2023 IEEE 19th International Conference on Automation Science and Engineering (CASE) | 979-8-3503-2069-5/23/$31.00 ©2023 IEEE | DOI: 10.1109/CASE56687.2023.10260673

Delay Modelling and Measurement of Multi-Agent Systems with Digital Twins in a Gear Assembly Use Case

Birgit Vogel-Heuser, *IEEE Fellow**, Yash Deshpande†, Fandi Bi*, Jingyun Zhao*, Dominik Hujo*, Wolfgang Kellerer†
André Kraft‡, Bernd Vojanec§, Timo Markert§
*Chair of Automation and Information Systems, Technical University of Munich, Germany
E-mail: {vogel-heuser, fandi.bi, jingyun.zhao, dominik.hujo}@tum.de
†Chair of Communication Networks, Technical University of Munich, Germany
E-mail: {yash.deshpande,wolfgang.kellerer}@tum.de
‡BMW AG, Munich, Germany, E-mail: andre.kraft@bmw.de
§WITTENSTEIN SE, Corporate Research & Development, Igersheim, Germany
E-mail: {timo.markert; bernd.vojanec}@wittenstein.de

***Abstract*—In the context of increasing automation and digitization of production processes, efficient communication and coordination among agents in Multi-Agent-Systems (MAS) is crucial for achieving optimal performance and productivity. This study analyzes a MAS's communication and coordination processes utilized in a gearbox assembly scenario. The system is decomposed based on the assembly process, focusing on delay times. Messages are classified based on sender and receiver agents, communication protocols, message types, and query content. Local and network delays of each communication message are measured and compared to modeled delays. The results indicate that modeling delay times before implementation can lead to a more efficient approach to elaborating on complex systems' hard and soft real-time capabilities. The delay times estimated through the analysis can be employed in forthcoming models, enabling the modeling of delay times before system implementation. Additionally, the generalizability of the findings allows for their application to repetitive modules within production systems.**

***Index Terms*—multi-agent-system (MAS), communication and coordination processes, gearbox assembly, delay measurement, delay modeling, real-time capabilities, digital twin**

I. Introduction

Agent-based systems are an effective way to implement a decentralized system architecture, which can be advantageous for large systems. However, due to the size of these systems, it may be difficult to measure operation times and delays directly. Instead, they are often modelled or calculated, which can lead to inaccuracies and lack of generalizability. Additionally, planning for these systems may be challenging as it is often done before the actual system is built. Therefore, it is important to carefully consider the limitations and potential challenges associated with using agent-based systems in order to ensure their effectiveness and success. In our research, we have identified limitations in the domain-specific language (DSL) for modeling larger systems and delays in DSL for robot-alike systems (DSL4RAS). To address these limitations, we propose an expanded and adapted version of the DSL. To evaluate the effectiveness of our proposed method, we applied it to two industrial use cases, including their Digital Twins (DTs), and compared the modeling outcomes with the measured outcomes for operation and delay times. Our results show that our method can successfully model and calculate delay times for modular systems in an agent-based environment, while it allows for a more generalizable way, which can be particularly useful for complex systems.

II. Background and Related Work

This section presents the standard commercial off-the-shelf (COTS) ethernet network components and distributed protocols for real-time communication and discusses its limitations. To address this challenge, This section presents Multi-Agent-System (MAS) control strategies described by DSL for timing characteristics and requirements of modular and distributed systems, especially in the context of Cyber-Physical Production Systems (CPPS) and DTs.

A. Network Delays

Standard COTS ethernet network components and distributed protocols are not designed for real-time communication. While average per-hop One-way delay (OWD) in the network could be low, individual outliers in the OWD could be very high. Such distributed Internet Protocol (IP) networks cannot guarantee reliable and timely information delivery. Modern factories with MAS will need higher bandwidth to run applications such as collective learning, point cloud updates for Digital Twin (DT), Augmented Reality (AR)/Virtual Reality (VR) etc. Such a factory will need very high bandwidth, which cannot be serviced by

legacy industrial networks which provide real-time guarantees. Moreover, agent-based endpoints may need to run on operating system (OS) such as Robot OS (ROS) [1] and not real-time Programmable Logic Controller (PLC) like controllers. Such OSs need a full IP network and usually run on COTS equipment.

Even in a simple assembly example, many short messages need to be exchanged by the agents to coordinate the start and completion of individual tasks in the entire assembly process. These messages must be reliable, and the sending agent must verify their delivery to the appropriate receiving agent. Thus, the Transmission Control Protocol (TCP) ensures that the agents' coordination messages are reliable, ordered, and error-checked. A particular TCP message sequence is called a flow. Many methods have been proposed to model the message delivery time of a TCP flow [2], [3]. The messages described in this paper are typically characterized by short sizes of only a few bytes that can fit into one packet. The flow completion time of such messages can be modeled using a more appropriate model considering the time of the TCP initiation handshaking phase [4]. Such a model sums the expected delays for the different phases of TCP. In section IV-C, we propose a model for the message completion time for a given TCP connection in an example factory with MAS.

B. Multi-Agent-Systems and DSL4RAS

MAS comprises numerous self-governing agents collaborating to accomplish a shared objective or resolve a problem. Each agent within a MAS possesses unique objectives, skills, and knowledge and can perceive its surroundings and communicate with other agents [5]. In production systems, MAS control strategies have been established to enhance the adaptability of intricate and dynamic manufacturing systems [6]. Agents in a modern industrial network rely on the socket interface to send data. Sockets use the layered OSI reference model [7] to send and receive data. Figure 1 shows the layers a packet goes through after it's written to the socket. However, the delay in the information as it passes through the layers in the agent devices is difficult to model until real-time OS are used at the agent's hosts.

There is a trend towards using DSL and models to crestandardate visual notations that are more familiar to industrial automation practitioners [8]. Multi-model approaches in the early design phase allow investigating system performance more transparently [9]. Accurately describing the timing characteristics and requirements of modular and distributed systems is crucial in the context of CPPS and DTs [10]. Hujo et al. [9] introduce an extension to an established notation that facilitates the examination of timing properties and demands of automation solutions and expands it by incorporating the relationship between physical devices and their controlling software based on soft and hard real-time capabilities. The majority of sensor and actuator delays were ascertained through measurement due to the limited

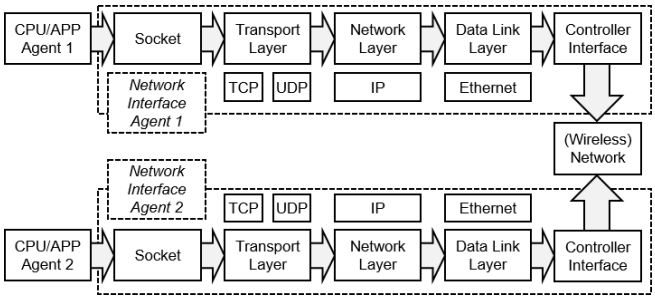

Fig. 1: The layered network stack in the Agent host. The packet goes from one application to the other through each layer, adding delay. This work proposes a simple delay model of the message completion time between agents. The network can either be a wired local network, wireless local network or internet.

information available in the corresponding data sheets. In the case of agents incorporating the digital twin, the number of reference points available is even more limited. To address this challenge, the proposed approach involves comparing delay times measured in the physical system to those predicted by the models, allowing for more precise information regarding the system's timing properties.

III. System Model

This section provides an introduction to the underlying network assumptions, the fundamental assumptions for the industrial setup, and a detailed description of the use case.

A. Network Assumptions

We assume that the network traffic is carried over COTS hardware. An IP address identifies each device, and packets are dropped in the network due to congestion. The delay of packets in the network from agent A to B is stable throughout the assembly process. Such a controlled network can be achieved using central monitoring and control in of the factory network. We use *Chameleon* [11] to make sure that no packets are dropped and the total end-to-end delay in the network is stable and bounded. Our future work will address the delays arising from faults and network path reconfigurations. All agent-based systems except for the Azure Analysis Services (AAS) based DT are implemented on the Linux OS. The Cubic TCP variant [12] is used as it is optimized for fast operation with short messages. For the sake of a simplified evaluation of our models, a single network switch connects all agents in our network except for the DT.

B. Basic Assumptions

Investigating system components in an CPPS setting, we assume a logistics and assembly scenario for this work. The system setup includes a storage agent (supermarket cell), a path planning agent ("Central.AI"), a logistics agent (MiR Transporter Robot), an assembly agent (Franka Emika Panda

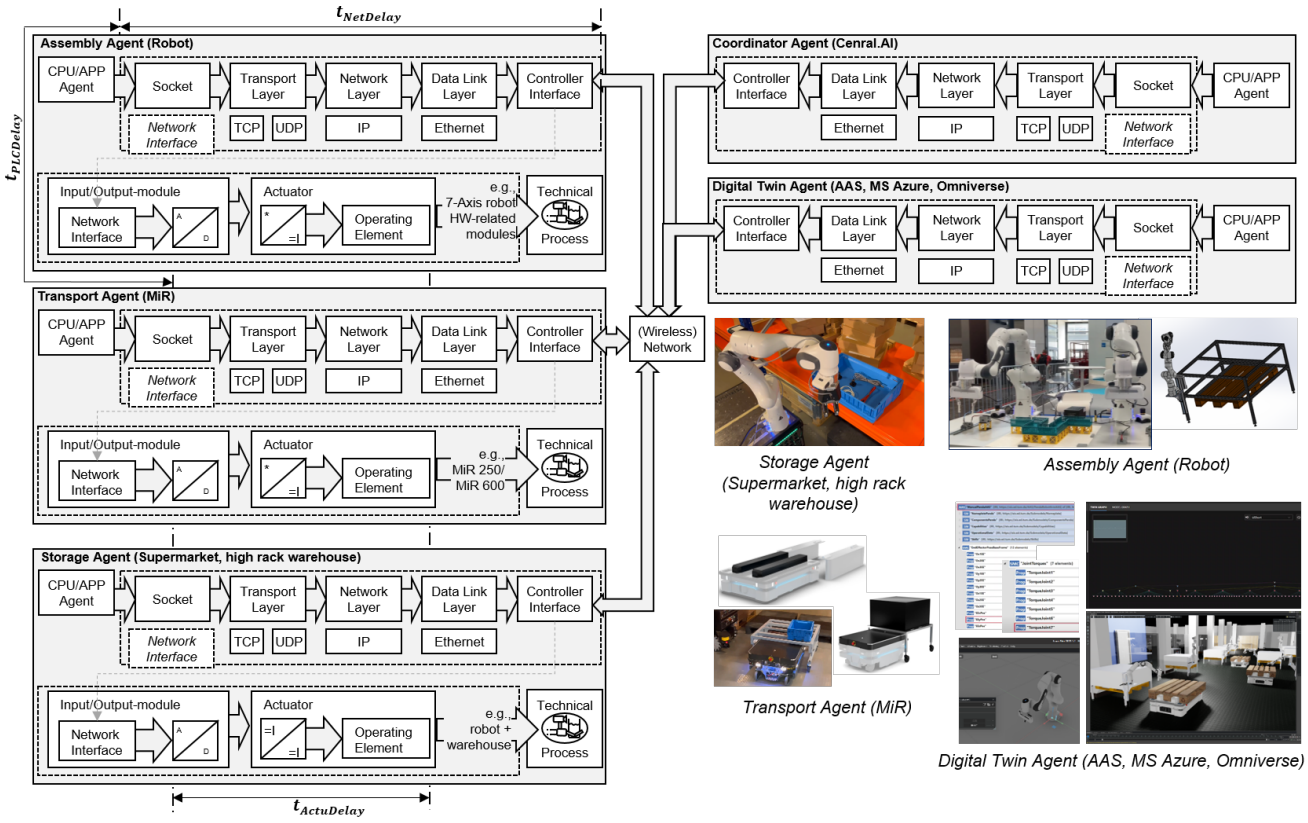

Fig. 2: Use Case Description of this work based on previous DSL design [8], [9] with lab implementations of the agents

robot), and the digital twin agent (implemented by the AAS, MS Azure Digital Twin, and Nvidia Omniverse), cp. Fig. 2. Given the academic phase of the setup, the system is not restricted by either hard or soft real-time boundaries; rather, the primary focus is to examine the system's real-time capabilities, including the investigation of mean delays associated with each setup's communication channels.

C. Digital Twin Setup

The system's architecture is based on a socket-based client/server principle, enabling bidirectional data exchange between applications via TCP/IP. The digital twin is created using the Asset Administration Shell (AAS) as a well-defined asset representation, deployed on the Azure DT platform. The AAS encompasses the necessary information for the assembly process of gearbox parts with the Panda Robot, obtained from relevant documentation and online resources. The AAS is deployed in Azure DT, and a 3D representation of the assets is created using Nvidia Omniverse Isaac Sim. The simulation environment allows for real-time monitoring and interaction with the digital twin. The Extensible Twin Communication Interface Module (ETCIM) serves as an interface application, facilitating the integration between the Azure DT AAS instances, the Isaac Sim extension, and the hardware control programs [13].

D. Gear Assembly Use Case Description

The study examines an industrial assembly use case of two gearbox parts by Wittenstein for a customer order. DSL4Production is applied on our assembly use case, illustrated in Fig. 2. The workflow (cp. Fig. 3) involves breaking down the customer order into tasks, which are further decomposed into skills, and primitives. The MAS then allocates the necessary agents, verifies their availability and capability, while updating their individual digital twin and the central path planning agent after succeeding each task. The use of task decomposition, agent allocation, and digital twin tracking enables assessing the automated processing of orders, delays, and resource utilization. In the context of DSL4Production, the majority of delay times are obtained through measurements. However, it requires significant practical effort to measure each communication individually. Once we verify that our model for the system's delay aligns with the measured delay times, we can rely on mathematical equations to make further assumptions about delays, thus simplifying the estimation process.

IV. Analysis

The following section provides an in-depth analysis of the experimental results and their implications, aiming to answer the research goal outlined in the introduction.

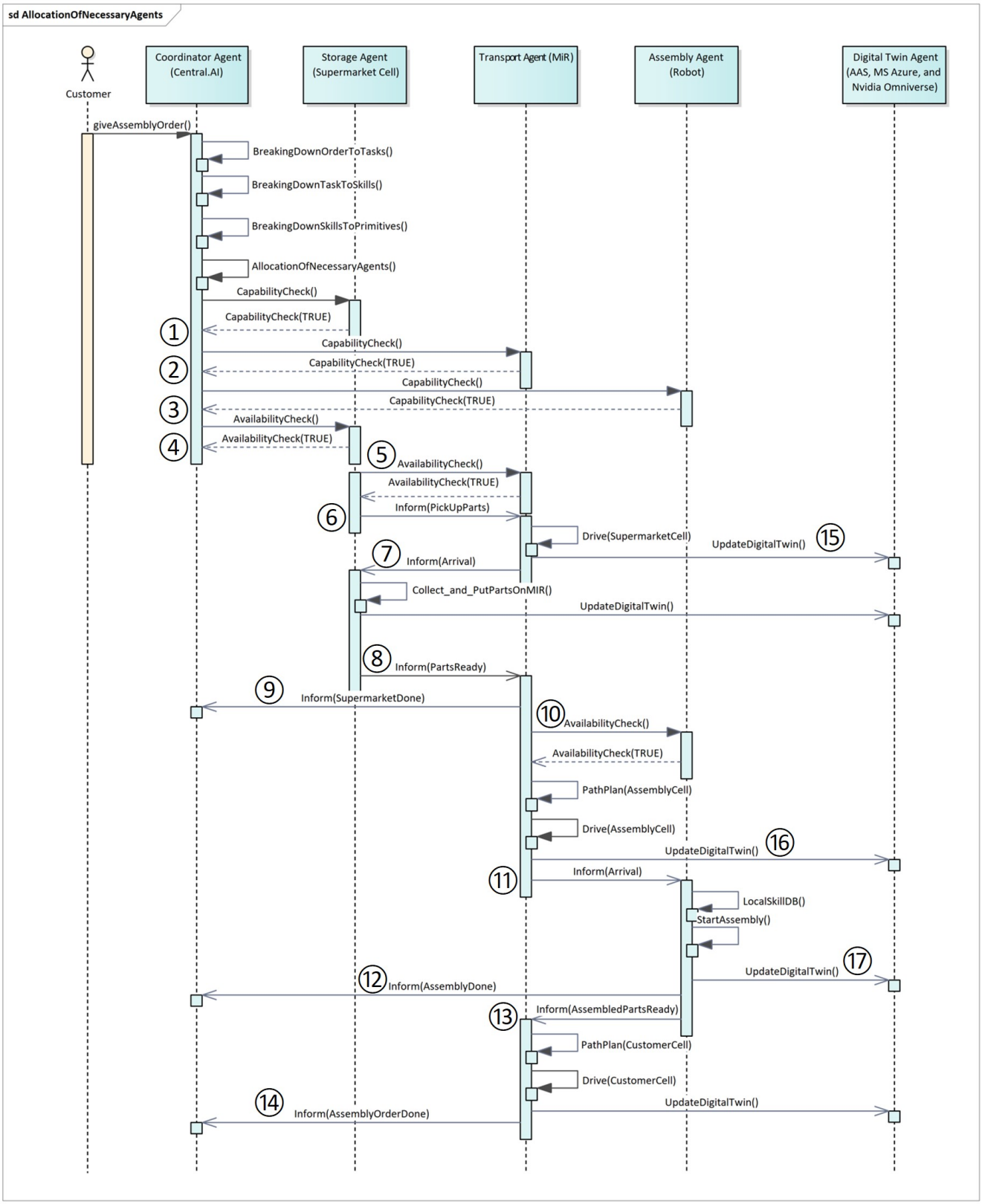

Fig. 3: UML Use Case Diagram of the assembly process of two gearbox parts in the MAS architecture

A. Decomposition of MAS communication

The analysis of the system focuses on the delay times but not the operation times. The system's decomposition grounds on the procedure of the assembly process (cp. Fig. 3). Following the sequencing of communication-based on the sender and receiver agents, communication types, message types, and concrete query content were assigned to each communication message, as shown in Tab. I. In addition to the agents' classification, the communication messages can be differentiated based on their content, including availability requests, detailed skill and primitive information queries, task completion or status reports, and position reports, such as those of the assembly or logistics agent.

All messages are of the type TCP. However, messages 15-17 go over the gateway to the AAS cloud hosted DT. The rest of the messages go over the local network. We further classify our local messages from Table I into 3 types:

1) Messages 1-3 initiate a connection and close it immediately.
2) Messages 4, 5, 10, and 11 initiate the connection but do not close it.
3) Messages 6-9 and 12-14 send the message on an already established TCP connection and terminate it.

B. Measured Delays

To obtain network measurements and determine the delay of the network, we employ application layer time-stamping at the start and end of a message. We run the applications of both the sender agent and the receiver agent on the same computer hardware. This allows us to eliminate any errors in the delay measurements arising from unsynchronized clocks between the two agents. The time stamps are taken from the CLOCK REALTIME on a Linux system. This time is synchronized with the DT via an Network Time Protocol (NTP) daemon running on the host computer. To ensure the CPUs are not overutilized, we use a computer with 16-core Intel i9 processors and 32 Gigabytes of RAM. The sender and the receiver agent have their dedicated network interface controller connected to separate PCIe slots on the computer's motherboard. We also separate these ethernet ports and the IP addresses into different network namespaces, otherwise, the Linux kernel will route the packets internally. The example network of a 1Gbps ethernet switch is connected between the sender port and receiver port. This approach guarantees consistent delay measurements, relying on the same CPU power and network hardware for all measurements. The delay measurements for each message are performed a hundred times. The distribution across the one hundred measurements for each message type is plotted in Figure 4. The mean delay times of each communication message can be found in Tab. I. These measurements also show that the total message delay for all the local messages never exceeded 1 ms pointing to its usability in real-time communication where such delays are tolerable. The delays for the messages with the DT were 3 orders of magnitude higher than the delays observed between the other 3 agents. These are plotted in Figure 5. These delay measurements also point to the fact that if cloud-based DT solutions are to be used in sync with the production environment, the communication delay might prove to be a bottleneck in terms of real-time communication.

C. Modelled Delays

We simplify the model from [4, Equation 25] by removing the delays caused due to packet losses and the slow-start phase of the TCP algorithm. The total expected message completion time for a TCP flow m is given by,

$$\mathbb{E}[C_m] = \mathbb{E}[D_{init}] + n\mathbb{E}[D_{packet}], \tag{1}$$

where $\mathbb{E}[D_{init}]$ is the time taken to complete the 3-way handshake and $\mathbb{E}[D_{packet}]$ is the time it takes to send a packet and receive its acknowledgment. Note that the FIN message to terminate a flow is sent with the last packet, and no *delayed ACK* mechanism is used as the messages are very short. The packet sizes for the messages in Table I are a maximum of 3 bytes, and the time difference in the transmission of a packet of 1 byte vs. a packet of 3 bytes over a 1Gbps link is 16 nanoseconds. Hence, for all practical purposes, we consider the delay of all packets to be the same. Round Trip Time (RTT) is the time a packet takes from one agent and back through the network. Since there is only one switch between all local agents in the network, which is not congested, this RTT is the same. We evaluate the RTT with a simple ping message readily available on all Linux OS. The mean RTT was found to be 0.274 ms. We use this value along with the message types explained in section IV-A to find the expected delay for each message from Table I. The estimated mean times for each message are shown in Table II.

D. Comparing Measured Delays to Modelled Delays

In a majority of cases, the model aligns with the measurements. All the theoretical mean delay times lie within one standard deviation of the measured value. If one were to extend the messages or message types in the production environment, the modeled delays could be summed along with the sub-task completion time to provide us with an estimate of the total task completion time. Interestingly, the models provided in the previous section were orders of magnitude off when the delay measurements of the messages with the DT from Figure 5. The mean RTT from each agent to the DT was measured to be 10 ms. This points to the fact the responses to the API calls to the AAS hosted DT take much longer internally. The modelling of the API response times of the DT is beyond the scope of the paper. However, one could rely on the measurements to obtain a rough estimate of the amount of time it takes for the completion of messages 15,16, and 17.

TABLE I: An overview of communication messages between agents

Index	Name	Agent sender	Agent receiver	Commun. protocol	Mess. type	Mess. content	Measures mean delay (ms)
Central AI schedules tasks to Supermarket cell, MiR and Robot for assembly							
1	Capability_SupCell	CenAI	SupCell	TCP	str	{"Task": "Assembly", "Product": "Gearbox", "Required resources": ["1000-122079b01000_Zahnrad_LP070.stp", "1000-122036B01000_Bolzen_LP090.stp", "1000-122323b01000_Ritzel_LP070_i10.stp"], "Required time": "60s"}	0.495
				TCP	byte	"SupCell can do task assembly."	
2	Capability_MiR	CenAI	MiR	TCP	str	{"Task": "Assembly","Product": "Gearbox","Required time": "120s"}	0.495
				TCP	byte	"MiR can do task assembly."	
3	Capability_RobAssem	CenAI	RobAssem	TCP	str	{"Task": "Assembly", "Product": "Gearbox","Required procedures":["pick up item", "assembly", "place item"], "Required time": "300s"}	0.530
				TCP	byte	"RobotAssem can do task assembly."	
MiR picks up parts from Supermarket cell							
4	Avalability_SupCell	CenAI	SupCell	TCP	byte	"Is SupCell free?"	0.555
				TCP	byte	"SupCell is free."	
5	Avalability_MiR	SupCell	MiR	TCP	byte	"Is MiR free?"	0.474
				TCP	byte	"MiR is free."	
6	SupCell_MiR_PickUpParts	SupCell	MiR	TCP	byte	"MiR picks up parts."	0.269
7	MiR_SupCell_AtSupCell	MiR	SupCell	TCP	byte	"MiR is at SupCell."	0.265
8	SupCell_MiR_PartsReady	SupCell	MiR	TCP	byte	"SupCell finishes task collect parts."	0.291
9	SupCell_CenAI_Done	SupCell	CenAI	TCP	byte	"SupCell finishes task collect parts."	0.264
MiR brings parts to Robot assembly and let it do the task assembly							
10	Avalability_RobAssem	CenAI	RobAssem	TCP	byte	"Is RobAssem free?"	0.605
				TCP	byte	"RobAssem is free."	
11	MiR_RobAssem_AtRobotAssem	MiR	RobAssem	TCP	byte	"MiR is at robot assembly"	0.303
12	RobAssem_CenAI_Done	RobAssem	CenAI	TCP	byte	"RobAssem finishes task assembly."	0.325
13	RobAssem_MiR_Done	RobAssem	MiR	TCP	byte	"MiR picks up assembled parts."	0.304
14	MiR_CenAI_Delivery	MiR	CenAI	TCP	byte	"MiR finishes task delivery."	0.309
Supermarket cell, MiR and Robobt assembly updates their digital twins							
15*	SupCell_UpdateDT	SupCell	DT	HTTP	json	"[{'op': 'replace','joint0': 'radius','value': 0.7899125}, ..., 'position_theta': 'radius','value': 0.2145814}]"	917.352
16	MiR_UpdateDT	MiR	DT	HTTP	json	"[{'op': 'replace','x': 'm','value': 10.688876152038574}, 'op': 'replace','y': 'm','value': 31.457216262817383}, 'op': 'replace','theta': 'degree','value': 101.11448669433594}]" "	815.126
17*	RobAssem_UpdateDT	RobAssem	DT	HTTP	json	"[{'op': 'replace','joint0': 'radius','value': 0.1254187349436}, ..., 'position_theta': 'radius','value': 0.3654874215457}]"	933.470

∗ The message content contains the information of joints, gripper and position of the robot, which are 12 degrees of freedom.

TABLE II: Estimated delay times for message types from Table I

Messages	$\mathbb{E}[C_m]$	**Delay Value (ms)**
1-3	$2 \cdot \mathbb{E}[RTT]$	0.548
4,5,10,11	$2 \cdot \mathbb{E}[RTT]$	0.548
6-9, 12-14	$1 \cdot \mathbb{E}[RTT]$	0.274

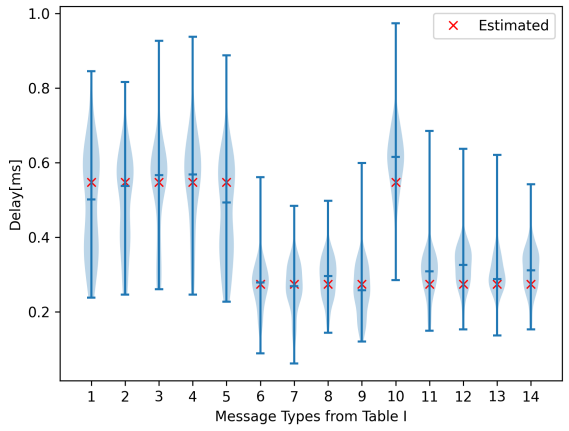

Fig. 4: The measured communication delay for all local messages. The measured mean agrees in most cases with the estimated mean provided in Table II

.

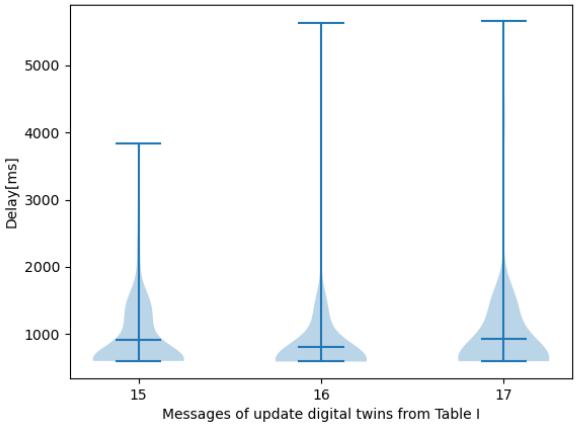

Fig. 5: The measured communication delay for Digital Twin related messages. These delays are much higher as they are not present in the local network as well as the application layer response time to the API calls to the DT are unknown.

V. Discussion

According to the assumptions made about the underlying network and the industrial setup, we assumed that the network traffic is carried over COTS hardware, each device with an IP address and the network conditions remain stable over time. The setup includes a storage agent, a path planning agent, a logistics agent, an assembly agent, and a digital twin agent. The system is not restricted by either hard or soft real-time boundaries, and the focus is on examining the real-time capabilities of the system, including the investigation of mean message delivery delays associated with each of the setup's communication channels. The study examines an industrial assembly use case of two gearbox parts by Wittenstein for a customer order. The authors apply DSL4Production to the assembly use case and use task decomposition, agent allocation, and digital twin tracking to assess the automated processing of orders, delays, and resource utilization. The authors obtain delay times through measurements, and they simplify the estimation process by relying on mathematical equations once they verify that their model for the system's delay aligns with the measured delay times.

The test setup still contains only a simple network with one ethernet switch in between any two local agents. The validity of the models over larger local networks needs to be tested. Variations in the network conditions such as congestion and packet drops can be modelled using methods such as network calculus (NC) [14]. While this would make the delay model much more complex, it would also be more robust in a real industrial deployment. The duration of delays is considerably influenced by the hardware specifications chosen, e.g., in our case by the protracted processing times of the MiR robot and its corresponding fleet manager for status and location information. Additionally, communication with the cloud-based digital twin exhibits extended delay times, which do not factor in the DT processing times of up to 200 ms. While the delays observed in the digital twin communication may initially appear concerning, it is important to consider the broader context and purpose of using a digital twin in the production scenario. The primary purpose of incorporating a digital twin in the system is to enable assessment and evaluation of automated processing of orders, delays, and resource utilization. Despite the delays observed in the digital twin communication, it still offers significant value in terms of system analysis, optimization, and decision-making. Hence, to facilitate a comprehensive modeling and pre-calculation for DSL4Production before conducting measurements or implementing the system, it is necessary to have more transparent and comprehensive information about hardware delays (e.g., from industrial data sheets, including sensors and actuators delay information).

VI. Conclusion and Outlook

In this section, we will conclude the study's findings and provide a outlook based on our results.

A. Conclusion

The methodology of the paper involves assumptions about the network, fundamental assumptions for an industrial setup, and a detailed description of a use case involving an industrial assembly scenario. The study applies DSL4Production on the assembly use case to assess automated processing of orders, delays, and resource utilization. The delay times are obtained through measurements, and mathematical equations are used to make further assumptions about delays. The focus is on examining the real-time capabilities of the system and investigating the mean delays associated with each of the setup's communication channels. We decomposed the system based on the procedure of the assembly process and classify communication messages based on sender and receiver agents, communication types, message types, and concrete query content. We performed measurements of the network delay of each communication message using and obtained the mean, maximum, and minimum delay times. The grouping of the violin charts are derived based on communication messages and their content for analysis. Modelled delays were also compared to the measured ones, and the mean delaysfrom the model were found to approximate the measured mean values in most scenarios. Based on the analysis, thus we were able to determine estimated delay times that can be generalized and utilized for future DSL4Production models.

B. Outlook

Further research is necessary to advance the field of communication and coordination processes in MAS for future production scenarios. Our study proposes the extension of our findings to repetitive modules within production systems, as well as an exploration of the impact of different network loads on delay times. Additionally, the application of machine learning approaches could be investigated for optimizing MAS communication in production scenarios. A more efficient approach for achieving hard and soft real-time capabilities in complex systems could be implemented by utilizing modeled delay times prior to implementation.

VII. Acknowledgement

The authors want to thank the Bavarian State Ministry for Economic Affairs, Regional Development and Energy (StMWi) Lighthouse Initiative KI.FABRIK, (Phase 1: Infrastructure and R&D program, grant no. DIK0249), for the funding.

References

[1] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074

[2] G. Luan, "Estimating tcp flow completion time distributions," *Journal of Communications and Networks*, vol. 21, no. 1, pp. 61–68, 2019.

[3] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The macroscopic behavior of the tcp congestion avoidance algorithm," *ACM SIGCOMM Computer Communication Review*, vol. 27, no. 3, pp. 67–82, 1997.

[4] N. Cardwell, S. Savage, and T. Anderson, "Modeling tcp latency," in *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No. 00CH37064)*, vol. 3. IEEE, 2000, pp. 1742–1751.

[5] B. Vogel-Heuser, M. Seitz, L. A. Cruz Salazar, F. Gehlhoff, A. Dogan, and A. Fay, "Multi-agent systems to enable industry 4.0," pp. 445–458, 2020.

[6] I. Kovalenko, D. Tilbury, and K. Barton, "The model-based product agent: A control oriented architecture for intelligent products in multi-agent manufacturing systems," *Control Engineering Practice*, vol. 86, pp. 105–117, 2019.

[7] M. M. Madden, *Challenges Using the Linux Network Stack for Real-Time Communication*. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2019-0503

[8] B. Vogel-Heuser, M. Zimmermann, K. Stahl, K. Land, F. Ocker, S. Rötzer, S. Landler, and M. Otto, "Current challenges in the design of drives for robot-like systems," pp. 1923–1928, 2020.

[9] D. Hujo, B. Vogel-Heuser, and L. Ribeiro, "Toward a graphical modeling tool for response-time requirements based on soft and hard real-time capabilities in industrial cyber-physical systems," *IEEE Journal of Emerging and Selected Topics in Industrial Electronics*, vol. 3, no. 1, pp. 13–22, 2021.

[10] L. Ribeiro and M. Hochwallner, "Time-related constraints in administration shell design within cyber-physical production systems," in *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, vol. 1. IEEE, 2019, pp. 1564–1569.

[11] A. Van Bemten, N. Derić, A. Varasteh, S. Schmid, C. Mas-Machuca, A. Blenk, and W. Kellerer, "Chameleon: Predictable latency and high utilization with queue-aware and adaptive source routing," ser. CoNEXT '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 451–465. [Online]. Available: https://doi.org/10.1145/3386367.3432879

[12] S. Ha, I. Rhee, and L. Xu, "Cubic: A new tcp-friendly high-speed tcp variant," vol. 42, no. 5, p. 64–74, jul 2008. [Online]. Available: https://doi.org/10.1145/1400097.1400105

[13] J. Höfgen, B. Vogel-Heuser, F. Bi, J. Zhao, A. Kraft, B. Vojanec, and T. Markert, "Architecture of a versatile digital twin with socket-based communication and azure dt," p. 8, 2023.

[14] J.-Y. Le Boudec and P. Thiran, *Network calculus: a theory of deterministic queuing systems for the internet*. Springer, 2001.