layer and network layer, it ensures that data generated by one application or device is correctly delivered to its intended receiver [16], [17]. The transport layer focuses on process-to-process communication, offering mechanisms for segmentation, reassembly, error control, flow control, and connection management [21].

There are two most widely used protocols in the transport layer. One being the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP). TCP provides connection oriented and reliable transport service. It ensures that data packets are delivered in sequence, without loss or packet level duplication. TCP implements several mechanisms to achieve mentioned behaviour, retransmission of packets being the most notable of these [17], [19]. UDP on the other hand is considered as a connectionless protocol. Connectionless here means that it offers low-overhead and best-effort delivery without build in reliability features for detecting packet-loss or duplicates on the transport layer. UDP is often considered more usable for applications where speed and low latency are prioritized over accuracy, especially when it inherently does not guarantee order or delivery. [16], [20]

Within transport layer OPC UA can utilize either TCP or UDP as its transport protocol. Depending on the used communication model. In OPC UA TCP ensures reliable, ordered, and error- checked delivery of messages between clients and servers, where UDP extends support for PubSub model, achieving lower latency with trade-off in reliability. These two different models are discussed in more detail in the OPC UA section itself. [24]

4.1.3 Network Layer

Positioned between the transport and physical layers is the network layer. As the second lowest layer, it creates a framework that allows systems on different physical networks exchange data [16], [17]. The primary protocol operating at this layer is the Internet Protocol (IP). IP defines mechanisms for addressing, routing, fragmentation, and reassembly of data packets, also known as datagrams, while they travel through different networks [19], [20]. Important clarification here affecting the study is to understand that segmentation happened on transport layer where fragmentation happened on network layer.

Datagram is the main unit of transmission at the network layer, containing needed information for data routing addition to the actual data. This separation of information and data makes datagrams constructs from a header and payload sections. The structure of a datagram is illustrated in Table 2 with the "DATA" section cut, meaning that it is often larger relative to its size in the shown table. The header carries essential control information and the payload portion encapsulates the data received from transport layer. [16]

Table 2. Structure of IP datagram header with cut payload seciton. [16]

Version	IHL	Type of Service	Total Length	
Identification			Flags	Fragmentation Offset
Time to Live		Protocol	Header Checksum	
Source Address				
Destination Address				
Options				Padding
DATA				

IP provides connectionless, best-effort packet transmission services where logical addressing and routing ensures that packets are directed toward their intended destination using IP addresses and routing tables [20], [21]. It does not create end-to-end connections for data, nor does it guarantee reliability or error correction since they are the responsibilities of transport layer[16], [17].

The IP has two versions primarily used in today's communication. IP version 4 (IPv4) and IP version 6 (IPv6). IPv4 has its limitations with having limited number of addresses only up to 2^{32}, where IPv6 was created to tackle this limitation with totally new addresses. Unfortunately IPv6 can't operate with IPv4 decreasing the possibilities for interoperability [21]. IPv6 has not reached wide usage in industrial sector yet, while IPv4 still remains the main protocol in use despite its limitations. [16]

4.1.4 Physical Layer

The bottom layer of the TCP/IP stack, Physical Layer, is often also called as Network Access Layer or Link Layer. Its primary responsibility is to provide the mechanisms by which data is physically transmitted over the network medium. Ultimately enabling whole transfer of data to the other devices which are connected to the same physical wired or wireless network [16]. The responsibilities of physical layer also extends to manage how physical addressing is implemented using Media Access Control (MAC) addresses. [20], [21]

Used protocols within the physical layer must comply with the specific characteristics of the underlying technology. This includes packet formatting, addressing methods, and different constraints in data transmission [16]. Each physical medium such operating on this layer as Ethernet or Wi-Fi has its own protocol defining how data is formatted and transmitted. This enables the flexibility of the TCP/IP model being able to operate over virtually any network medium. [17], [19]

As shown in the Table 1, in TCP/IP stack physical layer also, includes the data link part from the OSI-reference model. The table clarifies why the physical layer is not just raw bit transmission but rather handles also different tasks explained in this section.

One key feature to achieve all this is the encapsulation of IP datagrams into data frames which are eventually broken down to bits is. The process is then done backwards in the other end of physical layer. The encapsulation of IP datagrams helps in mapping the addresses to corresponding physical MAC addresses, enabling the transmission the data through the network hardware [16], [20]. While some physical layer technologies implement their own error detection and flow control, others rely these matters on upper layers such as the transport or application layers [17], [21].

In physical layer, OPC UA often operates on standard Ethernet-based infrastructure [18]. The mentioned communication median directly affects the RTT and its jitter in communication, since these parameters mostly depend on the characteristics of physical medium. Other affecting factors are the network layout and how switches and routers handle data.

4.2 From Serial Fieldbus to OPC UA

During 1970s and 1980s, the communication systems used in industry began shifting from traditional analogy signal communication to digital networks. These digital networks in industry are nowadays often called fieldbus networks. They reduced wiring complexity and introduced a structured means for connecting sensors and actuators to their control units. Fieldbus protocols such as Modbus and

PROFIBUS became the leading technologies for serial communication. They typically operated over RS-485 on the physical layer of TCP/IP stack. [18]

Even when these traditional fieldbus systems proved to be robust, simple to implement, and highly reliable for real-time process control, as industrial automation grew in scale and complexity, requirements for bandwidth, transmission distances, and interoperability increased. These demands revealed the limitations of traditional fieldbus systems using serial communication. [18]

Ethernet, which was already dominant at that time in office and IT environments was recognized as a promising candidate to tackle these problems. Its higher data rates, hardware availability, and compliance to relevant standards offered advantages over traditional fieldbus. The idea was to be able to extend the use of it from the higher levels of communication to the lower communication layers making communication more uniform. [25]

In order to connect the network layer to the physical wired communication in physical layer, ethernet used access method called CSMA/CD (Carrier Sense Multiple Access with Collision Detection). This method made the communication non-deterministic since transmission timing varied by random back-off intervals happening from multiple devices trying to transmit data over the same medium causing collisions in communication. Mentioned behaviour was unacceptable for real-time control in industrial systems where message delivery must occur within fixed time intervals. [25]

To come up with suitable digital communication method for industrial automation, vendors introduced real-time Ethernet (RTE). RTE solutions either modified or extended the standard ethernet to answer these problems stated previously. These modifications often included either to change or bypass lower parts of the TCP/IP layers. One major modification in RTE was to replace the CSMA/CD with more suitable access method resulting in deterministic communication timing required for filed-level control. On top of this, usage of RTE made integration with higher-level IT networks possible since they used the same underlying ethernet. Examples of RTE are Ethernet/IP, PROFINET, EtherCAT, and Modbus TCP. [18], [25]

Even though RTE seemed like a perfect answer for industrial automation communication, it still introduced new fragmentation of the communication. Each vendor's implementation to achieve the desired technological requirements differed from another. In result in many different incompatible solutions was formed. One of the major challenges was that integrating low-level RTE fieldbuses with higher level IT applications was difficult and often required custom drivers or gateways. The lack of interoperability was often viewed as a missed opportunity to establish a single unified communication solution based on ethernet [18].

As an answer to this interoperability problem with many vendor specific solutions, the OPC Foundation introduced OPC Classic in the 1990s. OPC Classic was meant to standardize software interface that enabled vendor-neutral communication and ultimately bridging the formed communication gap between lower-level OT and higher-level IT systems. [26]

4.2.1 OPC Classic

Instead of providing yet another totally new standardized communication protocol, OPC foundation introduced a software-level interface at first built on Microsoft's Object Linkin and Embedding (OLE) solutions running on Windows operating system. This solution was later transitioned to rely on Microsoft's newer COM (Component Object Model) and DCOM (Distributed Component Object Model) solutions with standardized interfaces. [26], [27]

The OPC Classic was often divided into three different standardized specifications, each addressing a specific data domain within industrial automation [26], [27], [28], [29]:

- OPC Data Access (DA): Enabled real-time data exchange of process values, quality and timestamps between devices such as sensors, PLCs and SCADA systems.
- OPC Alarms & Events (A&E): Managed on-demand event and alarm notifications, including state changes and event-driven messages.
- OPC Historical Data Access (HDA): Provided standardized methods for querying and analysing time-stamped historical data.

Using the design built on top of Windows simplified integration between sensors, PLCs and SCADA. This made OPC classic widely used in automation solutions in industry. The three specifications of the OPC Classic allowed different software applications to communicate with automation hardware using one common interface. Doing this eliminated the need for custom drivers to connect devices from different vendors, reducing the complexity of integration. [26], [28]

OPC Classic became widely used in industry. Even with its success, it still was not perfect, leaving room for improvement. Most notably it inherited several of its limitations from its Microsoft Windows operating system. The reliance on COM/DCOM restricted the use to only Windows systems. There was a way to go around this by emulating Windows on other platforms, but it still often caused compatibility and performance issues. In addition, configuring DCOM for secure remote access, often required in industrial environments required complex firewall and authentication settings. Also, each OPC specification maintained its own address space, preventing unified handling of related process data. [27], [29]

To answer these issues, the OPC UA was developed in the mid-2000s. It was created as a service-oriented successor for OPC Classic. It integrated all functionalities of the older model into a single, secure, and platform-independent framework where Microsoft ecosystem was no longer required. [7], [28]

4.3 OPC Unified Architecture

The bottom principle of how OPC UA achieves its platform independence, is by keeping communication layers separate from the specific hardware or operating system it runs on. OPC UA implementations range from sensors with embedded controllers all the way to cloud servers in industrial automation [14]. OPC UA follows SOA separating the information model, services, and transport layers [14], [30]. With usage of SOA, it allows technologies to be replaced or updated without modifying the entire system, providing flexibility for integrating new devices even with different communication models [31].

In OPC UA all data is modelled as nodes. These nodes are the single structure block in OPC UA which can vary from objects, variables, methods, and events for example. Creating an address space that can represent anything from a single sensor to entire production line. Each node has also attributes working as properties describing a node. Attributes include identification information from each node, which can be considered as the metadata about the node. While this kind of detailed node based data structure with additional metadata can require more processing power compared to simpler communication protocols, it also the enabler to achieve OPC UA key benefits of reducing the need for extra middleware between systems. [2]

While the study of this thesis does not include any security features in the OPC UA communication used, nor is any encoding methods varied to measure their effect in the communication performance metrics. The primary security and encoding methods are presented briefly. This is in purpose to highlight that in actual production usage, these have direct impact on the communication performance and should be considered when implementing OPC UA communication.

The used security features in OPC UA are encryption of data, authentication for users and applications, and signing of messages which ensures that data has not been modified during data transfer [14], [30], [32]. To achieve these security features standard defines three main security modes, which can vary in name by the vendor and used version. The names for these three modes the standard uses are None, Sign and SignAndEncrypt [32]. The names of the modes are rather self-explanatory and can be configured depending on the required protection level and system configuration.

OPC UA offers three main methods for encoding also. These are binary, JSON and XML encodings. The UA Binary encoding often combined with UA TCP offers lowest latency and highest throughput in client-server model since binary being the most wire level method of encoding. JSON encoding is primarily used with PubSub model providing web-friendly method of encoding. The XML encoding on the other hand can be understood rather as legacy encoding model specially from OPC classics utilizing COM/DCOM. [31], [33]

4.3.1 Client-Server

The OPC UA client-server model is the traditional and most widely used communication approach within the standard in industrial OT communication. In this model, the server provides access to its data by exposing it through a specific port, while the client requests or subscribes to relevant information. The communication is typically done over TCP, which naturally ensures reliable and ordered message delivery as a communication protocol itself [24]. To better understand how the communication of clients requesting and servers responding for data occurs, Figure 5 presents how does the communication between client and server typically operates and what ways data is exchanged.

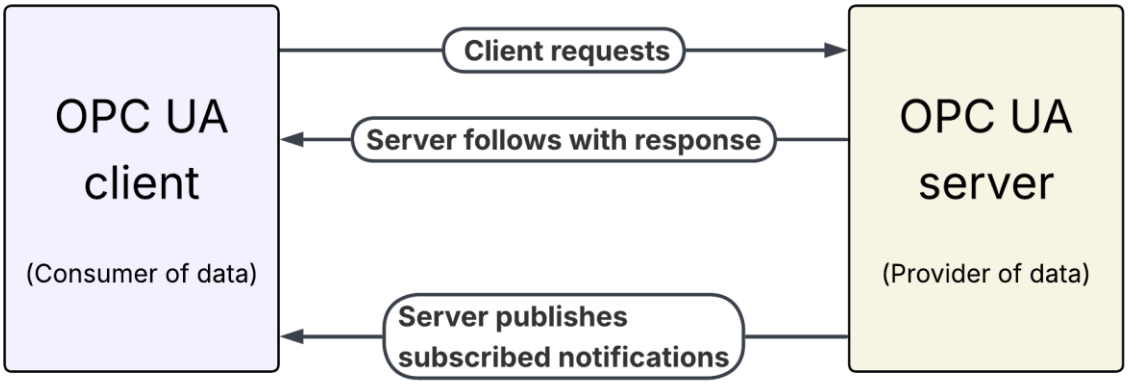

Figure 5. OPC UA client-server communication exchange methods separating request and subscription.

As illustrated in Figure 5, data can be obtained either through periodic polling representing the upper request and response sequence or through subscriptions showing on the lower from going only from server to client. In the request/response polling approach, the client periodically sends read requests to the server [34], [35]. The server responds with the requested data and client then receives the current values of variables. In contrast, the subscription mechanism allows the server to monitor selected data and notify the client when changes occur [34], [35]. While polling used in the industrial context case

and therefore in use for the study of this thesis, the theory of subscription model is discussed to provide knowledge of its advantages to provide asynchronous communication with lower communication load. The rather detailed explanation of the subscription model is justified for it being the possible interest for future work.

In subscription model, the connection is separated into Session, Subscription, and MonitoredItems [14], [31]. A session can be considered as a link between client and the server. The link is created on top of secure communication channel where client must open its own Session, and the server must allocate memory and processing resources for each of these sessions. As the number of clients for a single server increases it naturally causes higher CPU and memory usage on the server [24]. Which directly impacts the performance capabilities of the communication where the lover number of clients per server ensures better performance with given hardware. Sessions are independent of the underlying communication channel, meaning that the connection can be re-established if the network temporarily fails. Being an important benefit of the subscription model.

Within a Session, the Subscription manages the exchange of information. The information exchange can be for example variable updates or event notifications. For each subscription publishing interval is configured. Determining how often the server sends updated to the client. To control the sent information, subscription model integrates a feature where the client sends a list of publish requests to the server. These requests are queued until new data becomes available. This makes possible so called deadband filtering where a fixed threshold can be set for example a variable to change before new data is updated to the client. Which ultimately helps to reduce the overall network traffic. [31], [36]

MonitoredItems are the smallest parts in a Session. They are created by the client to track single nodes in the server's address space [14], [31]. For each MonitoredItem specific interval of sampling can be configured. The sampling in this context means how frequently the server checks for possible data changes. Together with sampling interval the queue size can be configured separately for each MonitoredItem. Queue size defines how many values are temporarily stored before being sent to the client. This buffering makes it possible that if the sampling rate of the client is higher than the publishing rate or the server, new values are queued until they can be transmitted.

In OPC UA client-server communication, data loss may still occur at the application level regardless of whether polling or subscription model used. While TCP in nature avoids clear data loss in the transport level with retransmissions, when the rate of data generation exceeds the transmission capacity with polling, some values are simply never requested. In case with subscription, OPC UA servers may discard data updates when MonitoredItem's queue overflows. This means that when the configured queue is full, the oldest or newest values are replaced according to the discard policy set in

the server [37]. This OPC UA subscription over TCP overflow handling is further illustrated in Figure 6. While in the polling-based model, the application-level data loss can mostly be due to timing mismatches explained in detail during the study of this thesis.

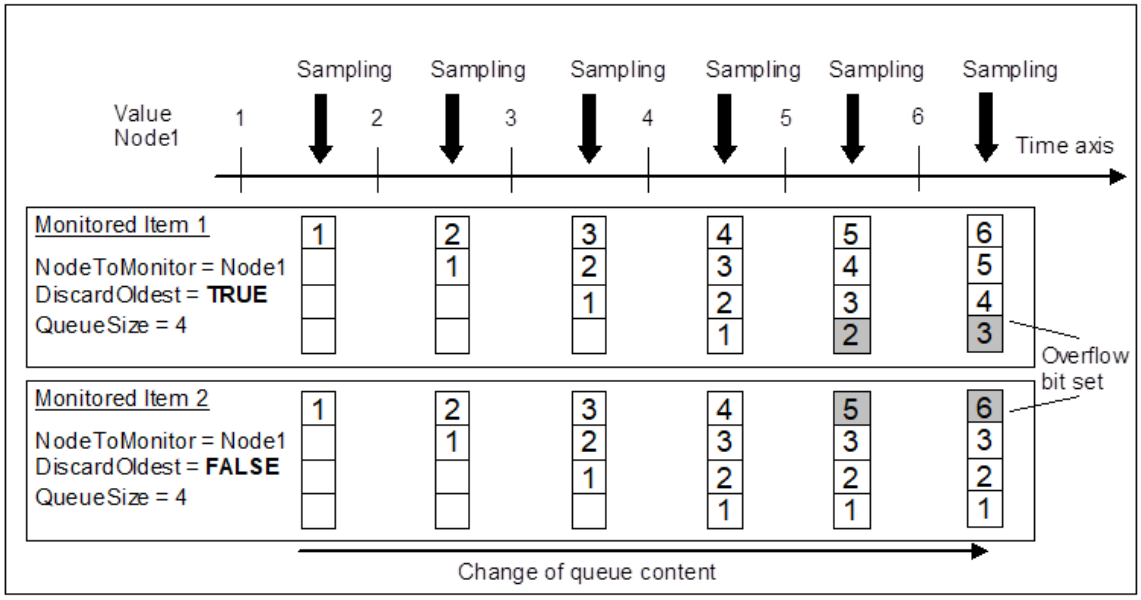

Figure 6. Subscription queue overflow handling showing a possible function for experienced data loss in client-server communication. [37, p. 4]

4.3.2 Publish-Subscribe

The OPC UA Publish-Subscribe (PubSub) model has a communication pattern where applications working as communication participants act as publishers or subscribers rather than clients and servers. It is important to mention that the PubSub model should not be confused with the client-server subscription method. In PubSub, a publisher's main functions are to read data from its data sources, generates messages, and send them out to the network. Subscribers declare interest in certain specified data and receive new values when publishers publish them. Publishers and subscribers do not need to know about each other's participation in communication network like in end-to-end communication. Meaning and they do not establish sessions or maintain connections. Any device part of the communication network can participate as a publisher, subscriber or both much like in client-server model. [38], [39]

As discussed, each new client adds processing load and memory consumption to the server in client-server communication. Advantage of PubSub model is to avoid this by sending data periodically without maintaining individual session. The general concept of PubSub with publishers publishing messages through the network infrastructure to subscribers interested in specific topics, is illustrated in Figure 7. This different model of communication in PubSub reduces the server's memory footprint and removes much of the overhead of data created by individual sessions and connections [40]. The

different working principle of PubSub makes it well suited for IIoT scenarios where data from field-level devices must be forwarded cloud applications [41]. With this, a function can be achieved where multiple sensors for example publish data to the network, and one subscriber can be subscribing to all those sensors with smaller communication load than in client-server model.

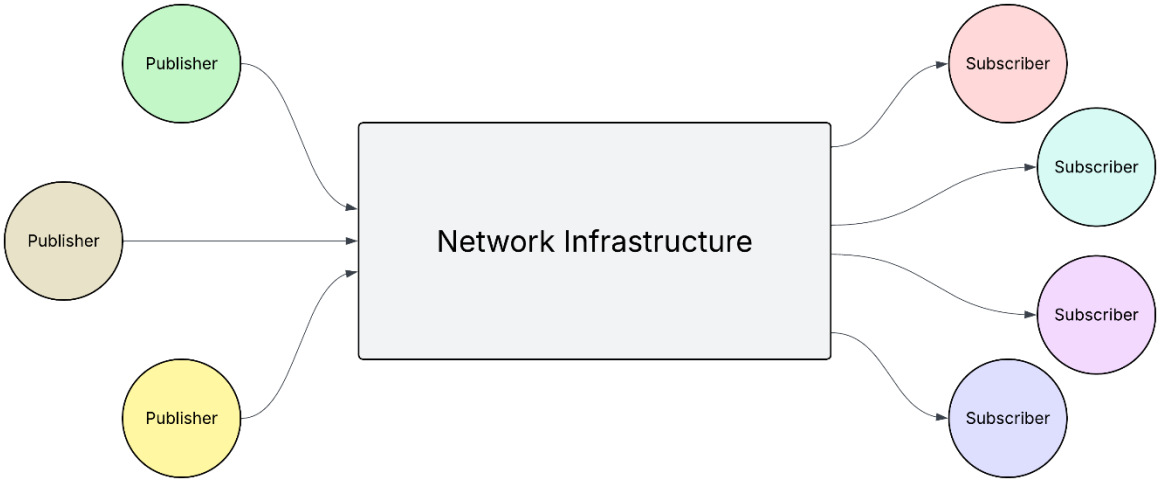

Figure 7. PubSub model where publishers publish data through network infrastructure and subscribers receive it based on their interested topic. [42]

The OPC UA offers two main methods of implementing PubSub communication: broker-less and broker-based [38], [39]. These models differ in how data is distributed on the network, the middleware they require for the communication, and the performance or reliability they offer.

In the broker-less model, publishers send messages directly onto the network often using UDP unicast or multicast messages [42]. The usage of broker-less model often relies on standard network equipment such as switched and routers that support multicast. After the messages containing the data have been transferred to the network, then these messages are forwarded to every device that have joined to the same multicast group. This way only subscribers which are configured for the same multicast address will need to process the incoming data [38], [42]. The Figure 8 visualises how broker-less communication works over UDP multicast highlighting the direct connection with no need for any additional middleware in the PubSub communication.

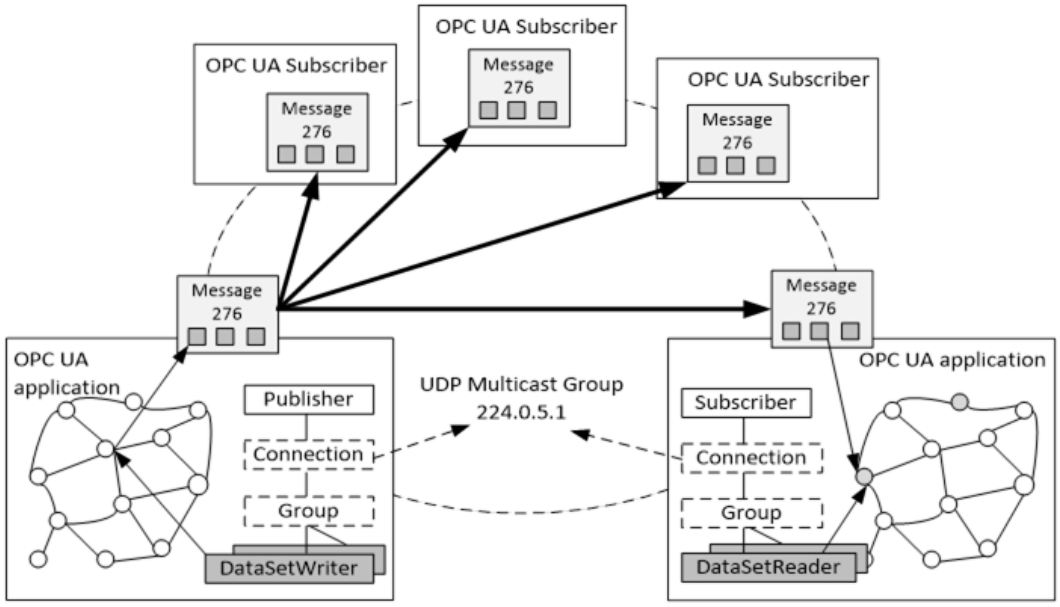

Figure 8. Illustration of broker-less model communication using OPC UA UDP. [42]

Even according to the on the OPC UA standard itself, using the UDP multicast does not guarantee "timeliness, delivery, ordering, or protection from duplicates" [42]. Reliability can be improved through the usage of sequence numbers, message repetition, or by sending full datasets at each cycle, but guaranteed delivery requires additional mechanisms from higher layers of the communication stack. These mechanisms are introduced in later section. Therefore, it can be concluded that broker-less UDP communication is best for low latency local networks where occasional variation in reliability is accepted. On the other hand, because this communication does not include additional middleware, it provides lower latency, smaller overhead, and efficient communication to multiple subscribers simultaneously [42].

In the broker-based model the communication between publishers and subscribers is handled using dedicated middleware called the broker [38], [39], [40], [43]. In broker-based model, publishers sends messages to the broker. The broker then temporarily stores these messages and forwards them to subscribers based on the given topics of the messages. Subscribers on the other hand interact only with the broker to declare which topics they are interested in [38], [39]. This decouples communication so that subscribers do not know which publisher the message came from, due to the broker handling it. Usage of broker-based model reduces the load on publishers, especially when many subscribers require the same data. The broker overview of the PubSub infrastructure can be observed from Figure 9 in a similar manner as was with UDP multicast for ease of comparison.

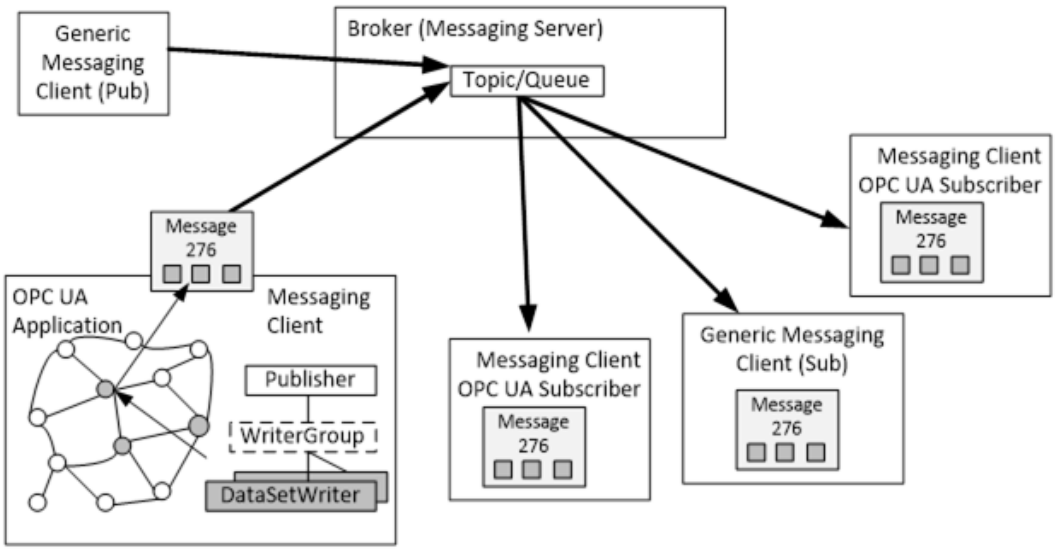

Figure 9. OPC UA PubSub broker-based model overview where broker handles message forwarding. [43]

OPC UA PubSub supports MQTT and AMQP brokers. MQTT is lightweight and commonly used in IoT and cloud systems, while AMQP offers richer messaging features and more complex routing capabilities [39], [40]. These are also presented later as standalone communication protocols for comparison without OPC UA integration of them only working as brokers. In OPC UA PubSub, MQTT or AMQP appears as standard broker communication to subscribers, even though the payload contains OPC UA encoding structures such as UADP (binary) or JSON (text) [39], [43], [44].

With broker-based communication, brokers can queue messages across networks as seen in Figure 9, fan out large number of subscribers, and even store messages from subscribers whose lifetimes do not overlap with the publisher [43]. Enabling integration across cloud platforms. To be noted, because MQTT and AMQP use TCP, broker-based PubSub is generally unsuitable for deterministic communication [40]. Also meaning that applications with strict timing requirements in communication, the broker could quickly become the bottleneck.

4.3.3 OPC UA FX

In both PubSub models, it still lacks the deterministic and time-synchronized behaviour desired for field-level communication. Such requirements have traditionally been addressed by dedicated fieldbus solutions using RTE protocols mentioned previously in this thesis. The PubSub would otherwise be considered as a potential alternative to replace the traditional fieldbuses in the lover levels of automation stack due to its low overhead and otherwise beneficial design. To address this, OPC

Foundations' Field Level Communications (FLC) working group introduced the OPC UA Field eXchange (UAFX) specification.

UAFX builds on the core OPC UA standard to enable interoperable communication at the field level. It uses the PubSub as the communication model and has selected mechanisms from Time-Sensitive Networking (TSN) standards to achieve the wanted behaviour which PubSub is otherwise lacking. [45], [46], [47]

TSN is a set of extensions to standard ethernet deployed by the IEEE 802.1 TSN Task Group. The idea of TSN was to introduce lower latency, jitter and deterministic communication in Ethernet networks. By relying on open IEE standards, TSN reduces reliance on proprietary industrial communication technologies which often are vendor-dependent . [24], [48]

A key feature of TSN is that it enables real-time and best-effort traffic to appear on the same physical network. Other features are the time synchronization between devices using Precision Time Protocol (PTP). The PTP ensures that all of them participating in the communication operate on a same time base. Which logically enables frames to be transmitted at correct times. Another feature that TSN provides is traffic shaping and scheduling which divides network communication into repeating cycles, and assigns exclusive time slots for high priority traffic to avoid unpredictable buffering delays. [24]

The PTP which TSN uses, is designed for precise time synchronization. PTP operates in defined networks using master-slave hierarchy where a single device is the master keeping the reference time and others are slaves trying to synchronize with the master time. It uses hardware assisted timestamping to achieve low jitter and high accuracy. [50], [51]

Other methods are also used for time synchronization. And since the thesis is done for pharmaceutical manufacturing environment the Network Time Protocol (NTP) should be mentioned since vendors GMP guidance recommends using it as the timing synchronization protocol [49]. NTP uses software-based timestamping instead of hardware and prioritizes robustness across large systems [50]. This results in lower accuracy than PTP, but broader interoperability.

To be noted that UAFX is not just OPC foundations attempt on trying to come up with OPC UA PubSub model over TSN, but rather a full specification optimised to possible replace traditional fieldbus systems and RTE protocols with unified interoperable system over time [45], [52]. Since the UAFX was released in 2022, it is still under development and newer releases with updated model structures are to be expected [46], [52].

4.4 Other Industrial Communication Frameworks

As previously stated, part of this thesis was to introduce other possible communication methods for the industrial usage for the collaborating company. While different variations of models in OPC UA provide wide range of choices, few separate communication protocols outside of OPC UA scope are presented to give other perspective for solving the observed issue. MQTT and AMQP was chosen due to their close relation in OPC UA in broker-based PubSub communication. Next MQTT and AMQP are introduced briefly with some highlights of their pros and cons in overall communication performance in industrial settings.

4.4.1 MQTT and AMQP – Protocol Overview

MQTT

MQTT was originally developed by IBM, but is now later standardized by OASIS [13]. MQTT operates via TCP/IP stack and uses PubSub as its communication model. In MQTT all participants of the communication can be referred as the clients. They use central broker to transfer information, similar to what is explained in previous section with broker-based PubSub OPC UA. MQTT offers few different kinds of readymade brokers such as Mosquitto or HiveMQ to store topic information and distribute published messages to all relevant subscribers [11], [54].

The MQTT protocol supports different Quality of Service (QoS) levels. These range from at most once to exactly once guarantees. The QoS affects the presence of duplicates and how flexible and light versus reliable the communication is. Providing flexibility for different reliability requirements. The Sparkplug specification refines MQTT with an optimized topic namespace, standardized payload structure, and design considerations for SCADA integration, this should be considered when wanting to integrate native MQTT as part of industrial production. [13], [53]

MQTT itself has small overhead and binary message format. This differs from MQTT usage along with OPC UA where it increases the overhead. Despite MQTT being designed for efficient communication in resource-constrained and unreliable environment, standalone MQTT also has notable limitations in certain industrial systems where timing restrictions are present. These constraints include the usage of TCP as stated previously and in security wise its requirement for TLS/SSL for secure communication [13]. MQTT itself also offers fewer control and configuration options compared to heavier messaging frameworks [13]. MQTT communication latency can vary depending on network conditions and broker processing. Latency often ranges from milliseconds to seconds, reflecting a design focus on reliable message delivery rather than deterministic communication with timing guarantees [13].

AMQP

While Advanced Message Queuing Protocol (AMQP) was not at the first interest of the collaborating company in a way that MQTT was, nor really either in a use case for this industrial context as will be explained shortly, it will still be introduced due to its relevance in OPC UA and show how it differs from MQTT. This is done to provide better understanding of the whole sector of industrial communication protocols. AMQP is also an open messaging protocol. Its intent is to to support reliable, interoperable, cross-platform communication using brokers and queues. It is widely adopted in industries requiring robust data exchange with high volumes. Good examples of these industries are finance and cloud computing. Here organizations like JP Morgan and NASA have reported large operational use of AMQP [13]. AMQP supports multiple communication models. Relevant ones are one-way messaging, request-response, publish-subscribe, and transactional exchanges. AMQP also operates over TCP using binary encoding and includes mechanisms to ensure reliable delivery such as broker restart handling, and data priority handling. [13], [55]

AMQP's messaging model is built using predefined components in communication: producers, consumers, brokers, exchanges, bindings, and queues. Producers send messages to exchanges, which then route the message according to its binding rules into appropriate queues. From these queues consumers retrieve them. Building on these predefined components introduced briefly. It gives AMQP the ability to support a wide range of messaging scenarios while maintaining strong reliability guarantees. These are regarded features in finance and cloud computing. [13], [55]

Because AMQP also relies on TCP and requires brokers to maintain an active connection for each client, its scalability might be limited when communicating with large numbers of lightweight devices simultaneously [55]. In relation to MQTT, AMQP it typically uses more bandwidth for overhead and does not provide mechanisms for resource discovery [13]. Making MQTT more desirable solution specifically in many IIoT systems. AMQP also relies that all participants in communication know the used broker's address in advance [13]. This makes extra end-to-end connection to the network increasing the complexity. It should be noted that AMQP is still a robust choice for right use case. These cases can be applications demanding guaranteed delivery and advanced routing logic such as is the case in finance and cloud computing as stated previously.

4.5 Performance Metrics

Following section reviews the key performance metrics commonly used when testing IIoT communication performance. The purpose for this section is to offer an overview of the used metrics in OPC UA testbed setup together with other common metrics used in similar performance analysis

from literature. Chosen metrics from this section are used in the study to evaluate the given OPC UA testbed performance.

Performance metrics in communication systems are typically grouped into hardware related and protocol execution categories. Hardware related metrics are for example latency, bandwidth usage, CPU utilization, and energy consumption. Protocol execution metrics on the other hand are the number of transmitted messages, payload characteristics, and data loss [9].

Latency is one of the most critical parameters in industrial data communication, as many control systems rely on precise timing [56], [57]. Latency measurements may include round-trip time (RTT), application-level delay, or jitter measurements. These latency metrics can be influenced by many different factors from either the hardware or software side of the communication. For example, Bandwidth usage and packet overhead determine how efficiently a protocol transmits data, larger headers increases resource consumption which directly affects latency and also throughput [15]. CPU usage measures the computational load created by the data communication itself, this metric is especially relevant for devices with limited hardware resources [12].

Metrics related to protocol execution describes how well a communication system performs under realistic load conditions. Packet loss may arise from limited channel capacity, weak connectivity, or too high publish rates [15]. It can be also influenced by buffer overload as discussed previously in OPC UA client-server section. Packet loss and especially application-level packet loss or duplicates are the main concern of this thesis related to metrics in protocol execution. Another notable metric here is the time-to-completion (TTC), which provides an application-level view of latency. To give concrete examples of it, TTC is can be seen for example the time between publishing and receiving of data in PubSub systems, or the full read request response cycle in client-server communication [15].

5 Methodology

Now with the theoretical background discussed, methodology chapter presents the methodology for the study conducted. The purpose of the study was to evaluate the performance of polling-based OPC UA client-server communication in the industrial PLC scenario. Which was done to establish a reference point for the system's data transfer limits.

This section begins with the study objectives. Objectives include the practical problem identified in the company's production in pharmaceutical manufacturing. Next, the selected research approach is described with the scope of the study. Then the practical things related to the actual testing are explained with testbed architecture, experimental procedure, and measurement metrics for the performance study.

5.1 Study Objectives

The thesis was conducted in collaboration with a pharmaceutical manufacturing company. As this company operates in a pharmaceutical environment, production systems must comply with applicable regulatory frameworks. Few of the most meaningful regulations in relation to the study are GMP and GAMP5. These define constraints on system design, for this thesis particularly regarding reliability of stored data [58]. To comply with these requirements, the communication architecture of machines and their control systems must be designed according to these regulations and separately validated.

The primary objective of this study is to recreate a communication setup comparable to that used in the collaborating company's production and evaluate its performance under varying communication load. The data communication in test scenarios were designed to simulate realistic production conditions. To specify, these are the conditions in which communication anomalies have been observed. Different methods for generating the required load were applied to benchmark the communication setup.

The motivation for this study comes from the need to ensure reliable data in manufacturing processes. The anomalies observed by the collaborating company have caused uncertainty in consistency of data. This directly impacts the monitoring of the process and specially which is problematic, the quality assurance. Through systematic testing, this study aims to identify potential causes of these anomalies and to determine whether they originate from the OPC UA communication, the implementation of PLC logic, or other parts of the communication stack. The results may also support troubleshooting efforts by enabling certain components or mechanisms to be ruled out based on them.

Introduced in the literature review, OPC UA client-server model supports both polling-based communication and subscription mechanism through MonitoredItems. In Siemens systems, the subscription mechanism is implemented under the “Subscription” functionality. According to Siemens documentation, subscriptions have advantages due to their efficiency in reducing network load and enabling event-driven communication [59].

However, even with possible benefits of subscriptions, this study focuses specifically on polling-based OPC UA client-server communication. Here PLC cyclically reads individual tags from the server instead of receiving value updates through event based notifications as is the case in subscriptions [14]. The client’s polling frequency is determined by the PLC program cycle and the configured timer logic used to trigger read requests.

The decision to use the polling-based method was driven by its use in the collaborating company’s production environment. As mentioned, to analyse the observed anomalies under realistic conditions, the experimental setup was designed to closely replicate the production architecture within the constraints of available resources. For this reason polling based method was chosen together with Siemens S7-1500 PLC, since the same setup is used in the production. Detailed information regarding the selected hardware and configuration is provided in the Testbed Architecture section.

Varying load-based testing was selected as an appropriate approach because controlled load variation enables systematic evaluation of how polling-based OPC UA communication behaves as the chosen key parameters change. By increasing system load in controlled manner, it becomes possible to identify which parts of the communication affects the performance most. Also, the anomalies may not be observable under standard operating conditions in this testbed environment since it can operate only limited amount of time.

The study objectives are derived from the research questions presented in Chapter 1. Each research question examines a specific feature that can be varied and is believed to influence the performance of the used OPC UA communication. These features are total system load, polling interval, number of nodes, parallel read jobs, and payload size. With these, the study also aims to determine whether the anomalies can be reproduced and to analyse their underlying causes.

5.1.1 Industrial Case Context

This thesis uses the following observed reliability issue as an example case for the study setup. However, the findings of this study are intended to be used more broadly to similar OPC UA communication systems within the company’s facility. The observed issue is believed to occur in data

transfer between ISA-95 Level 0 and Level 2. In the specific case under investigation, the OPC UA polling-based client-server model is used for the data exchange.

A subsystem within a larger production line includes a device that measures dimensional data from the product in scale of millimetres. Used device utilises a radiographic measurement and operates with its own operating system. This is meaningful since it has internal data handling, formatting and temporary storage mechanisms. The device is operating as the OPC UA server for data transfer.

The term smart sensor is used in this thesis to refer to the specific measurement device. The measured dimensional values recorded from the smart sensor are then transferred along with other production relative information to the PLC of the whole production line using wired Ethernet. To be noted, this transfer from smart sensor is done through the company's internal network. Its effects on the communication are also mentioned in the Discussion section but not tested in the testbed setup.

To give better perspective, when looking at the ISA-95 automation pyramid standpoint, the smart sensor is on level 0 and the client PLC operates at Level 1. In relation to the smart sensor function, the PLC controls generation of alarms and process control depending on the smart sensor values. The PLC then transfers the data to Level 2 on ISA-95, where SCADA and HMI systems process and archive it. Here the measured data is archived using an integrated historian and analytics add-ons in the SCADA system [60]. Ensuring compliance with regulatory requirements and to enable use of measurement data for quality control. Process monitoring and operator interaction are also performed at this level through HMI. The overall communication is illustrated in Figure 10.

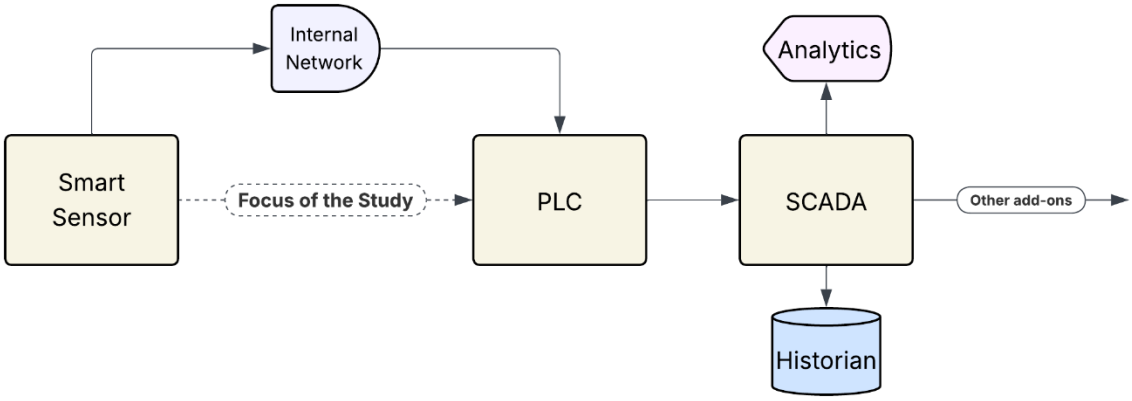

Figure 10. Data exchange hierarchy in industrial context case. The PLC reads the measured data from Smart Sensor via OPC UA client-server communication.

It should be noted that the data exchange includes significantly more than the dimensional measurement values alone, but for the scope of this thesis, the form of the other data was not studied.

This was due to it being production sensitive, but a rough estimate of its total amount was obtained by inspecting a backup copy from one of the production line's PLCs. The additional data naturally increases the overall communication load. Multiple variables are read and written through the PLC in parallel read jobs, further contributing to communication complexity. The PLC and SCADA with its add-ons are implemented using Siemens industrial automation products.

As mentioned, within this industrial context case communication anomalies have been observed and verified. The SCADA historian and monitoring, occasionally displays invalid "zero" values and duplicate measurements are also present. Investigation has shown that the smart sensor itself records all measurement values correctly without invalid values or duplicates. The investigation was done by logging data directly from the smart sensor during operation. When again the same data through the SCADA system indicated zero values and duplicate readings. This suggests that while the sensor measurements are valid, anomalies may be introduced during data communication between the smart sensor and the SCADA system. The study was defined to focus on the communication between the smart sensor and the PLC, where the testbed isolates this interface.

5.2 Testbed Architecture

The testbed used for was created using available hardware from the collaborating company. Several configurations for the testbed were initially considered. These include a setup with two PLCs where one device would act as the OPC UA server and the other as the client. Because the PLC software limits the configurability of the OPC UA server, this approach was deemed unsuitable. The final chosen architecture used an emulated smart sensor capable of generating varying data load using a custom OPC UA server implementation. Detailed hardware specifications are presented in Table 3.

A Lenovo ThinkCentre M900 desktop was selected to host the emulated smart sensor with OPC UA server. This piece of hardware is hereafter referred to as PC 1. The OPC UA server was developed using Python as a programming language. Which enables configuration of update intervals, timing behaviour, data payload characteristics, and even made logging the updated values on server side simple.

PC 1 was connected to port 1 of a managed Ethernet switch. All network connections in the testbed were established using wired Ethernet links and all the Ethernet communication operated over 1 Gbps cable. The OPC UA client was connected to port 2 of the switch and was operated on a Siemens SIMATIC S7-1500 PLC. The type of PLC was selected due to its use within the client company's production. PLC being the most meaningful piece of hardware for performance evaluation. A 24 V DC power supply was used to provide sufficient power to the PLC.

A dedicated packet capture host (hereafter referred to as PC 2) was connected to port 3 of the managed switch. PC 2 was used to monitor network traffic between the OPC UA server and the PLC client. Traffic was passively captured using software called Tshark, which is a command-line based tool for packet capture and protocol analysis. To enable capturing the network traffic of OPC UA communication without affecting the traffic itself, port mirroring (Cisco SPAN) was configured on the managed switch. The PLC connected to port 2 was configured as the mirror source and port 3 as the mirror destination. This ensured that each Ethernet frame was captured only once.

A third piece of hardware (referred to as PC 3) was used exclusively for PLC configuration and logic programming. This was done on a Siemens SIMATIC Field PG M5 engineering laptop. The PLC was configured using Siemens Totally Integrated Automation Portal (TIA Portal). Using TIA Portal was necessary to create and deploy the OPC UA client logic for the PLC, since the PC 3 included needed software licenses. PC 3 was also used to monitor client behavior and count the number of observed anomalies. Table 3 presents the detailed hardware specifications of all devices used in the testbed. Figure 11 illustrates the actual testbed architecture with the OPC UA data flow marked as green between components.

Table 3. Hardware specifications used in testbed setup with relevant information shown.

	Role	**Model**	**CPU**	**RAM**	**Firmware**	**Base frequency**
PC 1	OPC UA Server	Lenovo ThinkCentre M900	Intel i5-6500T	8 GB	Windows 11 Pro 64-bit	2.5 GHz
PC 2	Packet Capturing	Lenovo ThinkCentre M900	Intel i5-6500T	8 GB	Windows 11 Pro 64-bit	2.5 GHz
PC 3	PLC Configuration	SIMATIC Field PG M5	Intel i7-6820EQ	32 GB	Windows 10 Enterprise 64-bit	2.8 GHz
PLC	OPC UA Client	SIMATIC S7-1500	1511C-1 PN	Program: 175 KB Data: 1 MB	V2.8	-
Switch	Managed Switch	WS-C3560Cx-12PC-S	-	-	15.2(7)E6	-

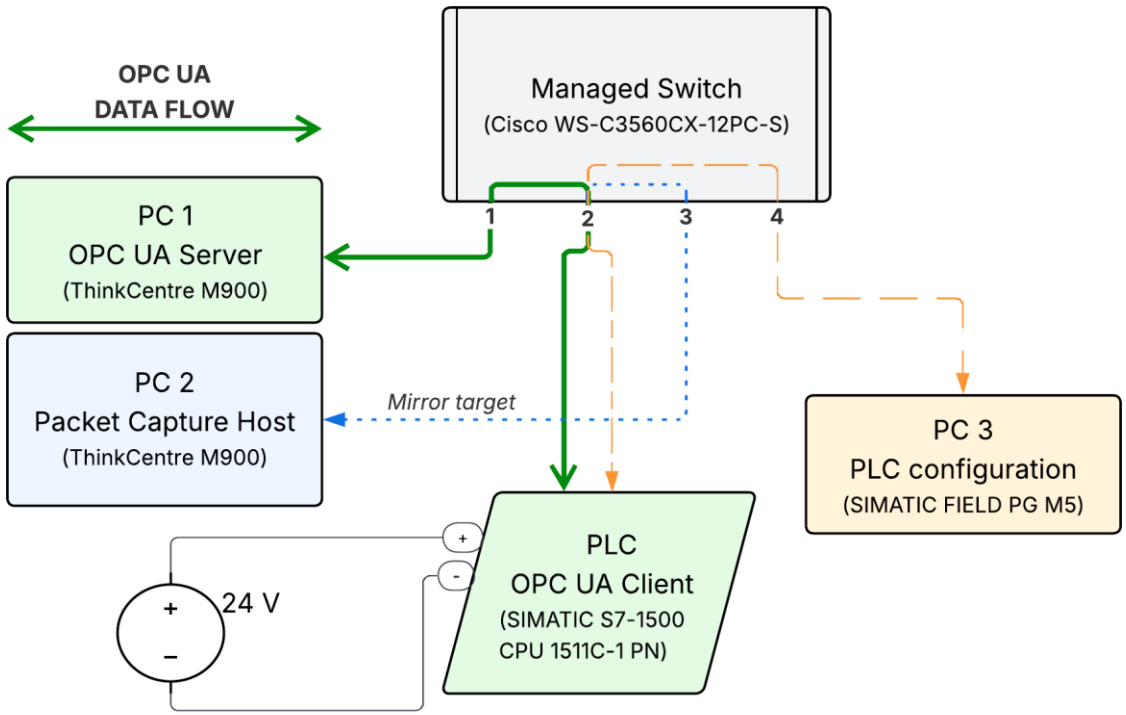

Figure 11. Testbed setup used in experiments. Data flow illustrated with green colour to clearly identify the OPC UA communication path used in the test runs. The communication happens between the client PLC and the server PC 1.

NTP time synchronization was configured between PC 1, PC 2 and PLC. PC 1 operated as the NTP server with the reference time for other devices in testbed. Although the reported performance metrics were derived from local timing measurements and packet capture analysis. Therefore, the testbed with the results ended up not relying on NTP clock alignment. The NTP time synchronization was still left operating for the testing. This was reasoned as in industrial automation environments operating under GMP requirements, time synchronization plays a critical role in ensuring consistent timestamping of alarms and archived process data for example. Siemens itself recommends NTP synchronization within SIMATIC systems. So implementing NTP synchronization reflects standard industrial practice, ensuring that the testbed setup aligns better with common requirements in GMP environments. [49]

5.2.1 OPC UA Server Configuration

The OPC UA server was created in mind to emulate the existing smart sensor in the collaborating company's production. The server was created with Python (version 3.11) using the asyncua library (v1.1.8). This kind of custom implementation was selected to provide full control over different variables of the server. These variables are for example update timing, payload size, number of load variables, and logging behaviour. The variation of these was needed for systematic performance

evaluation. The server was set to expose the nodes using standard port 4840. All variables used in test runs were set under an object named “Sensor” in the address space. The security policy was set to None, meaning that no encryption or message signing occurred. This was done to ensure that security features of OPC UA would not influence measurements.

The nodes in the server address space were structured in the following way. Two core variables “Seq” (Int32) and “TsServerMs” (Int64) were updated every cycle. The “Seq” indicated a sequence counter which was incremented per each sent packet. The sequence variable made detection of missed or duplicated updates simple at the client side. Server timestamp provided a reference for latency and jitter analysis on the server side using the logging feature. In addition to these core variables, two float type variables were implemented to simulate real process measurements. These “ID_value” and “OD_value” variables were generated using sinusoidal functions with predefined minimum and maximum bounds during each test run. Using this approach ensured deterministic and continuously varying values within known ranges. The ID and OD values allowed verification if invalid values were received by the client.

To create the variable communication load, configurable number of string variables were created under dedicated “Load” object. The total number of nodes in the address space was defined by the parameter TOTAL_LOAD_NODES which was derived from the limitations from the manuals of how many nodes in total the PLC could handle [61]. While ACTIVE_LOAD_NODES determined how many of these nodes were updated during each server cycle, which allowed control of how many writes were done per update cycle. To be noted, while all the load nodes were defined in the address space for each test, only the number of ACTIVE_LOAD_NODES were polled from the PLC side also. This resulted so that the empty nodes were left unread and would not cause extra load to the system.

Each updated load node received a randomly generated string with configurable length (STRING_LEN). By adjusting the number of active nodes and string length, the total amount of load could be systematically varied without modifying the address space structure itself. This design enabled so that creating totally new address space for each test run was not required, saving time and making testing more straightforward.

The server update cycle operated using scheduling mechanism defined by UPDATE_PERIOD_MS variable. This variable defined a set time interval between each update of the variables. At each scheduled update, the server incremented sequence counter, generated the timestamp and computed process values, and generated new random load nodes. The next update time was then calculated relative to the previously done update time rather than the actual completion time, thereby maintaining stable periodic behaviour. The execution duration of each cycle was measured and compared to the

configured update period. If the cycle execution exceeded the allocated period, a deadline miss indicator was recorded.

To enable the communication between the server and client, a nodeset export mechanism was required for client configuration. Relevant nodes were exported to an OPC UA XML nodeset file. This file contained namespace definitions, object hierarchy, node identifiers, and data type information for the client. The exported nodeset was then imported into the TIA portal environment for it to generate client variable mappings. As explained, the benefit of defining separate total and active nodes was that the nodeset export was only required during initial configuration, as the structural model remained constant, and the number of nodes could be varied along with the string length from server side across experiments.

To support the analysis, a CSV-based logging mechanism was integrated into the server. One logging was written per update cycle containing information for server analysis. The logging feature was line-buffered to minimize delay in data writing. These server logs enabled verification of server's timing stability and allowed comparison with network packet captures obtained by packet capture host if needed for the analysis. Which helped to better separate between server's and client's performance characteristics under varying loads.

Before each measurement run, the server executed a set warmup period. The warmup period was done mainly to ensure proper connection and made possible so that the client could be executed before any value updates were present. The warmup was required because the connection would not work if the client CPU was turned on before the server was running. Together with warmup the server was set to run a pre-defined time before automatically exiting the run. Automatically controlled runtime ensured consistent measurements across all configurations.

5.2.2 PLC Client Implementation

The OPC UA client was implemented using Siemens TIA Portal (version V15.1 Update 4). A single OPC UA client interface was created to establish the connection to the server. The client was configured to also use port 4840 for OPC UA communication, matching the server configuration. The session timeout and monitoring interval parameters were set to 6000 ms and 100 ms, respectively. As with the server, all security settings were disabled by configuring both the security mode and policy to "No Security," and user authentication to "Guest." As discussed in the server configuration section, this ensured that communication performance could be analysed without the influence of encryption or authentication overhead.

After the interface configuration, five separate ReadLists were created to enable parallel read requests, which are supported by the PLC platform [61]. The XML nodeset file generated by the server was imported into the TIA Portal environment to provide the client with the predefined namespace and node definitions. For each test run, the number of nodes defined by the server's ACTIVE_LOAD_NODES parameter was allocated to the ReadLists in a predefined and consistent manner. When the number of active nodes changed between runs, only the allocation of nodes to the ReadLists needed to be modified. Again, leading to no additional XML imports. For unknown reason empty ReadLists are not permitted in TIA Portal. So, instead of deleting unused ReadLists, which would have required redefining the logic for each configuration, a single unused load node was assigned to them. This preserved the structure of the implementation without affecting measurements, as the corresponding node was neither updated by the server nor actively evaluated by the client.

Once the OPC UA client interface was configured from the settings of TIA portal, the actual polling logic was implemented. The logic was created directly in the OB1 block using Ladder Diagram (LAD) language. A step counter was introduced at the beginning of the logic to manage the sequential initialization of the OPC UA connection. The step counter ensured that each stage of the connection process including address space defining completed successfully before proceeding to the next step. For the used TIA portal version, establishing an OPC UA client connection required four separate function blocks executed in a specific order. The step counter advanced only when the "Done" output of the current block was set to true.

The first step involved generating a rising-edge pulse to the OPC_UA_Connect function block, which created the connection session with the OPC UA server. Once the "Done" output of this block was true, the step counter incremented to prevent further connection requests and proceeding to the next step. The executed the OPC_UA_NamespaceGetIndexList (NGIL) function block. NGIL retrieves the namespace indices of the connected server. In this setup it was done based on the imported XML file. After successful execution of the NGIL block, the step counter advanced to activate the OPC_UA_NodeGetHandleList (NGHL) function block. The NGHL block registers the OPC UA node identifiers that are to be read from the server. Since up to five ReadLists were used, the NGIL and NGHL function blocks were executed separately for each ReadList at the beginning of each test run. Initialization was performed for each test run during the warmup period, ensuring stable working connection before the actual testing. [61]

After successful completion of the connection phase, the polling of read requests were initiated using the OPC_UA_ReadList function block. This block initiated the read jobs for each ReadLists. To ensure that only the read requests were executed during normal operation of the testrun, the step

counter was set to a fixed state once initialization was complete. Preventing the connection setup sequence from being executed repeatedly.

The number of parallel read requests was varied by enabling or disabling specific read jobs using their EN input. Disabled read jobs remained configured but inactive, allowing flexible modification of parallelism without restructuring the program. All active read jobs were triggered using the same REQ pulse. To avoid overlapping read requests, the “Busy” output of each read job was set so that new reads were issued only when all active read jobs had completed their previous cycle. Finally, after each full read cycle, a separate evaluation pulse was generated to initiate the gathering of client metrics for each run. The detailed explanation of this is described in the Measurements and Metrics section. [61]

There are few different ways of generating the timing for polling the server values. In the used testbed a TON timer was used to achieve this. Ton was set before the REQ pulse generation to the read jobs. The TON timer therefore defined fixed minimum polling interval of the client. Unlike the server’s UPDATE_PERIOD_MS mechanism working as an asynchronous update timing where the actual processing time was taken into an account for the next update, the TON timer introduced a synchronous reading with fixed delay before each poll on top of the client’s processing time. Meaning that the effective polling interval consisted of the configured TON delay plus the execution time of the read cycle. While mentioned that there are other and similar ways to the server’s asynchronous in the PLC logic. This implementation was still selected for its simplicity and reliability. Although it does not guarantee a strictly constant polling period, it provides consistent behaviour where the client processing delay is clearly seen. The implications of this timing approach are further discussed in the discussion section of the study.

5.3 Measured Metrics

To evaluate the performance of the OPC UA communication, the performance metrics were gathered from three different sections of the communication. These metrics were therefore grouped into three categories based on from which section of the communication it was gathered from. The sections are server metrics, network metrics, and client metrics. Used approach is believed to help identify which part of the system is most likely responsible for any detected anomalies.

5.3.1 Server Metrics

As mentioned in the working assumptions, the polling interval was considered a potential root cause of duplicate values. Because of this, the server metrics were designed primarily to verify that the server

completed each update cycle withing the given update time and operated otherwise reliably so that the server itself could be excluded from causing the any occurring anomalies.

For this purpose, latency and jitter proxies were defined. The term proxy refers here to an observable approximation of internal server processing delay and its variability. An important distinction to make is that the latency proxy does not represent network latency, instead it represents the server-side internal TTC of one update cycle.

The server latency proxy is measured using the update_duration_ms variable, which is logged for each update cycle. Update duration variable represents the actual server handling time required to update the nodes during one cycle.

As described in the server architecture, the system operates dynamically:

- If update_duration_ms < UPDATE_PERIOD_MS, the server waits until the next scheduled update period.
- If update_duration_ms ≥ UPDATE_PERIOD_MS, the server cannot complete the update within the target cycle time. In such cases, a flag is written to the log file indicating that the delay was caused by the server processing time exceeding the configured update period.

Thus, the latency proxy directly represents the internal processing duration per cycle.

While the server latency proxy answers for how long the server takes to update values in one cycle. The server jitter proxy answers to how closely does the actual update interval follow the configured update period. The jitter proxy represents the variation of the full server update cycle during operation. Whereas the latency proxy is directly calculated inside the server code by subtracting timestamps before and after node updates, the jitter is derived from the timestamps of consecutive update cycles using Equation (1):

$$jitter_i = (t_i - t_{i-1}) - T_{target} \quad (1)$$

Where:

t_i = ts_server_ms[i] (timestamp of current update cycle)

T_{target} = UPDATE_PERIOD_MS

A positive jitter value indicates that the update occurred later than the configured update period. A negative jitter value indicates that the update occurred earlier than the configured update period.

Together, the server latency and jitter proxies provide insight into the internal timing behaviour of the server under different load conditions. These help to determine whether observed communication anomalies originate from server's timing behaviour or from other parts of the system.

5.3.2 Network Metrics

Unlike the application-level metrics collected from the server and client, the network (wire-level) metrics were obtained from packet captures recorded on a dedicated capture host using Tshark (v4.6.3). Full packet traffic was captured during each test run and the relevant metrics were derived from the resulting Tshark pcapng packet capture files collected at the mirrored client's network port. The packet capture files were converted into tsv format to enable structured processing and metric calculation. Tshark was used to reconstruct OPC UA MSG service messages exchanged between the server and the client.

An important clarification must be made regarding the network measurements. Tshark captures raw network packet traffic [62]. Resulting that the analysed OPC UA messages are reconstructed from this raw network packet traffic, rather than directly observed at the application layer. Under higher load conditions, some test runs exhibited irregular behaviour in the reconstructed OPC UA message files. For example only one-directional MSG packets were present in some of the reconstructed data, or significant inconsistencies were observed in the reconstructed OPC UA message streams, while the corresponding TCP segment captures showed little or no observable difference between runs.

Due to these inconsistencies in reconstruction of the OPC UA messages, inconsistent test runs were excluded from the result results. The distinction of the inconsistencies being due to the network traffic reconstruction was justified by separately analysing the corresponding TCP streams reconstructed from the same packet captures.

The network metrics are clearly presented in the Table 4 with the definition and short explanation to what it measures for ease of reading and to help evaluating the meaning of the results.

Table 4. Metrics used for performance analysis derived from wire-level packet captures.

Metric	Definition	What it Measures
RTT	Round-trip time of the first MSG response packet.	End-to-end request/response delay
Jitter	RTT variability (IQR)	Stability of the cyclic delay
Poll Drift	Polling interval excess over client TON timer	Deviation from the configured poll period.

The RTT measures the time between the first OPC UA request (*MSG*) from the client and the arrival of the corresponding server response (*MSG*), represents the communication latency in one polling cycle for one message. Enabling to show time required for the wire request-response exchange while largely excluding all client processing from the measured time. To get meaningful plots, only values satisfying $0 < RTT_i \leq 5$ s were accepted. This filtering was done to remove mismatched request-response pairs and extreme outliers when creating the plots from the packet capture files.

The jitter here represents its usual meaning, the variability of RTT values within a single test run. The jitter was calculated using the interquartile range (IQR) providing a resistance of outliers affecting the calculated metrics. Jitter here shows the variability in response time in each cycle. In the context of this study, jitter may indicate lower timing stability in all the parts of the communication combined.

To find out how big of an effect the client processing had in the possible timing mismatches between the communication participants, the drift in polling interval was calculated for each test run. As described earlier, client polling was created using synchronous TON timer in the PLC logic. The timer waits for the set period after each full successful poll. Leading to the behaviour explained that the execution and handling time forming the client processing time in the polling operation is added on top of the TON timer.

To show the timing drift quantitatively, the median of the excess part over the configured TON period was calculated as poll drift. Clearly showing how much the actual polling cycle differed from the intended TON timer under varied load conditions.

The calculation procedure consisted of the following steps:

1. The time difference between consecutive client requests was computed.

2. A *cluster gap* threshold was set to determine which consecutive requests belong to the same polling round consisting from one to five read jobs.
3. When the time difference exceeded the cluster gap threshold, a new polling round was identified.

For most test runs, a cluster gap between 0.2 and 0.5 seconds produced reliable results. If the cluster gap was set too low, it resulted in negative poll drift values. After identifying the first timestamp of each polling round, the poll drift was calculated for each run using Equation (2):

$$Poll\ Drift_{median} = \left(t_{round\ start_{j+1}} - t_{round\ start_j}\right) \times 1000 - TON_{test\ run} \quad (2)$$

Equation 2 calculates the median difference between consecutive polling rounds converted to milliseconds and subtracts the configured TON period of the respective test run from this.

5.3.3 Client Metrics

All client metrics were recorded and collected directly from the PLC logic using internal counters that incremened whenever a predefined event occurs. As described in the client implementation, after each completed read cycle of the first read job, a boolean tag is set to TRUE which generates a rising-edge pulse. The pulse then triggers evaluation logic. The evaluation is intentionally triggered immediately after the first read job. The first read job reading the first ReadList always contains the monitored variables (Seq, ServerTsMs, ID_value, and OD_value), whereas additional ReadLists used in higher-load scenarios contain only the synthetic load data. Since the correctness of the load data itself is not relevant to this study, detection of duplicate values, missing values, and invalid values are performed using only the first ReadList.

When the evaluation is triggered, the current sequence number and server timestamp are stored in temporary variables. These variables are then compared with their previously received values. If the current sequence number equals the previous sequence number, the duplicate counter (DupCount) is incremented. Similarly, if the current server timestamp equals the previous timestamp, a stale timestamp counter (StaleTsCount) is incremented. To prevent false detections right after warmup, a validation condition was created to ensure that the sequence number is not zero before evaluation pulse triggering.

The validity of the controlled process values was also monitored allowing the detection of invalid values including unexpected zero values observed in production system. As defined in the server architecture, the ID and OD values were set to stay within predefined minimum and maximum limits.

Both of these values are checked after each polling cycle to verify that they remain within the set range. If either value is something other than within defined limits and the sequence number is not zero indicating warmup. Then an invalid value counter (InvalidCount) is incremented.

Lastly, the detection of missed packets was implemented. Missed packets may occur when the server updates values faster than the client can read them. Missed packets are identified by also observing the sequence counter in each message. The last received sequence number is first computed and then (LastSeq) compared with the newly received sequence number (Seq). If the received sequence number exceeds the expected value (LastSeq + 1), one or more packets have been missed. In such cases, a missed packet counter (MissCount) is incremented accordingly. These missed packets are later referred as missed values.

Although the client metrics do not provide precise timing measurements comparable to RTT or poll drift, they serve as behavioural validation indicators that support the timing mismatch working assumption.

5.4 Experimental Procedure

To ensure repeatability and comparability of results, a controlled testing procedure was used consistently across all test runs. Before each run the following sequence was done. First, the polling interval and server update cycle was defined by setting the UPDATE_PERIOD_MS parameter on the server side and the TON timer on the client side. Next, the number of active nodes and the payload size were configured on the server. Done by adjusting the number of enabled load nodes and the string length. Finally, the number of parallel read jobs was defined on the client side and then distributing the active load nodes across the used number of ReadLists.

Each test configuration was executed twice. Initially, the plan was to perform three or more repetitions per configuration, but after first few runs, observing highly consistent results between repetitions, two runs per configuration was enough to provide reliable results and keep the testing time manageable. Then the actual test runs were executed following the same sequence of operation. First, the packet capture host was started to monitor and record network traffic between the OPC UA client and server. Initiating packet capture before any other component in the testbed ensured that all messages in communication were recorded. Then the OPC UA server was started. Once the server was operational it initiated the fixed 20 second warmup period. During this period the client CPU was started by setting it to RUN mode. Doing it in mentioned order prevented the missed initial updated by ensuring that the client polling cycle was active before the server began updating its values.

Alternative starting sequences were tested during early experimentation. These testings occasionally resulted in missed initial server updates. But the adopted procedure provided least disturbances for the communication startup.

The different test runs were designed to address the defined research questions individually. Table 5 provides an overview of each test run with parameters used. Each test run was assigned a unique identifier (ID) to simplify analysis. The runs are also grouped in Table 5 according to the specific research question they were intended to address.

Table 5. Experimental test configurations with separation of groups to answer each specific research question. This includes verification runs to verify special scenarios.

Group	ID	ReadLists	Load Nodes per ReadList	Total Load Nodes	String Length	Polling (ms)	Server Update (ms)	Runtime (min)
Baseline	BL-1	1	20	20	10	1000	1000	4
Baseline	BL-2	1	200	200	100	1000	1000	4
Parallelism	P-1	1	200	200	50	1000	1000	4
Parallelism	P-2	2	$100 + 100$	200	50	1000	1000	4
Parallelism	P-3	3	$67 + 67 + 66$	200	50	1000	1000	4
Parallelism	P-4	4	50×4	200	50	1000	1000	4
Parallelism	P-5	5	40×5	200	50	1000	1000	4
Payload/Nodes	I-1	5	40×5	200	5	1000	1450	5
Payload/Nodes	I-2	5	40×5	200	254	1000	1450	5
Payload/Nodes	I-3	5	$200 \times 4 + 150$	950	5	1000	1450	5
Payload/Nodes	I-4	5	$200 \times 4 + 150$	950	254	1000	1450	5
Timing	L-1	5	$200 + 25 \times 4$	300	100	250	900	5
Timing	L-2	5	$200 + 25 \times 4$	300	100	125	900	5
Timing	L-3	5	$200 + 25 \times 4$	300	100	10	650	5
Verification	T-1	5	40×5	200	50	1000	1450	5
Verification	LL-1	5	$296 + 150 \times 3 + 200$	946	250	10	2100	5

After few initial test runs, the runtime was extended from four minutes to five minutes. This adjustment was made to ensure enough samples, as longer update intervals resulted in fewer client updates. The runtime change not affect the validity of the results, as the calculated metrics are independent of total runtime. The runtime primarily determines the total number of packets per run.

The baseline group in Table 5 was defined as the control group against other test configurations. The parallelism test group was designed to illustrate the effect of parallel read jobs while keeping all other parameters constant. The payload and node test group examined how changes in payload size and the number of nodes influence communication behaviour. The timing group focused specifically on client-side behaviour under modified polling and server update intervals. A small number of verification runs were also conducted to validate specific polling behaviours; these are discussed more in the Results and Discussion sections.

Once the predefined runtime was reached, the server was automatically terminated. Network traffic capture was then manually stopped on the packet capture host using a keyboard interrupt to ensure complete recording of the communication session. Client-side metrics were recorded manually by capturing screenshots of the PLC interface. The wanted values from these screenshots were then manually taken for further analysis. This manual approach for the client metrics was considered sufficient due to the small number of client counters and the limited number of test runs.

6 Results

The results begin with a comparison of different scenarios featuring varying total system load, establishing a baseline for the measured metrics. Following this, the effects of individual variations according to the research questions are presented.

6.1 Comparison of System Total Load Scenarios

The purpose of Figure 12 is to illustrate the overall differences between high-load and low-load scenarios. The difference observed here provides a reference scale for the magnitude for possible total variation across test runs.

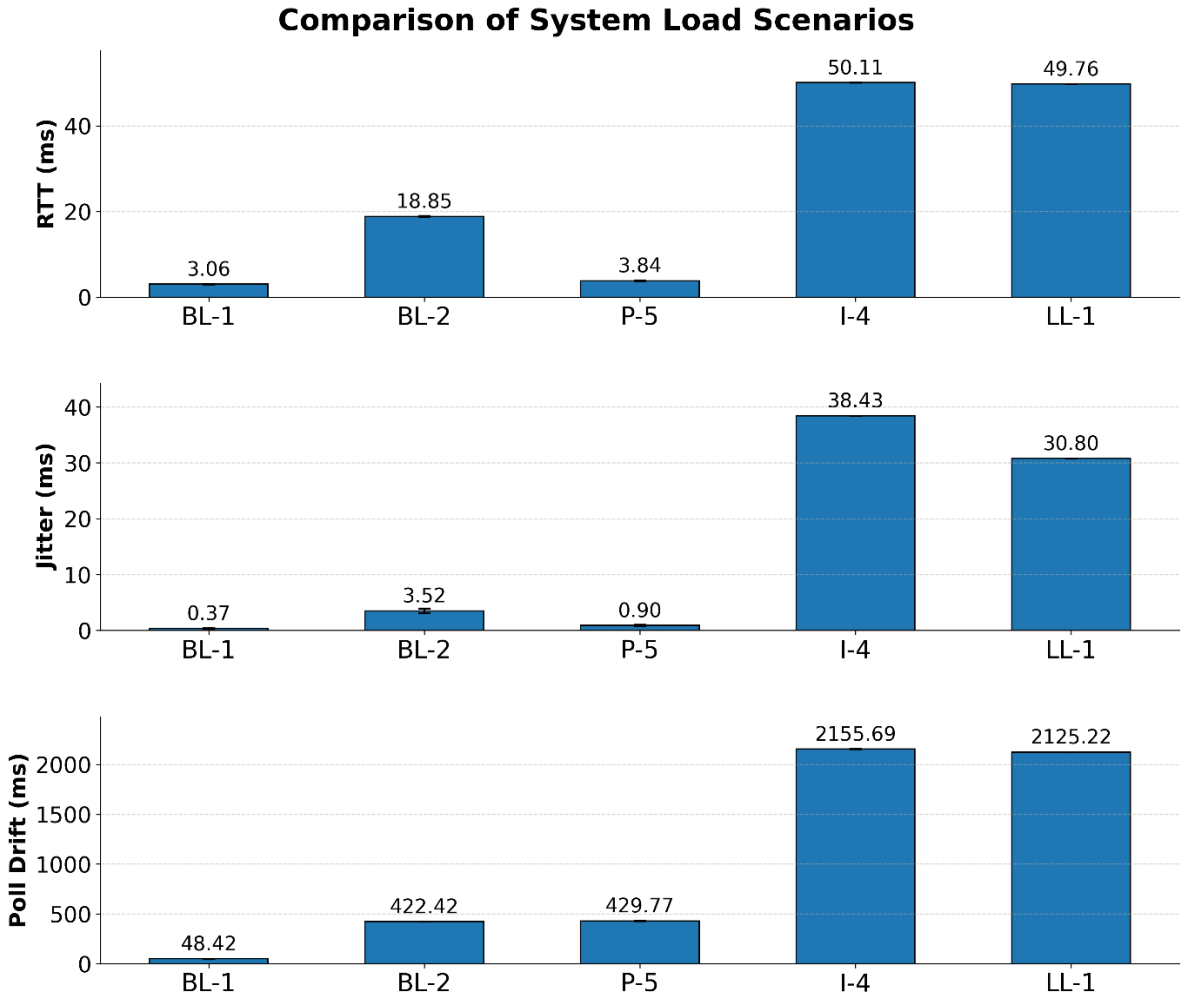

Figure 12. Comparison of System Load Scenarios to provide reference scale of magnitude between test runs for each measured performance metric.

For all presented data in results, RTT and poll drift are the median values over two test runs. Each bar includes error indicator on top showing the minimum and maximum range from both runs. As previously defined, jitter is calculated as the IQR of the RTT samples also as the median over the two test runs. Observed in Figure 12, the measurements show consistent behaviour with minimal error

bars. Explained consistency was seen across most runs. Only few cases showing noticeable difference from this.

Bar charts were selected for data presentation due to them being easy to compare one to other with different test cases. Since the values are derived from individual configurations rather than being continuous measurements, this format was considered more appropriate compared to some form of line chart for example and is used consistently throughout the results.

In Figure 12, BL-1, BL-2, and P-5 correspond to low demand scenarios. Here BL-1 is with the lowers overall load among all experiments. I-4 and LL-1 operate close to the set hardware limits by the PLC. From Figure 12 alone, a trend is highlighted: increasing overall load leads to a significant increase in all of the performance metrics.

A recurring observation across all configurations in test runs is that poll drift increases with system demand and consistently exceeds RTT in every scenario. Because the Figure 12 experimental setups differ substantially in almost all the parameters, direct comparison between them is not reliable. Therefore, a more detailed analysis of individual factors is provided in the following sections, where each load type is examined with reduced influence from other variables.

6.2 Number of Elements

As previously discussed, the number of elements is considered a key factor influencing communication load. In the Siemens PLC software, the entries in a ReadList are referred to as elements, whereas in OPC UA terminology they correspond to variable nodes in the address space. In this thesis, the terms element and node are used interchangeably when referring to individual variables accessed through the ReadList. Since the PLC used in this study limits the maximum number of nodes per read job, achieving a wide range of load conditions required the reading of multiple separate ReadLists. The results from varying the number of nodes, are divided into three comparison categories based on the total number of nodes. To further clarify the terminology, ReadList is the actual list containing the elements or nodes, as the read job is the function block making the read polls in the PLC logic.

In Figure 13, BL-1 and BL-2 are first compared. Both configurations use a single read job, with BL-2 containing ten times more nodes and a slightly larger payload size. The number displayed under each bar in the plot indicates the total node count for that test run. What can be clearly observed is that even with a single read job, all evaluated metrics increase as the node count rises. The initial comparison shows a good reference as the total number of nodes in BL-1 is relatively small compared to the

studied industrial scenario analysed in this study. For BL-2, the number of nodes was increased to closely reflect load conditions in production from OPC UA traffic only.

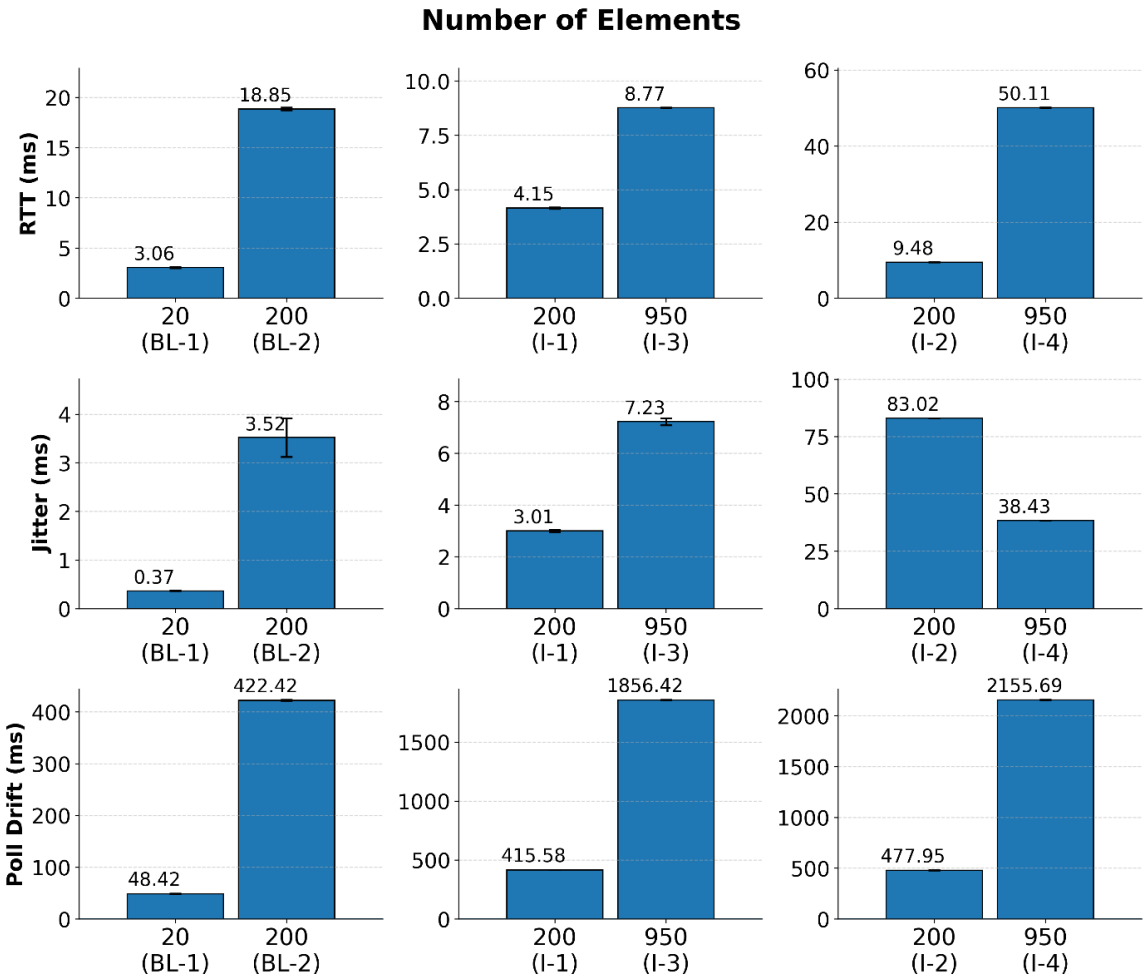

Figure 13. Results from number of elements comparison between different test runs. Large number of nodes clearly increase poll drift and RTT.

The comparisons between I-1 and I-3, as well as between I-2 and I-4, form the actual analysis in Figure 13 regarding the impact of node count on OPC UA communication performance. To extend the range of total nodes, five parallel read jobs were used in these configurations. I-1 and I-3 operate with a small payload size, resulting in moderate overall system utilisation even as the node count approaches the PLC defined maximum. In contrast, I-2 and I-4 use a larger payload size, where the I-4 operates close to the PLC data transfer capacity limits.

A consistent trend is again observed from the I-runs: increasing the node count leads to higher RTT and significantly increases polling drift. However, a notable exception appears in the high load comparison between I-2 and I-4. While RTT and polling drift follow the expected upward trend, jitter, decreases markably in the highest load configuration.

6.3 Polling Interval Behaviour

The polling interval was specifically examined to evaluate its effect on the performance metrics and to verify the underlying assumptions regarding duplicates and missed values in polling-based client-server communication.

In Figure 14 (left) three configurations with decreasing polling intervals are compared under similar system conditions using five read jobs. The L-3 configuration is highlighted separately due to its shorter server update period. Shorter server update in L-3 does not compromise comparability within Figure 14. As the server update period primarily affects client timing behaviour rather than the overall system demand.

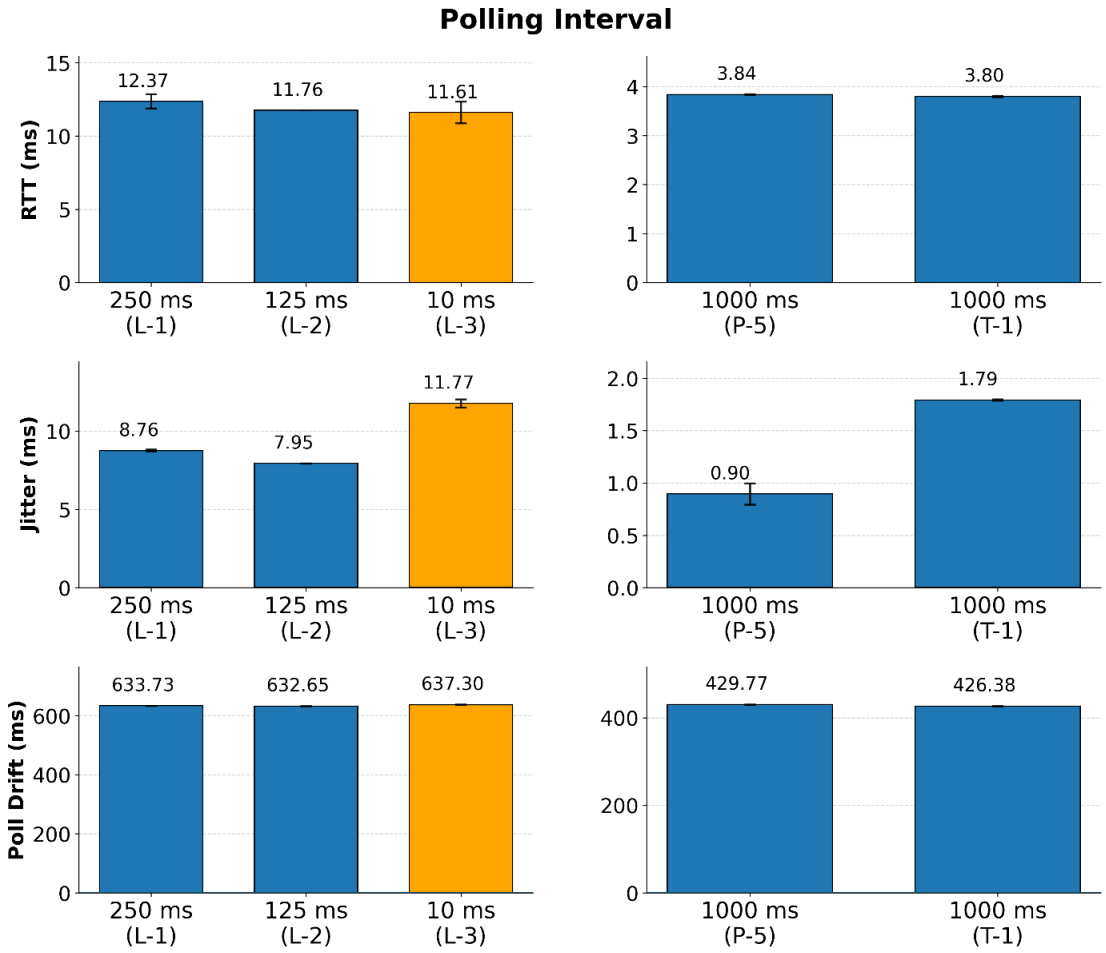

Figure 14. Polling interval effect comparison. L-3 marked as orange to separate its different server update cycle.

When comparing the L-runs, the polling interval itself has small impact on the primary performance metrics even when reduced substantially in L-2 and L-3. Clear example of this is when halving the polling interval from L-1 to L-2 results in only a minimal change in polling drift. The effect of timing

mismatch causing duplicates and missed values is on the other hand well observed from the client-side metrics. In L-1, the experiments showed an average of 8 duplicates and 2 missed values. Where in contrast L-2 showed no missed values and an average of 62 duplicates. This behaviour aligns with the relatively constant polling drift observed between these two configurations, where it affects deterministically to the number of duplicates and missed values when taken the timer itself into an account.

To better verify the effect of polling timing, the server update period was reduced in L-3 and the client polling interval was set to low 10 ms. Resulting an average of 2 duplicates and 1 missed value. Indicating that the polling timing shows little to no uncertainty even at very short intervals. Interestingly, in the second measurement cycle of L-3, neither duplicates nor missed values were observed. This suggests that higher jitter would contribute to variation between repeated runs, which is relatively intuitive.

On the right side of Figure 14 test runs P-5 and T-1 were compared to validate polling timing behaviour. To validate timing, the runs were chosen so that they are identical except that, in T-1, the server update period was adjusted to 1450 ms. This server update period determined experimentally by gradually increasing the update interval while monitoring duplicates and missed values. At 1450 ms, the results showed only 4 duplicates and no missed values. When compared to 71 missed values in P-5, it shows that the timing mismatch of server and client was mostly fixed at 1450 ms server update period in P-5 and T-1 runs. The average number of duplicates and missed values for each polling configuration is presented in Table 6. The table is shown to illustrate the effect of timing adjustments on the number of duplicates and missed value.

Table 6. Timing effect on client-side behaviour.

	L-1	**L-2**	**L-3**	**P-5**	**T-1**
Duplicates	8	62	2	0	4
Missed Values	2	0	1	72	0

What can be concluded from these restuls, is that when the polling interval is relatively short compared to the server update period, duplicates tend to occur and they are increased as load increases. When again the server update period is shorter relative to the polling interval, missed values are observed, being further increased with communication load. It should be noted that for the missed values the actual number of them may higher than reported in metrics. Due to implemented logic recording missed values only when a value was missed within a cycle, not how many consecutive

values were skipped. Meaning that in some cases, the server may have progressed through multiple updates before the client retrieved a new value.

6.4 Parallel Read Jobs

Five specific test runs were conducted to evaluate the impact of parallel read jobs on the OPC UA performance. These runs being P-1 to P-5, which are identical in overall all configurations except that the 200 load nodes are distributed evenly for all the ReadLists used. As previously presented in Table 5, This approach was designed to maintain the same total load in each P-run while isolating the effect of parallel read job reads on communication. The P-1 uses 1 read job where P-2 uses 2 and so on. The results of these experiments are shown in Figure 15.

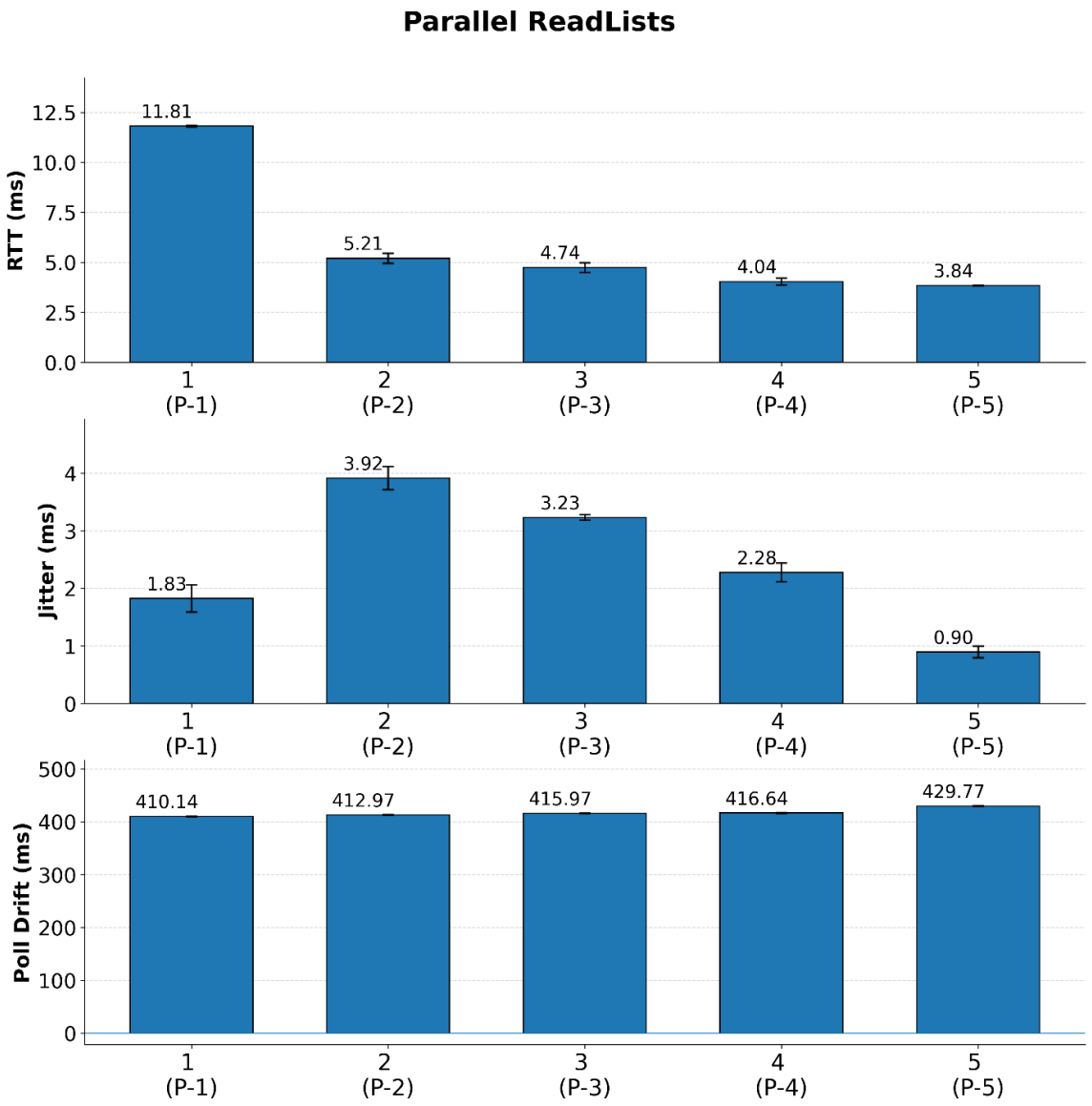

Figure 15. Comparison of the parallel read jobs. The poll drift staying constant where as RTT and jitter show specific variation related to the number of read jobs.

Figure 15 shows that polling drift remains relatively stable across all configurations these runs. Only a slight increase is seen each time an additional ReadList is introduced. RTT and jitter on the other hand have more significant influence from the number of read jobs.

With a single read job in P-1, RTT stays higher while jitter remains low relative to other runs. This finding on the plot can be also observed when analysing the recorded packet captures directly. Runs which have total loads and multiple read jobs, the client processes each read job as an individual OPC UA message. This is even done in a very systematic manner as in the I-4 recordings a consistent pattern of five polling request messages were always followed with five corresponding response messages occurring in repeating sequence.

This sequential pattern becomes less deterministic as the total load decreases. Packet captures from lower load runs show that requests and corresponding responses no longer follow a perfectly cyclic order. The change in behaviour is possibly also reflected in Figure 15. When distributing the same payload across multiple OPC UA messages, reduces RTT possibly because of smaller individual message sizes, while jitter increases with more messages are transferred causing varied timing in read request sequences.

Increasing the number of parallel read jobs even further, gradually balances this effect as seen in Figure 15. With five read jobs in P-5, the communication has the lowest RTT and jitter among all P-runs but also the highest polling drift.

6.5 Size of the Payload

Payload size impact on OPC UA performance was lastly analysed. In this context, payload size refers to the string length of each node, which are generated as the communication load by the server. The longer the string, the larger the total transmitted payload per node. To isolate only the payload size for comparison, the total number of nodes was kept constant within each pair of runs shown in Figure 16.

The left comparison in Figure 16 (I-1, I-2) represents the lower total load situation with 200 total nodes on each. On the right the high load cases are shown with 950 total nodes (I-3, I-4). The string lengths used in the experiments are indicated below each bar. The string lengths were set to either 5 characters or 254 characters.

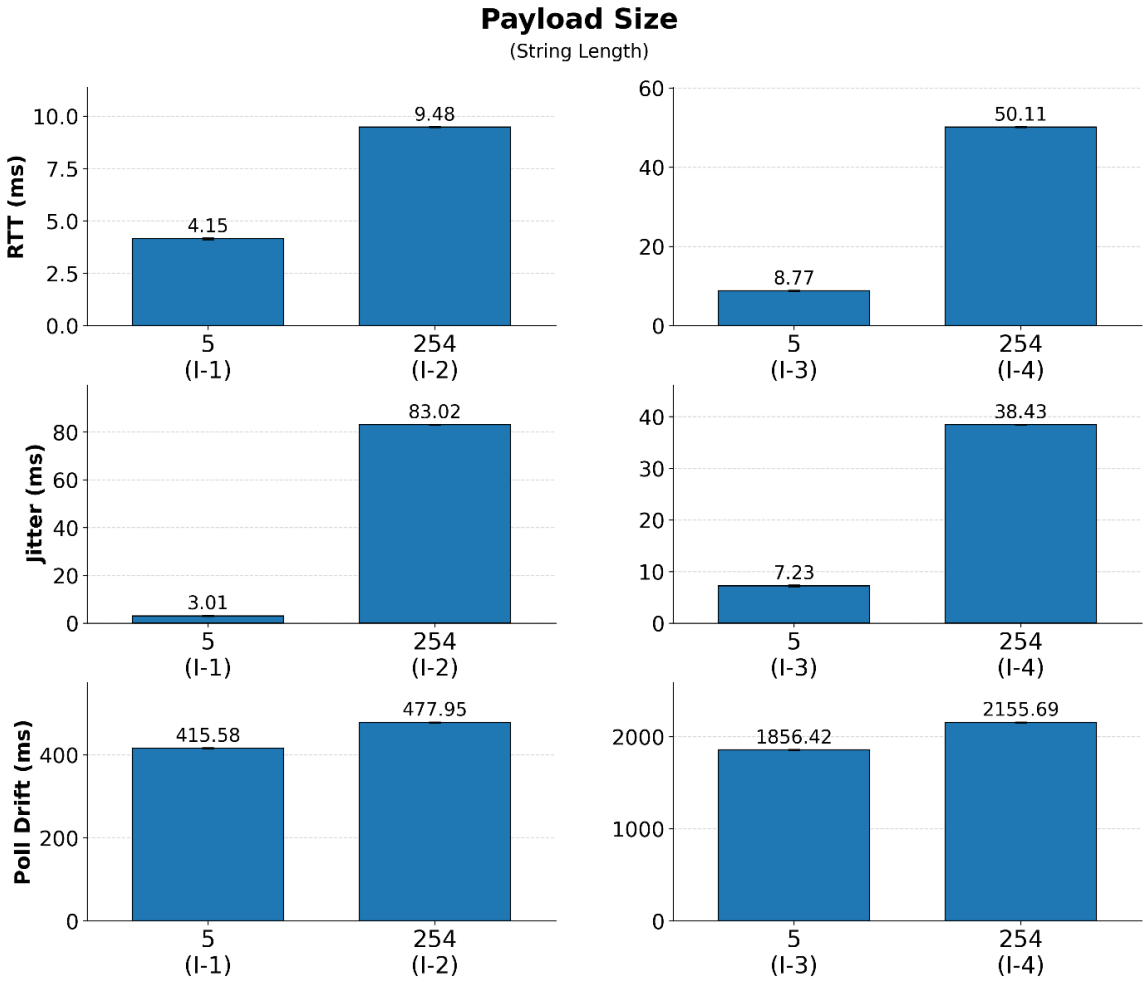

Figure 16. Performance effect of different payload sizes by varying string length indicated under each bar. Jitter shows to be highly dependent on the payload size.

Interesting result is revealed from Figure 16. In earlier run setups focusing on the impact of total node count, increasing the number of nodes led to a substantial rise in polling drift. Here, when the payload size is increased significantly while keeping the node count constant, the increase in polling drift is considerably less pronounced as seen in Figure 16.

The larger payloads are then clearly evident in RTT and jitter metrics. Both of them increase substantially with payload size. It should be noted however that payload size and node count are still inherently related. This comes from as increasing the number of nodes also increases the total transmitted data also contributing to the payload size. Again, when short string lengths (smaller payload) are used the total communication load does not grow as dramatically even if the number of nodes is increased.

7 Discussion

The discussion section tries to address each of the research questions. The objective is to identify the factors that most significantly influence the overall performance of OPC UA communication in the testbed setup. And, to some extent explain the observed anomalies.

A consistent and documented finding across all experiments is that increasing overall system load leads to a clear rise in all measured metrics. This observation serves as a baseline for the analysis. The limitations of the study and results relation to existing OPC UA performance research are also discussed.

7.1 Increasing the Payload

From the results a clear correlation between increasing total payload and rising of performance metrics can be observed. This relationship is rather intuitive. The greater the amount of transmitted data, the longer the communication processing requires and the higher the potential for timing variation. This assumption formed from the initial working assumptions and is supported by the results of the study.

However, notable differences emerge depending on which type of load is increased. As shown in Figure 13, polling drift increased approximately ninefold between BL-1 and BL-2. In the other node-count comparisons, the increase in polling drift between low-node and high-node configurations was approximately fivefold on average. The more pronounced increase in the BL-runs comparison is reasonable as BL-1 operated with only 20 nodes, resulting in a very low overall demand where even the RTT was nearly negligible.

In contrast, configurations with 200 nodes (BL-2, I-1, and I-2) showed polling drift values within less than 100 ms of each other, despite differences in payload size. Notably, BL-2 used less than half the string length of I-1 and I-2 while still the polling drift remained relatively similar.

Further evidence is provided in Figure 16, where payload size is varied while keeping the number of nodes constant. In these comparisons, polling drift remains relatively stable even when the string length is increased significantly. Suggesting that, the primary factor influencing polling drift is the number of nodes rather than the payload size itself within this study. This finding is consistent with the operation of the Siemens PLC devices, where each node must be individually defined, registered, and processed for every read job [61]. So it can be undestood that regardless of the actual payload size in a node, it must still undergo the same handling procedure.

What can be concluded from these is that the client processing time contributes directly to the PLC cycle time which is then seen in polling drift variation. To optimize improve consistency the total load of the communication lowering the total number of nodes may have the single highest effect to reduce overall communication load, which can cause timing delays.

The behaviour of RTT and jitter which is observed in Figures 16 and 19, appears to be more closely related to the overall transmitted data amount than the number of nodes alone. Increasing either the number of nodes or the payload size leads to higher RTT and jitter consistently in most configurations. Aligning with the general principle that adding nodes inherently increases the total transmitted data, seen as the increase in total communication load. To better separate the relative impact of node count and payload size on RTT and jitter, further experiments with stricter isolation of these variables would be required.

7.2 Behaviour of Multiple Read Jobs

As previously stated in the results, the number of parallel read jobs had small direct impact on polling drift when compared to other factors such as node count. A small increase in polling drift was observed as additional ReadLists were introduced. But this small increase is considered reasonable, because with multiple read jobs, the client must manage multiple concurrent read operations. Leading to increases in scheduling complexity and resource usage of the communication which are seen as the small increase in polling drift.

Siemens has clearly taken its hardware computational limits in account and restricts the maximum number of parallel read job for each CPU type of its PLC product line [61]. It is reasonable to assume that the mentioned limitation is intended to prevent excessive consumption of computational resources within the PLC. With the CPU used in the testbed, the maximum parallel read jobs were set to five.

An interesting observation was made in testing relating to the parallelism of the read jobs. The vendor-defined parallelism appears to refer to the number of read jobs that can simultaneously be in a “BUSY” state, rather than true parallel processing of separate reads. Explained conclusion came from the observation that while in the P-runs the load was distributed along multiple read jobs, the poll drift increased slightly. This could be the increase due to some sort of buffering of the read jobs instead, if actual parallelism was present, the poll drift would be expected to actually decrease. This is because then each read job would process less nodes each in parallel, resulting in faster client processing.

Another indicator for this definition of parallelism was already presented in the results. From the packet captures of high load runs such as I-4 and LL-1, the client transmitted five OPC UA messages

in sequence, after which the server returned five corresponding responses repeating the same pattern consistently throughout the run. These five OPC UA messages are understood as representing the five read jobs. While with low load conditions, this behaviour was not present, and the request-response message sequence appeared irregular from the packet capture data. While the second indicator does not directly prove the parallelism being as defined in the first place, it gives indicators to support the conclusion made.

Jitter initially increases when multiple read jobs are introduced and then decreases again as the maximum number of parallel jobs is approached. A similar effect can be observed in Figure 13, where jitter decreases when the total payload increases significantly from I-2 to I-4. One possible explanation is the just explained firmware scheduling mechanism of the PLC handling multiple parallel read jobs. Although additional experiments would be required to confirm this behaviour especially in the I-2 and I-4 case instead of just being a result of corrupted data. It is reasonable to consider whether the PLC firmware processes higher total load in a more consistent sequential manner than with smaller loads causing less predictable traffic patterns.

Overall, the communication demonstrates sequential polling of each read job before responses are received in high load runs. Where low load runs such as I-2 showed less cyclic and more irregular message behaviour. This difference can contribute to the significant increase in RTT observed between I-2 and I-4 in Figure 13. The more deterministic polling pattern in higher load may increase RTT, suggesting that the PLC firmware adopts a more structured but slightly less latency-optimised scheduling strategy under heavier demand.

7.3 Timing and Client Anomalies

The motivation for this study originated from the industrial case example, in which duplicates and invalid values were observed in OPC UA communication during production. The working assumption was that duplicates were caused by a mismatch between the sampling interval of the client and the update cycle of the server, rather than by an underlying communication fault.

To support this assumption, the deterministic behaviour of the server must first be verified to exclude it causing these anomalies. In Figure 17, the median server update duration is presented for all I-series test runs. The created boxplots in Figure 17 illustrate the average IQR of the update cycle from both test runs per single configuration. The median is indicated by a horizontal line within each box. As described in the server implementation section, the update duration represents the actual time required by the server to update its values before pausing until the next scheduled update tick. The configured server update interval for the runs is indicated at the top of the figure as 1450 ms. Being the target update period for all test runs shown in Figure 17.

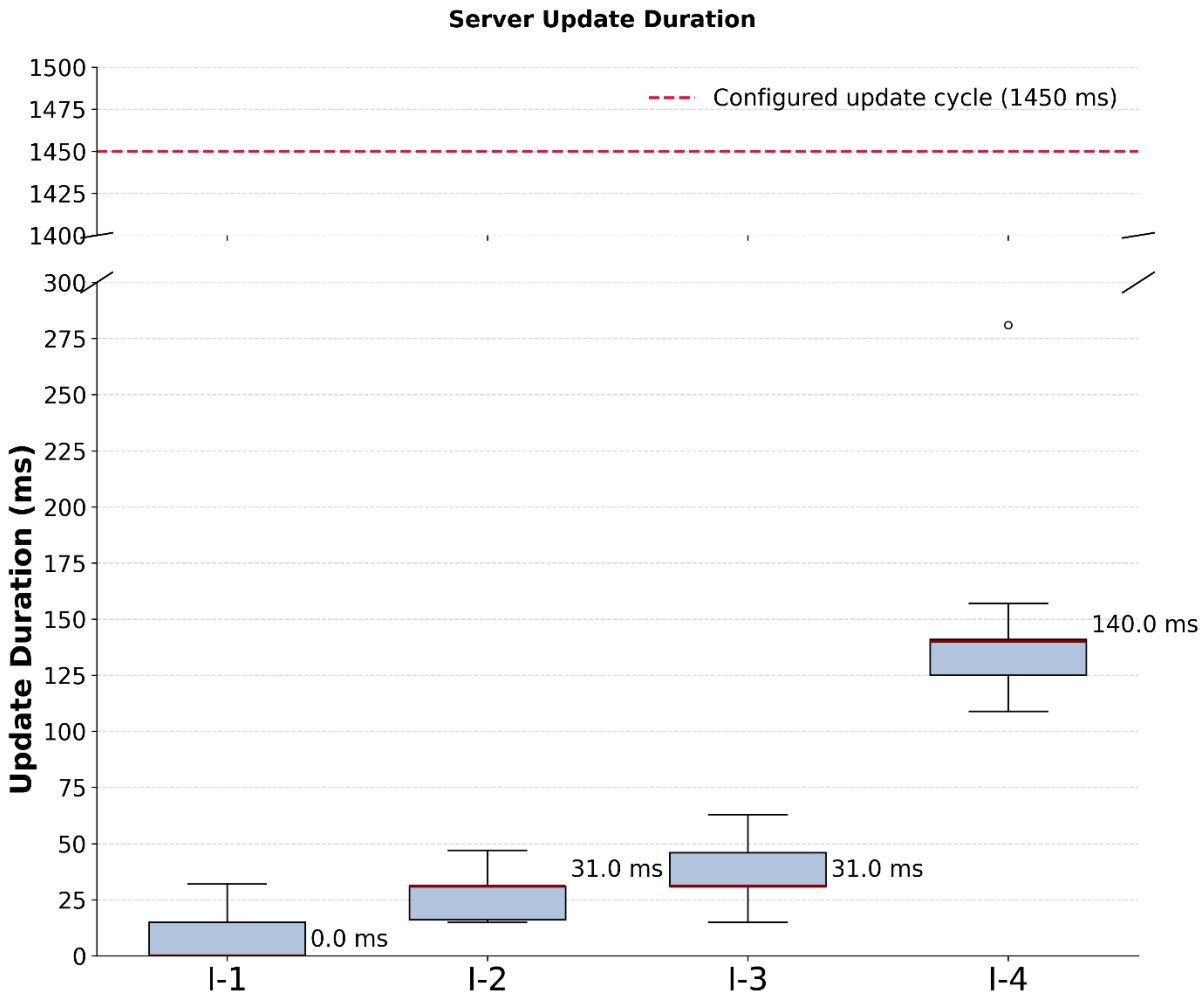

Figure 17. OPC UA server update duration with target update interval at 1450 ms. All the update durations clearly fall below the set target update cycle.

What can be clearly seen from Figure 17, is that the actual server update duration does not exceed the configured update interval. Even in the high load configuration (I-4), the median update duration is 140 ms. The upper whiskers of the boxplots remain below 200 ms, and even the outliers indicating the most extreme individual values, do not exceed 300 ms. Showing that the server updating process completes always within 300 ms. Therefore, it is rather evident that the server updates its values well within the configured 1450 ms interval.

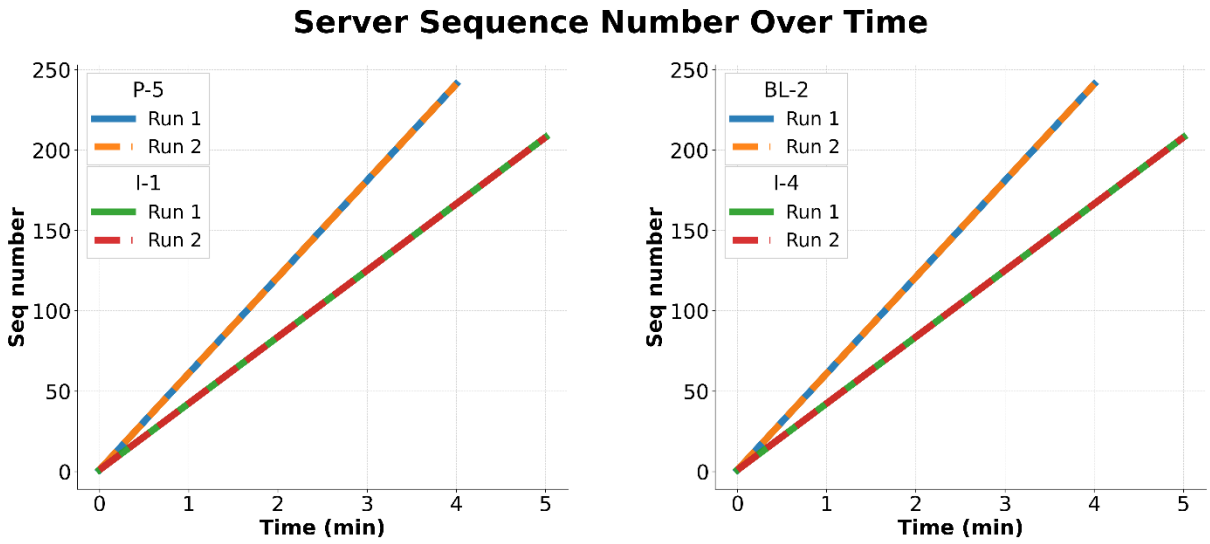

Figure 18. Server sequence number value over time, illustrating continuity.

Figure 18 was created just to demonstrate that the server updated every value without interruptions. The lines here indicate the sequence number of each update relative to the time of the run. The lines here show no deviation between runs with a constant gradient, indicating that the server sequence number over time for several run configurations was deterministic across updates. Further, no jumps or drops are visible which would indicate the irregular behaviour. Also, the created flag in server logs, indicating a missed update deadline was never triggered during any of the run configurations. These strongly suggest that the duplicates and anomalies observed in the measurements did not originate from server overload or missed update cycles.

While the test server created was very simplified version of the OPC UA server used in production and therefore they can't be directly compared. The testbed is still valid to use in for this test since most of the actual quantitative analysis focuses on the client behaviour.

Lastly, the jitter of the server update cycle was analysed using data from the server logs. This was also done in order to show the timing results independence from the server update behaviour. Figure 19 illustrates the variation in timing of each update cycle from the configured 1450 ms interval. The median jitter remains consistently at approximately +1 ms, indicating an actual effective update cycle of around 1451 ms. Even when considering the most extreme outliers with 35 ms deviation from the configured interval, the variation remains smaller compared to the lowest recorded median polling drift of 48 ms in configuration BL-1.

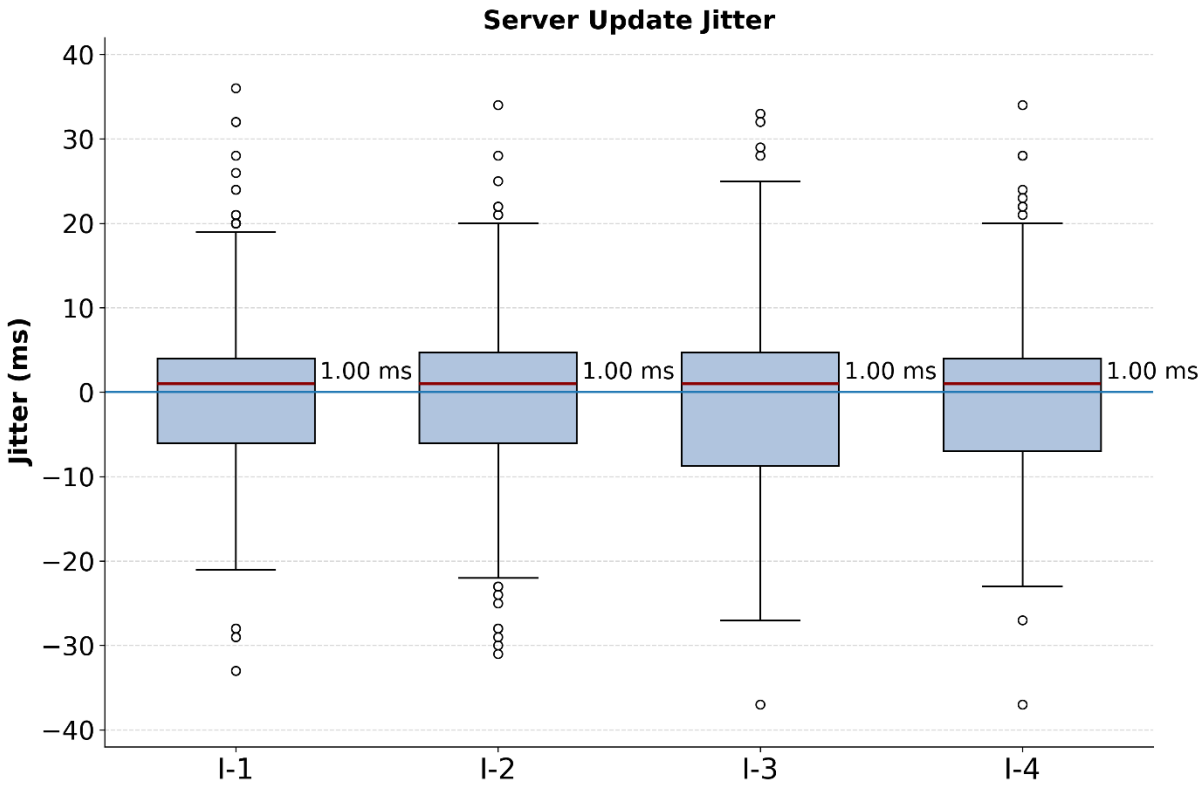

Figure 19. Jitter on server update time. The median jitter stays constantly small at 1 ms, but outliers do go all the way to 38 ms. This is still smaller than all the measured poll drift values, indicating minimal effect on the system behaviour.

Shown comparison demonstrates that client polling drift likely has greater impact on timing behaviour than the variation in the server update cycle. Within the constraints of this testbed setup, the server can therefore be reasonably excluded as influencing the occurrence of duplicates or missed values in any meaningful way. Only under very light load conditions could server timing variation be large enough, relative to client timing to potentially contribute to observed anomalies.

Having examined and reasonably excluded server behaviour as the cause of the observed anomalies, the focus shifts to the client to explain the occurrence of duplicates and missed values. As shown in Figure 14, variations in the polling interval had little effect on the primary performance metrics, since polling drift remained relatively constant across configurations. However, the client counters for duplicates and missed values varied significantly within these runs.

In configuration L-1, both duplicates and missed values were recorded in small quantities. In contrast, configuration L-2 had substantial increase in duplicates. To analyse this behaviour, the configured server update interval of 900 ms must be considered in relation to the timing of the client polling.

Looking at the Figure 14, the median polling drift in L-1 is 634 ms and the polling interval is set to 250 ms. Using the knowledge of how the TON timer is implemented in the logic, the actual client

polling cycle time can be calculated consisting of the configured polling interval and the observed polling drift. Therefore, the total client cycle time can be approximated as poll drift together with set TON timer resulting in 884 ms. The result of 884 ms is relatively similar to the server update interval of 900 ms. The small mismatch between these cycles times explains why only few duplicates and missed values are present in L-1. These are therefore which are likely caused by jitter in the client polling together with minor with this small mismatch in the server update timing.

From Figure 14 looking at the run L-2, the polling interval was reduced to 125 ms. The median polling drift here was 633 ms. The effective actual client cycle time calculated as previously is therefore the poll drift of L-2 together with set polling interval resulting in 758 ms together. The 758 ms here is significantly shorter in time than the set 900 ms server update interval. Creating situation where the client polls faster than the server update its values, leading to repeated reads of unchanged data with the same sequence number. This clearly aligns with the observed average of 62 duplicates across two test runs in this configuration.

Once this function is known, the polling interval could be adjusted to closely match the server update cycle by calculating what is the actual polling cycle with given load configuration. When the calculated polling cycle was applied to the test runs, it resulted in almost no duplicates or missed values. The few of them that remained, can reasonably be understood to be caused from timing jitter on the client side, or possible even the combined effect of client and server timing variation together mentioned previously.

For this study, the adjustment was tested and together with that shown a very strong proof that the duplicates and missed values are in fact due to these timing mismatches. The LL-1 configuration categorized as verification was done to show exactly the timing mismatches. LL-1 was executed under near maximum possible load conditions based on the restrictions of the PLC CPU [61]. During the test runs of LL-1, monitoring memory workload during the run in TIA Portal, indicated that 95% of the available CPU work memory was in use. This confirmed that the system was operating close to its operational limits.

In LL-1, the polling interval was set to 10 ms and the server update interval to 2100 ms. As shown earlier in Figure 12, the measured polling drift was approximately 2125 ms. Adding the 10 ms polling interval yields an effective client cycle time of 2135 ms. This value closely matches the set server update interval of 2100 ms. In LL-1 run the client metrics recorded only one missed value and zero duplicates. Since here the actual effective client cycle was slightly longer than the server update interval the single duplicate is considered reasonable and further supports the conclusion that duplicates are results of timing mismatches, rather than from high communication load causing anomalies.

Leading to the final discussion. Since the number of duplicates and missed values, are strongly influenced by the implemented TON timer logic, alternative timer configurations could potentially mitigate these overall. For example, an asynchronous timer mechanism operating independently of the read cycle could be implemented in which new values are only polled once in the separately defined timer threshold has been exceeded instead of waiting for all the reads to complete before next poll.

However, this consideration primarily concerns the structure and configuration of the client logic rather than a fundamental performance limitation of OPC UA communication. The used implementation was intentionally selected because it allows the impact of client processing load on large data volumes to be clearly and quantitatively observed.

Even with an alternative timing mechanism, similar challenges would likely arise in the presence of cycle jitter or when the client processing time, represented in this study by polling drift, exceeds the configured polling interval. In such cases, timing mismatches between client and server would still occur. This could be totally mitigated with using event-based communication such as the subscription model in OPC UA client-server communication.

7.3.1 Invalid Values

The anomaly of invalid values observed in the production environment where measurement values occasionally dropped to zero could not be reproduced in any of the test runs with the method of detection from client metrics. Throughout all test runs the invalid value counter remained at zero. Showing that every ID and OD value received by the OPC UA client was within the accepted range.

The reason for these invalid values occurring can therefore be only discussed with no actual results. One potential explanation to this relates to error handling in the PLC logic. In the testbed PLC logic, errors in of the read jobs were not processed separately. Only the “Busy” and “Done” outputs of a read job were used to know if a read job was completed. In the actual production if the read job outputs an error during operation, it could assign some default value (e.g., zero) to the variable node before a valid update is received, depending on the system configuration. The SCADA system may then momentarily read the default value. Which could logically result in observed anomalies in the HMI system.

Another possible explanation could be related to the firmware of the client. The experiments were conducted using TIA Portal V15.1 and an older firmware version. Differing from the versions used in the production. Firmware bugs could introduce subtle behavioural differences causing these anomalies for some unknown reason. Therefore, maintaining up-to-date firmware and development environments is even recommended by the vendor, to minimise the risk of malfunctions [63], [64].

7.4 Limitations of the Study

The primary limitation affecting how reliable the results are, has to do with how the data had to be gathered due to restrictions within the TIA Portal OPC UA client environment. Within the TIA portal itself the received client values could not be stored anywhere for logging purposes. Instead, the packet captures mechanism had to be implemented. Relating to this, the OPC UA messages used to analyse the performance were reconstructed using Tshark. Introducing potential inaccuracy related to TCP desegmentation and overall OPC UA message reconstruction. Using a dedicated historian with the PLC to record the received data directly, would have provided more reliably what data the client received. Now it cannot be entirely excluded that the results may have been influenced by the inaccuracy in data capture and message reconstruction.

The client PLC which was the part of the testbed most studied was from a single vendor. Therefore, the results best reflect the behaviour of the Siemens S7-1500 PLC with specified firmware and CPU. Meaning that the results are not directly transferable at a metric level to other OPC UA implementations. However, the general observations regarding polling behaviour, timing mismatch, and how the load affects performance can still offer useful information for other OPC UA communication systems.

Newer versions of TIA Portal support the subscriptions method in OPC UA communication. Subscriptions may reduce network load and improve communication efficiency compared to polling-based approaches [59]. Since this study focused exclusively on polling-based communication, it could be seen as a limiting factor related to this study.

Furthermore, the OPC UA server used in the experiments emulated the OPC UA server part of the smart sensor used in production. The potential impact of the actual smart sensor, with its own processing delays affecting the communication can work as a one limiting factor since the actual smart sensor should be implemented in the testbed to verify its behaviour under high load conditions.

Finally, as explained in the industrial context section, the company's internal network was excluded from the testbed. Research was conducted with the company's own specialists to point out how the internal network could potentially affect the communication in the production. Within the scope of communication between the smart sensor and the PLC, the internal network infrastructure was concluded to introduce no significant delay or filtering mechanisms beyond standard routing using network switches.

7.5 Results Relation to Other Research

Although the limitations indicate that the results of the conducted study are to some extent hardware-specific, several previous studies report findings that show some similarities with the observed OPC UA client-server behaviour in this study.

As previously mentioned, L. Freitas et al. [9] conducted an OPC UA performance evaluation using PLC operating as an OPC UA server. Their configuration was also conducted on a different vendor PLC and only RTT was measured in comparison with MQTT. Still the reported results provided a reference point to use. Although the authors did not explicitly state the total data amount transferred but instead referred to a “defined number of nodes” [9], their reported average RTT was 3.65 ms. Aligning well with the low load RTT measurements obtained the thesis testbed being around 3 ms. This gives some support to the validity of the RTT measurements conducted in the present study.

Another relevant study also mentioned in related work section by X. Wang et al. [15] evaluated multiple communication protocols within a fully software-based test environment. Several performance metrics were compared. From this study, interesting result is the reported packet loss ratio when using similar polling-based request-response communication with OPC UA. The study concluded that, with a 0.5 s sampling interval, the client received 496 out of 500 transmitted messages. Whereas with a 0.002 s sampling interval, only approximately 5% of the messages were successfully received. It is reasonable to argue that this behaviour does not reflect actual IP-level packet loss, but rather application-level timing mismatch. Which closely corresponds to the findings of missed values and duplicates in the present study, where timing mismatches between client polling and server update cycles resulted in logical data loss rather than actual packet loss.

Numerous other studies have investigated OPC UA performance. However, many of them focus on either PubSub or subscription OPC UA communication. These not being directly comparable to the approach examined in this thesis. For example, C. Zunino et al. [65] evaluated subscription-based OPC UA communication and reported mean network latencies between 1 and 2 ms in a LAN setup using non-industrial hardware. However, observed low latency does align with Siemens documentation suggesting that subscription mechanisms can reduce network load and improve communication efficiency [59].

Directly comparable study with an identical hardware configuration and polling-based communication was not identified, the partially overlapping findings in the literature support the general validity of the observations presented in this work. Most meaningful of these being the consistency in RTT ranges and the possible documented timing mismatch in of polling-based communication support the broader applicability of the conclusions drawn in this thesis.

8 Conclusion

The goal of this study was to load test the polling-based OPC UA client-server communication with given hardware specifications. Through the conducted experiments, it was possible to determine which factors had the greatest influence on communication performance and which mechanisms could lead to the anomalies observed in the industrial context case. The main conclusions of the study presented below are organized to answer each of the research questions:

RQ1: The overall communication load determined by both the payload size and the number of nodes, was found to have the strongest impact on RTT and its associated jitter.

RQ2: The results indicate that the number of nodes has the greatest influence on client-side processing delay. Which in turn increases polling drift under high load conditions.

RQ3: During short sampling intervals the communication performance was primarily constrained by client processing time rather than by the configured polling interval itself.

RQ4: Under the given restrictions of the hardware, the number of parallel Read requests from separate read jobs was observed to only have minor direct effect on the measured performance metrics and the reliability of the communication. Different scheduling behaviours associated with multiple read jobs were however identified. These are believed to be firmware related.

RQ5: Varying payload sizes mostly affected the total overall communication load, therefore contributing more to increase in RTT and jitter rather than the polling drift which was mostly affected by the number of total nodes.

RQ6: The study demonstrated that the observed duplicates and missed values originated mainly from timing mismatches between the client polling cycle and the server update cycle, rather than from actual data loss or communication errors. Invalid values were not observed during testing. Possible caused for them are likely related to PLC logic or firmware.

The created testbed provides a controlled environment for analysing communication behaviour with OPC UA and measuring the approximate client processing time. This makes optimisation of polling interval possible. Furthermore, the alternative communication approaches are discussed. highlighting the OPC UA subscription model as a potentially more reliable solution for the production with event-based communication. Particularly in terms of reducing system load and improving robustness. The

importance of maintaining up-to-date hardware, software, and especially firmware versions is also emphasised as a means of ensuring system reliability.

The main limitation of this study is related to the specific testbed configuration and the absence of direct client-side data logging. The results are inherently specific to the used vendor's hardware, which limits direct comparison to other OPC UA platforms. The experimental setup also focused solely on communication between the smart sensor and the PLC. The potential influence of the SCADA system on the observed anomalies was not included. Finally, the emulated OPC UA server implementation lacked some features present in smart sensor used in production, meaning that the results were more directed to the behaviour of client PLC.

Future research should extend the study to include the SCADA system with possible historian logging as part of the testbed. This would make possible to capture actual values received by the client. As the OPC UA security features were excluded from this study, their impact should also be examined because encryption and authentication mechanisms may affect communication performance. Most importantly, a direct comparison between subscription-based and polling-based OPC UA communication under identical hardware conditions would provide further validation of the recommendations derived from this work.

To conclude, polling-based OPC UA communication is reliable within its limits when properly configured. Performance constraints should be understood in terms of communication architecture and implementation factors rather than communication method limitations alone. This study provides practical guidance for the design, configuration, and troubleshooting of given OPC UA communication in industrial automation environments.

References

[1] B. Babayigit and M. Abubaker, “Industrial Internet of Things: A Review of Improvements Over Traditional SCADA Systems for Industrial Automation,” *IEEE Syst. J.*, vol. 18, no. 1, pp. 120–133, Mar. 2024, doi: 10.1109/JSYST.2023.3270620.

[2] A. Morato, S. Vitturi, F. Tramarin, and A. Cenedese, “Assessment of Different OPC UA Implementations for Industrial IoT-Based Measurement Applications,” *IEEE Trans. Instrum. Meas.*, vol. 70, pp. 1–11, 2021, doi: 10.1109/TIM.2020.3043116.

[3] F. Salcher, S. Finck, and M. Hellwig, “A Smart Shop Floor Information System Architecture Based on the Unified Namespace,” in *2024 IEEE International Conference on Engineering, Technology, and Innovation (ICE/ITMC)*, Jun. 2024, pp. 1–9. doi: 10.1109/ICE/ITMC61926.2024.10794387.

[4] P. Denzler, T. Frühwirth, A. Kirchberger, M. Schoeberl, and W. Kastner, “Static Timing Analysis of OPC UA PubSub,” in *2021 17th IEEE International Conference on Factory Communication Systems (WFCS)*, Jun. 2021, pp. 167–174. doi: 10.1109/WFCS46889.2021.9483614.

[5] M. S. Sonkor and B. García de Soto, “Operational Technology on Construction Sites: A Review from the Cybersecurity Perspective,” *J. Constr. Eng. Manag.*, vol. 147, no. 12, p. 04021172, Dec. 2021, doi: 10.1061/(ASCE)CO.1943-7862.0002193.

[6] E. M. Martinez, P. Ponce, I. Macias, and A. Molina, “Automation Pyramid as Constructor for a Complete Digital Twin, Case Study: A Didactic Manufacturing System,” *Sensors*, vol. 21, no. 14, p. 4656, Jan. 2021, doi: 10.3390/s21144656.

[7] M. Ladegourdie and J. Kua, “Performance Analysis of OPC UA for Industrial Interoperability towards Industry 4.0,” *IoT*, vol. 3, no. 4, pp. 507–525, Dec. 2022, doi: 10.3390/iot3040027.

[8] P. Dimitrov and M. Alexandrova, “From Automation Pyramid to Industry 4.0: Transitioning Process and Practical Applications,” in *2024 International Conference Automatics and Informatics (ICAI)*, Oct. 2024, pp. 71–75. doi: 10.1109/ICAI63388.2024.10851506.

[9] L. Freitas *et al.*, “OPC-UA in interoperability – a performance comparative testing,” *IFAC-Pap.*, vol. 58, no. 8, pp. 240–245, Jan. 2024, doi: 10.1016/j.ifacol.2024.08.127.

[10] L. Freitas *et al.*, “OPC UA and MQTT performance analysis within a unified namespace context,” *Internet Things*, vol. 33, p. 101734, Sep. 2025, doi: 10.1016/j.iot.2025.101734.

[11] S. Profanter, A. Tekat, K. Dorofeev, M. Rickert, and A. Knoll, “OPC UA versus ROS, DDS, and MQTT: Performance Evaluation of Industry 4.0 Protocols,” in *2019 IEEE International Conference on Industrial Technology (ICIT)*, Feb. 2019, pp. 955–962. doi: 10.1109/ICIT.2019.8755050.

[12] A. Burger, H. Koziolek, J. Rückert, M. Platenius-Mohr, and G. Stomberg, “Bottleneck Identification and Performance Modeling of OPC UA Communication Models,” in *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, in ICPE ’19. New York,

NY, USA: Association for Computing Machinery, Apr. 2019, pp. 231–242. doi: 10.1145/3297663.3309670.

[13] D. Silva, L. I. Carvalho, J. Soares, and R. C. Sofia, “A Performance Analysis of Internet of Things Networking Protocols: Evaluating MQTT, CoAP, OPC UA,” *Appl. Sci.*, vol. 11, no. 11, p. 4879, Jan. 2021, doi: 10.3390/app11114879.

[14] S. Cavalieri and F. Chiacchio, “Analysis of OPC UA performances,” *Comput. Stand. Interfaces*, vol. 36, no. 1, pp. 165–177, Nov. 2013, doi: 10.1016/j.csi.2013.06.004.

[15] X. Wang, İ. Mutlu, F. Rani, L. Drowatzky, and L. Urbas, “A Comparative Study to Evaluate the Performance of Communication Protocols for Process Industry,” in *2022 32nd International Telecommunication Networks and Applications Conference (ITNAC)*, Nov. 2022, pp. 170–177. doi: 10.1109/ITNAC55475.2022.9998327.

[16] C. Hunt, *TCP/IP Network Administration*. O’Reilly Media, Inc., 2002.

[17] L. Parziale *et al.*, *TCP/IP Tutorial and Technical Overview*. IBM Redbooks, 2006.

[18] H. Trifonov and D. Heffernan, “OPC UA TSN: a next-generation network for Industry 4.0 and IIoT,” *Int. J. Pervasive Comput. Commun.*, vol. 19, no. 3, pp. 386–411, Dec. 2021, doi: 10.1108/IJPCC-07-2021-0160.

[19] R. T. Braden, “Requirements for Internet Hosts - Communication Layers,” Internet Engineering Task Force, Request for Comments RFC 1122, Oct. 1989. doi: 10.17487/RFC1122.

[20] “TCP/IP Model,” GeeksforGeeks. Accessed: Oct. 30, 2025. [Online]. Available: https://www.geeksforgeeks.org/computer-networks/tcp-ip-model/

[21] A. Tyagi, “TCP/IP Protocol Suite,” *Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol.*, pp. 59–71, Jul. 2020, doi: 10.32628/CSEIT206420.

[22] A. Arestova, M. Martin, K.-S. J. Hielscher, and R. German, “A Service-Oriented Real-Time Communication Scheme for AUTOSAR Adaptive Using OPC UA and Time-Sensitive Networking,” *Sensors*, vol. 21, no. 7, p. 2337, Jan. 2021, doi: 10.3390/s21072337.

[23] C. Bayılmış, M. A. Ebleme, Ü. Çavuşoğlu, K. Küçük, and A. Sevin, “A survey on communication protocols and performance evaluations for Internet of Things,” *Digit. Commun. Netw.*, vol. 8, no. 6, pp. 1094–1104, Dec. 2022, doi: 10.1016/j.dcan.2022.03.013.

[24] R. Neumann, C. von Arnim, M. Neubauer, A. Lechler, and A. Verl, “Requirements and Challenges in the Configuration of a Real-Time Node for OPC UA Publish-Subscribe Communication,” in *2023 29th International Conference on Mechatronics and Machine Vision in Practice (M2VIP)*, Nov. 2023, pp. 1–6. doi: 10.1109/M2VIP58386.2023.10413399.

[25] N. I. Aristova, “Ethernet in industrial automation: Overcoming obstacles,” *Autom. Remote Control*, vol. 77, no. 5, pp. 881–894, May 2016, doi: 10.1134/S0005117916050118.

[26] “What is an OPC classic?,” OPC Expert. Accessed: Oct. 31, 2025. [Online]. Available: https://opcexpert.com/opc-faq/what-is-an-opc-classic/

[27] M. H. Schwarz and J. Börcsök, “A survey on OPC and OPC-UA: About the standard, developments and investigations,” in *2013 XXIV International Conference on Information, Communication and Automation Technologies (ICAT)*, Oct. 2013, pp. 1–6. doi: 10.1109/ICAT.2013.6684065.

[28] “Classic,” OPC Foundation. Accessed: Nov. 01, 2025. [Online]. Available: https://opcfoundation.org/about/opc-technologies/opc-classic/

[29] M. Son and M.-J. Yi, “A study on OPC specifications: Perspective and challenges,” in *International Forum on Strategic Technology 2010*, Oct. 2010, pp. 193–197. doi: 10.1109/IFOST.2010.5668110.

[30] “Unified Architecture - Landingpage,” OPC Foundation. Accessed: Nov. 14, 2025. [Online]. Available: https://opcfoundation.org/about/opc-technologies/opc-ua/

[31] S. Cavalieri and G. Cutuli, “Performance evaluation of OPC UA,” in *2010 IEEE 15th Conference on Emerging Technologies & Factory Automation (ETFA 2010)*, Sep. 2010, pp. 1–8. doi: 10.1109/ETFA.2010.5641184.

[32] A. Tidrea and A. Korodi, “ECC Implementation and Performance Evaluation for Securing OPC UA Communication,” in *2023 IEEE 22nd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, Nov. 2023, pp. 1704–1711. doi: 10.1109/TrustCom60117.2023.00232.

[33] “UA Part 6: Mappings - 5 Data encoding.” Accessed: Mar. 22, 2026. [Online]. Available: https://reference.opcfoundation.org/Core/Part6/v104/docs/5

[34] “Polling versus Subscription,” OPC Expert. Accessed: Mar. 08, 2026. [Online]. Available: https://opcexpert.com/polling-versus-subscription/

[35] “OPC UA Data Exchange & Structures.” Accessed: Mar. 08, 2026. [Online]. Available: https://www.winccoa.com/documentation/WinCCOA/3.20/en_US/OPC_UA/opc_ua_c_data_exchange.html

[36] “UA Part 4: Services - 5.13.1 Subscription model.” Accessed: Mar. 22, 2026. [Online]. Available: https://reference.opcfoundation.org/Core/Part4/v104/docs/5.13.1

[37] “UA Part 4: Services - 5.12.1.5 Queue parameters.” Accessed: Nov. 15, 2025. [Online]. Available: https://reference.opcfoundation.org/Core/Part4/v104/docs/5.12.1.5

[38] Q.-D. NGUYEN, S. DHOUIB, and P. BELLOT, “A Unified Method to Design Bridges for OPC UA PubSub Networks in the Industrial IoT,” in *2022 IEEE 27th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, Nov. 2022, pp. 47–52. doi: 10.1109/CAMAD55695.2022.9966908.

[39] D. Hästbacka, P. Kannisto, and A. Kätkytniemi, “Interoperability of OPC UA PubSub with Existing Message Broker Integration Architectures,” in *IECON 2022 – 48th Annual Conference of the IEEE Industrial Electronics Society*, Oct. 2022, pp. 1–6. doi: 10.1109/IECON49645.2022.9969039.

[40] Z. Liu and P. Bellot, “A configuration tool for MQTT based OPC UA PubSub,” in *2020 RIVF International Conference on Computing and Communication Technologies (RIVF)*, Oct. 2020, pp. 1–6. doi: 10.1109/RIVF48685.2020.9140792.

[41] Z. Liu and P. Bellot, “OPC UA PubSub Implementation and Configuration,” in *2019 6th International Conference on Systems and Informatics (ICSAI)*, Nov. 2019, pp. 1063–1068. doi: 10.1109/ICSAI48974.2019.9010445.

[42] “UA Part 14: PubSub - 5.4.6.2 Broker-less Middleware.” Accessed: Nov. 15, 2025. [Online]. Available: https://reference.opcfoundation.org/Core/Part14/v105/docs/5.4.6.2

[43] “UA Part 14: PubSub - 5.4.6.3 Broker-based Middleware.” Accessed: Nov. 15, 2025. [Online]. Available: https://reference.opcfoundation.org/Core/Part14/v105/docs/5.4.6.3

[44] O. Gilles, D. Gracia Pérez, P.-A. Brameret, and V. Lacroix, “Securing IIoT communications using OPC UA PubSub and Trusted Platform Modules,” *J. Syst. Archit.*, vol. 134, p. 102797, Jan. 2023, doi: 10.1016/j.sysarc.2022.102797.

[45] “UAFX Part 80: Overview and Concepts - 5 Overview.” Accessed: Nov. 20, 2025. [Online]. Available: https://reference.opcfoundation.org/UAFX/Part80/v100/docs/5

[46] “The OPC Foundation releases the OPC UA Field eXchange (UAFX) Specifications,” OPC Foundation. Accessed: Nov. 20, 2025. [Online]. Available: https://opcfoundation.org/news/press-releases/the-opc-foundation-releases-the-opc-ua-field-exchange-uafx-specifications/

[47] K. Justmann, “Automation of an industrial OPC UA FX use case through the usage of proactive Asset Administration Shells”.

[48] T. Zhang, G. Wang, C. Xue, J. Wang, M. Nixon, and S. Han, “Time-Sensitive Networking (TSN) for Industrial Automation: Current Advances and Future Directions,” *ACM Comput Surv*, vol. 57, no. 2, p. 30:1-30:38, Oct. 2024, doi: 10.1145/3695248.

[49] “GMP Guide for SIMATIC WinCC (TIA Portal) V15.” Siemens AG, Mar. 2019.

[50] Reinhard Exel, Thilo Sauter, Paolo Ferrari, and Stefano Rinaldi, *Chapter 21 Clock Synchronization in Distributed Systems Using NTP and PTP*, 2nd Edition. in Industrial Communication Technology Handbook.

[51] A. Libri, A. Bartolini, D. Cesarini, and L. Benini, “Evaluation of NTP/PTP fine-grain synchronization performance in HPC clusters,” in *Proceedings of the 2nd Workshop on AutotuniNg and aDaptivity AppRoaches for Energy efficient HPC Systems*, Limassol Cyprus: ACM, Nov. 2018, pp. 1–6. doi: 10.1145/3295816.3295819.

[52] “UAFX Part 80: Overview and Concepts - 6 UAFX Concepts.” Accessed: Nov. 20, 2025. [Online]. Available: https://reference.opcfoundation.org/UAFX/Part80/v100/docs/6

[53] E. Shahri, P. Pedreiras, and L. Almeida, “A Scalable Real-Time SDN-Based MQTT Framework for Industrial Applications,” *IEEE Open J. Ind. Electron. Soc.*, vol. 5, pp. 215–235, 2024, doi: 10.1109/OJIES.2024.3373232.

[54] K. Kamoun, F. Hmissi, S. Ouni, and S. Ouni, “Improvement of MQTT semantic to minimize data flow in IoT platforms based on distributed brokers,” *Trans. Emerg. Telecommun. Technol.*, vol. 35, no. 2, p. e4945, 2024, doi: 10.1002/ett.4945.

[55] E. S. Llamuca, C. A. Garcia, J. E. Naranjo, and M. V. Garcia, “Cyber-Physical Production Systems for Industrial Shop-Floor Integration Based on AMQP,” in *2019 International Conference on Information Systems and Software Technologies (ICI2ST)*, Nov. 2019, pp. 48–54. doi: 10.1109/ICI2ST.2019.00014.

[56] F. Iqbal, M. Gohar, H. Alquhayz, S.-J. Koh, and J.-G. Choi, “Performance evaluation of AMQP over QUIC in the internet-of-thing networks,” *J. King Saud Univ. - Comput. Inf. Sci.*, vol. 35, no. 4, pp. 1–9, Apr. 2023, doi: 10.1016/j.jksuci.2023.02.018.

[57] M. S. Rocha, G. S. Sestito, A. L. Dias, A. C. Turcato, D. Brandão, and P. Ferrari, “On the performance of OPC UA and MQTT for data exchange between industrial plants and cloud servers,” *Acta IMEKO*, vol. 8, no. 2, pp. 80–87, Jun. 2019, doi: 10.21014/acta_imeko.v8i2.648.

[58] F. Pedro, F. Veiga, and F. Mascarenhas-Melo, “Impact of GAMP 5, data integrity and QbD on quality assurance in the pharmaceutical industry: How obvious is it?,” *Drug Discov. Today*, vol. 28, no. 11, p. 103759, Nov. 2023, doi: 10.1016/j.drudis.2023.103759.

[59] “SIMATIC STEP 7 and WinCC Engineering V18 System Manual.” Siemens AG Digital Industries, Nov. 2022. Accessed: Feb. 19, 2026. [Online]. Available: https://cache.industry.siemens.com/dl/files/056/109815056/att_1121875/v5/STEP_7_WinCC_V18_enUS_en-US.pdf

[60] “SCADA Systems: Industrial Operations and Integration,” Siemens Digital Industries Software. Accessed: Feb. 19, 2026. [Online]. Available: https://www.siemens.com/en-us/products/scada/

[61] “S7 user block for the OPC UA client of a SIMATIC S7-1500.” Siemens AG, Aug. 2021.

[62] “Wireshark User’s Guide.” Accessed: Feb. 19, 2026. [Online]. Available: https://www.wireshark.org/docs/wsug_html_chunked/

[63] “Firmware update for CPU 1511C-1 PN - ID: 109479287 - Industry Support Siemens.” Accessed: Feb. 26, 2026. [Online]. Available: https://support.industry.siemens.com/cs/document/109479287/firmware-update-for-cpu-1511c-1-pn?dti=0&lc=en-CL

[64] “TIA Portal V21 Updates - ID: 109989775 - Industry Support Siemens.” Accessed: Feb. 26, 2026. [Online]. Available: https://support.industry.siemens.com/cs/document/109989775/tia-portal-v21-updates-?dti=0&lc=en-KR

[65] C. Zunino, G. Cena, S. Scanzio, and A. Valenzano, “Experimental Characterization of Asynchronous Notification Latency for Subscriptions in OPC UA,” in *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sep. 2021, pp. 01–08. doi: 10.1109/ETFA45728.2021.9613502.