# PHYSICAL IMPLEMENTATION OF A DISTRIBUTED, AGENT-BASED CONTROL FOR FLUID SYSTEMS USING OPC UA

Tobias C. Meck[1]*, Oscar L. Lefemmine[1], Kevin T. Logan[1], Peter F. Pelz[1]

[1]*Chair of Fluid Systems, Technische Universität Darmstadt, Otto-Berndt-Str. 2, 64287 Darmstadt*

* Corresponding author: Tel.: +49 6151 1627126; E-mail address: tobias.meck@tu-darmstadt.de

## ABSTRACT

Conventional control strategies for fluid systems often rely on local control of the system's components, like pumps and valves. Here, communication between the control units is non-existent, which can have a negative impact on the energy efficiency. Distributed control is a promising alternative where so-called agents are assigned to components. These agents are autonomous units with individual goals. They can perceive and influence their environment through sensors and actuators. Furthermore, they are able to share information with each other. This leads to an increased energy efficiency while maintaining the positive aspects of local control, such as a low implementation effort and high robustness. The concrete methods are the subject of current research and are typically only verified in simulations. For a thorough evaluation and broad acceptance in industry, an assessment of the methods when facing real systems is crucial.

In this work, we therefore focus on the physical implementation of distributed control. We examine a simple fluid system with a centrifugal pump and a valve. A valve agent measures its volumetric flow rate and communicates this information to a pump agent via Wi-Fi and OPC UA. The pump agent has the goal of achieving a target flow by using a PI controller and adjusting the rotational speed. The results are promising and easily scalable to more complex systems and control methods.

***Keywords:*** Multi-agent systems, OPC UA, Microcontrollers, Control, Robustness

## 1. INTRODUCTION

When it comes to the energy efficiency of fluid systems, such as the water supply in buildings or whole cities, expectations usually do not meet reality. Society demands special attention to be paid to the matter. This is once again confirmed when looking at the recently revised Energy Efficiency Directive 2023/1791 from the European Union, which puts "Energy Efficiency First" [1].

Still, conventional control strategies for fluid systems are usually not tailored for efficiency. They use *local control* of components, mostly valves, to reach certain set points for the pressure or the volumetric flow rate, e.g., using PID controllers. In this way, the introduced energy from pumps gets dissipated due to throttling losses.

These losses can potentially be avoided if additional information of the system, such as the current state or the topology, is used [2]. This information is used to decide if it is possible to reduce the introduced energy, e.g., by reducing the rotational speed of pumps, rather than throttling excessive energy, e.g., by shutting down valves. In *central control*, the information from all components and available sensors is collected and possibly enriched by a system model in a central controller. Using techniques from mathematical optimization, it is then even possible to calculate optimal operating modes with respect to efficiency, as shown in [3].

While this method theoretically offers the greatest possible improvement in this dimension, Logan et al. [2] list further requirements to control approaches that have to be taken into account. Looking at fluid systems as a vital part of critical infrastructure or production systems in industry, it becomes clear that the robustness of the control is of great importance. Perturbations in the operating phase, such as unforeseen load cases or wear of components up to component failures, should have a minimal influence on the overall function of the system. Central control systems can cope with slight perturbations, but pose a single point of failure and often rely on sufficiently accurate models. Local control, on the other hand, can in most cases still maintain a basic functionality, even if failures occur.

Another aspect worth considering is the implementation effort of control systems. Fluid systems are getting increasingly complex and individual. Modelling is therefore cumbersome and accompanied with various uncertainties, for instance regarding the calculation of pressure losses. Over time, there might also be changes or extensions of the system, requiring further adaptions. These reasons serve as a possible explanation why local control is still preferred in a lot of applications.

Aiming at combining the advantages of local and central control, Logan et al. [2] focus on *distributed control*. Here, the local controllers are designed as agents, which can access the sensors and actuators of the connected components and thus perceive and influence their environment. In contrast to local control, they are also able to exchange information through communication. Furthermore, they have individual goals. For a pump agent, this might be the desire to minimize the pumps input power.

The concrete methods differ with regards to the decision rules of the agents, i.e., how they manipulate the actuators based on measurements and acquired information due to communication to reach their goals. The authors compare approaches from optimization theory, machine learning and game theory in simulations. The latter approach, referred to as *market mechanism*, is particularly promising. Here, a virtual budget is assigned to the agents, which is used to trade volumetric flow rate guarantees. Similar to a real market, one expects to reach an efficient allocation without explicit system knowledge. As this method therefore does not rely on system models or extensive training, it is highly flexible and transparent, which is beneficial for the acceptance in practical applications.

Regarding the acceptance, it is also crucial to proof the validity of the concepts using experiments and real systems. The step from simulations to experiments often offers additional insights and is necessary for an overall evaluation. A first step towards a validation of the methods controlling real systems was presented in [4]. While the simulation of the fluid system was exchanged by the real counterpart, the agent system was still simulated on a central machine. Nevertheless, the results show a significant increase of the energy consumption compared to the simulation, which does not consider dynamics, such as limited opening rates of valves. Replacing the simulation of the agent system introduces additional complexity, such as limited computing resources of edge devices and communication latencies. In this work, we focus on establishing the necessary framework that enables an all-embracing validation of distributed control methods for fluid systems. We apply the results to a minimal example of distributed control that is outlined in the next section.

## 2. CONTROL TASK

As a model system, we consider the water supply of a residential building. In order to have a sufficient pressure on the higher floors, it is necessary to install pumps, which are usually placed in the basement of the building. To control the volumetric flow rate, the different floors are equipped with valves.

This model system can be scaled down to a test rig, that is shown in **Figure 1**. Water is pumped by a booster station from a tank to five floors, which are equipped with valves and flow sensors. The available pumps are centrifugal pumps used in heating applications. They possess integrated frequency converters which allow an adjustment of the rotational speed $n$. It is possible to measure

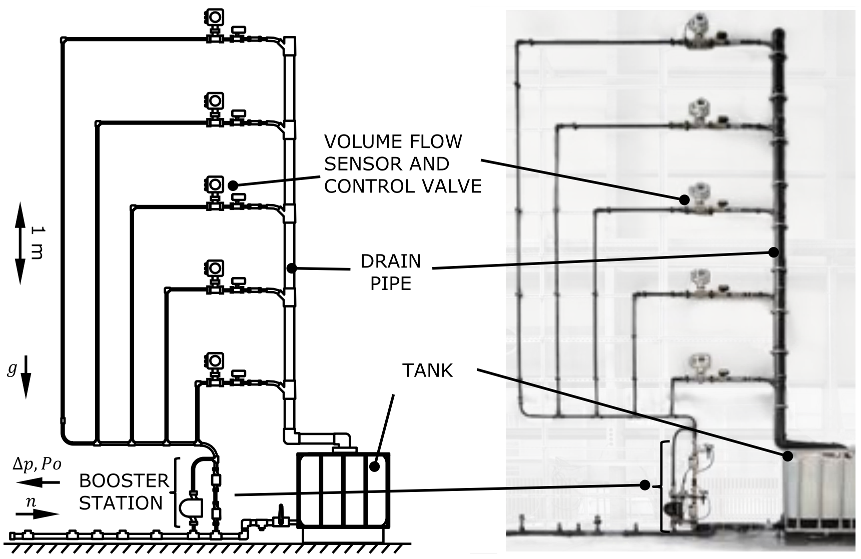


**Figure 1:** Schematic (left) and photograph (right) of the modular test rig [5].

the pressure difference $\Delta p$ and the electrical power consumption $Po$ of each pump. From the different floors, the water then returns to the tank through a drain pipe. Because of the modular design of the test rig, it is possible to realise different topologies. For a detailed description of the test rig we refer to [5].

In this work, we only consider the simple case where a single pump is placed in the booster station and all valves except for the valve in the first floor are shut. The control task is to fulfil a target flow rate by using a PI controller, which is assigned to the pump. This serves as a minimal pump agent. The counterpart of the pump agent is the combination of the valve and the flow sensor on the first floor, which constitutes a valve agent. As the actual flow rate is not directly accessible to the pump agent, it needs to acquire this information through communicating with the valve agent.

## 3. DEVELOPED FRAMEWORK

### 3.1. Hardware

To be able to realize the proposed control task, it is first and foremost necessary to physically represent the agents with appropriate hardware. For this purpose, ESP32-S3-DevKitC-1 [6] development boards are used as a basis, which combine an ESP32-S3 microcontroller with additional peripherals to form a programmable PCB board. This allows to program and process the internal logic of the agents. The microcontrollers are equipped with 512 KB of SRAM, up to 16 MB of Flash and have a maximum clock speed of 240 MHz. Additionally, they include Wi-Fi capabilities, which can be used for the physical layer of the agent communication. However, as this requires an additional access point which poses a single point of failure, an Ethernet interface is connected to the development board which can be used as an alternative. To be able to connect the sensors and actuators of the agents to the general-purpose input/output (GPIO) pins of the microcontroller, further components, like connectors, voltage dividers and Digital-to-Analog-Converter (DAC) modules, are installed. With this, input signals from 0-10 V/ 4-20 mA and output signals from 0-10 V are possible. The final hardware is shown in **Figure 2**. A distinction between pump agents and valve agents is done solely on the software-side, which enables a straightforward addition of other agent types.

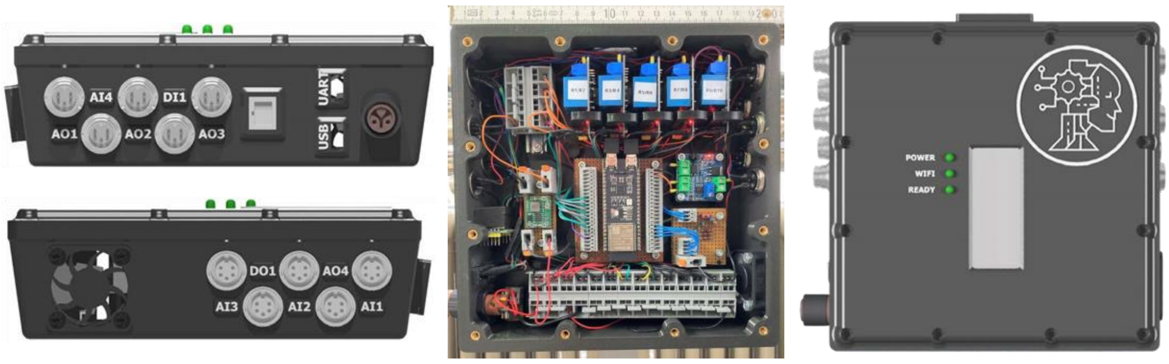


**Figure 2:** Side view (left) and top view with open (middle) and closed lid (right) of the hardware implementation of the agents

## 3.2. Software

The software of the agents has to perform different tasks. On the one hand, the control algorithms have to be carried out, which are specific to the subtype of distributed control being considered. For the example in the scope of this work, this reduces to a discrete PI controller for the pump agent. Associated with this are also the sensor readings and the actuator control.

On the other hand, the software needs to manage the connection and communication of the agents. This normally involves additional aspects, such as the discovery of other agents in the network and their topological relationship, i.e., if another agent is connected upstream, downstream or in parallel. We consider this information given. As a possible extension, network discovery can easily be achieved by performing mDNS or ARP scans.

*OPC UA*

For the communication of the agents, we employ the platform independent machine-to-machine standard OPC UA (*Open Platform Communication Unified Architecture*) [7]. OPC UA offers a wide range of features, such as security settings, that partially extend the scope of our work. We therefore refer to the official reference [7] for additional information.

In OPC UA, not only the raw data transport is considered, but rather the transport of information through an extensive information modelling framework. In essence, this information model is a graph consisting of nodes and references between them [8]. Different node types are available, e.g., object nodes and variable nodes. Object nodes represent physical or abstract elements of the system and may contain further nodes. Variable nodes represent values (e.g., sensor data) that can be read or written and have an associated data type. The nodes have different attributes, such as unique node ids, descriptions or time stamps [7].

For the lower-level transport layer, different protocols are available, of which the binary TCP-based protocol used here is the most common [8]. Depending on the chosen protocol, two different communication models, namely client-server and publish-subscribe (PubSub), can be applied. We focus on the client-server communication model, although OPC UA PubSub is a promising alternative for applications with very high latency requirements, as demonstrated in [9].

In the client-server-model, servers offer services to clients, such as reading or writing of variable nodes values. The question if a particular component should be implemented as a client or server depends greatly on the use case. As advocated by Rinaldi [10], assigning a server *and* clients to components might be beneficial. In our use case, we implement every agent that has access to data that needs to be shared, e.g., from sensors, as a server. If an agent needs to access data from other agents, it additionally needs to implement a client for each of those agents.

For the servers, a set of nodes has to be defined. The valve agent's server contains a variable node for the volumetric flow rate, which gets updated through readings of the respective input. An additional variable node is the valve opening. Writing to this node allows clients to directly influence the valve agent's output. Similarly, the pump agent's server contains nodes for the pressure difference, the rotational speed, the on/off state and the volumetric flow rate. The latter is updated whenever the valve agent communicates new values. Additionally, the pump agent's server contains a PI controller object node, which in turn contains variable nodes for the set point and the gains as well as an activity node that can be used to enable and deactivate the control. As described above, these node sets constitute basic information models.

For the data exchange, we make use of a subscription concept defined by the OPC UA standard. With this, it is possible to make servers send out notifications at a predefined rate, if the value of a variable node has changed. This especially facilitates the reading of constantly changing values. In our example, the pump agent subscribes to the value change of the volumetric flow rate node of the valve agent's OPC UA server at a publishing rate of 10 Hz.

To implement the OPC UA concept, the open source OPC UA stack open62541 [8] is used. The stack is written in the C programming language and offers support for FreeRTOS [11], which is a well-established operating system for ESP32 microcontrollers. As a basis for our software, a GitHub project [12] by GitHub user *cmbahadir* was used, which implements a simple OPC UA server based on the open62541 stack. Additionally, the project implements the necessary Wi-Fi connection and a time synchronisation via a NTP server.

### *Control Dashboard*

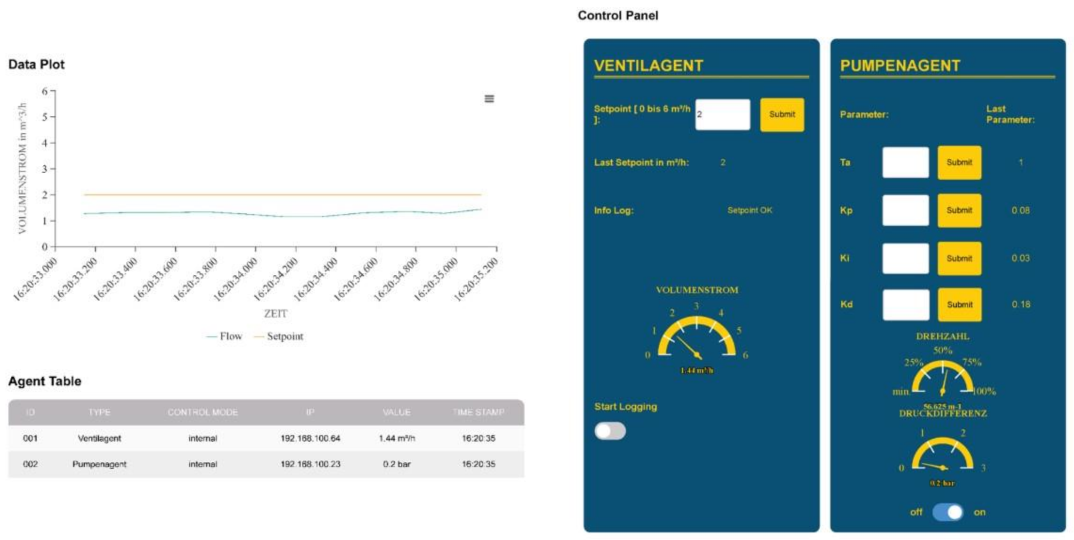
For the purpose of logging and supervising the control process, which includes visualisation, starting the PI control or manually adjusting the pump's rotational speed, a central computer is used. As for instance the PI control could also be enabled directly from the start in the microcontroller program, this does not impair the idea of a distributed control but rather simplifies the conduction of experiments.
The central computer hosts an HTTP server based on the Python framework Flask [13] with a website based on the JavaScript library Highcharts [14] shown in **Figure 3**. This acts as a control dashboard. Two additional OPC UA clients for the valve and pump agents, based on the Python library opcua-asyncio [15], share data with the HTTP server. In this way, reading information (e.g., the current volumetric flow rate) and writing information (e.g., the gains of the PI controller) of the OPC UA servers is possible from any computer or smart device in the same network without additional software. For the visualization of sensor values, the aforementioned subscription concept is again applied.

The final framework is summarized in **Figure 4**.

## 4. RESULTS

To prove the functionality of the presented framework, the control task described in section 2 needs to be assessed. For this purpose, two different experiments are conducted. In the first experiment, the pump is set to its maximum rotational speed before starting the PI control. As a set point, a volumetric flow rate of 1 $m^3/h$ is specified. In a second experiment, the pump is set to its minimum rotational speed before starting the PI control to reach a set point of 3 $m^3/h$.

The controller is parametrized for the first set point starting with the well-known Ziegler-Nichols method with a critical gain of 0.38 and a critical period of 6.7 s. Afterwards, the proportional and


Data Plot

Agent Table

| ID | TYPE | CONTROL MODE | IP | VALUE | TIME STAMP |
|---|---|---|---|---|---|
| 001 | Ventilagent | internal | 192.168.100.64 | 1.44 m³/h | 16:20:35 |
| 002 | Pumpenagent | internal | 192.168.100.23 | 0.2 bar | 16:20:35 |

Control Panel

VENTILAGENT

Setpoint [ 0 bis 6 m³/h ]: 2 Submit

Last Setpoint in m³/h: 2

Info Log: Setpoint OK

Start Logging

PUMPENAGENT

| Parameter: | | | Last Parameter: |
|---|---|---|---|
| Ta | | Submit | 1 |
| Kp | | Submit | 0.08 |
| Ki | | Submit | 0.03 |
| Kd | | Submit | 0.18 |

**Figure 3:** HTML page based on the JavaScript library Highcharts [14], which serves as a control dashboard.

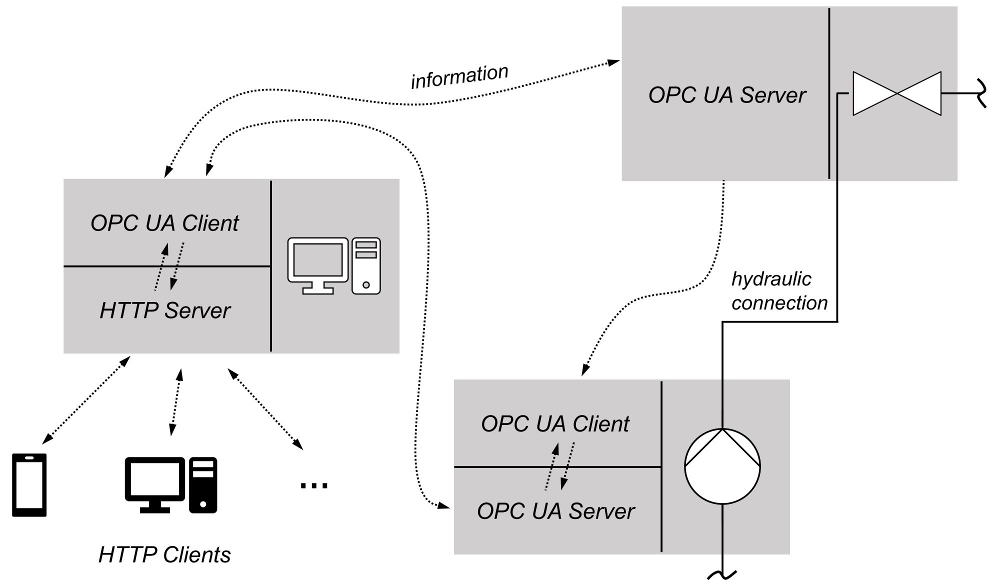


**Figure 4:** OPC UA-based distributed control framework.

integral gain are slightly adjusted for a better controller performance. The sampling rate is set to 2 Hz. As shown in **Figure 5**, the control starts after approximately 20 s and is able to reach the set points quickly. However, the volumetric flow rate signal from the valve agent is highly noisy, which leads to an even higher fluctuation around the set point due to the PI control. Worth noting is also the apparent increase of the volumetric flow rate in the first experiment before the set point is reached. A possible explanation is that the PI controller, having a different subscription than the logging computer, receives a value below the set point, causing an increase of the rotational speed of the pump.

Looking at the sample mean after sufficient time, one can conclude that the control at least on average reaches the set point with sufficient precision. **Table 1** shows the numerical values of the mean and the standard deviation of the volumetric flow rate signal.

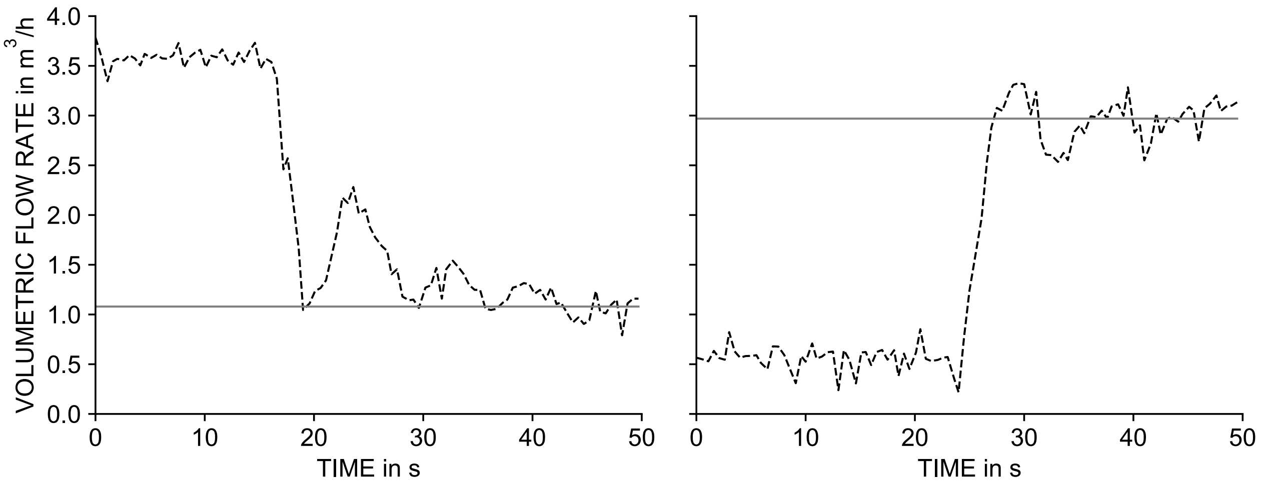


**Figure 5:** PI control starting from maximum speed with a setpoint of 1 $m^3/h$ (left) and from minimum speed with a setpoint of 3 $m^3/h$ (right). The gray line indicates the mean volumetric flow rate after 40 s. The flow measurements are taken from the valve agent's OPC UA server.

**Table 1:** Sample statistics of the flow rate in $m^3/h$ (100 samples per experiment at 2 Hz).

| Experiment | Set point | Mean, last 10 seconds | (Corrected) standard deviation, first 10 seconds | (Corrected) standard deviation, last 10 seconds |
|---|---|---|---|---|
| 1 | 1 | 1.0795 | 0.086 | 0.126 |
| 2 | 3 | 2.969 | 0.098 | 0.161 |

Another important aspect of the presented framework is the performance of the communication. As defined by the OPC UA standard, a *sourceTimestamp* is applied by an OPC server at the data source to a variable value at each change, and a *serverTimestamp* denotes the time when a server receives the value. The difference between these two timestamps can be used to approximate the overall latency of the communication. In our example, the valve agent's OPC UA server assigns a sourceTimestamp when performing a reading from the flow sensor. When the pump agent's OPC UA client receives this value through the subscription, the variable value gets written to the corresponding variable node alongside its sourceTimestamp. The serverTimestamp, on the other hand, gets set to the current time. For a meaningful comparison, the clocks of both agents need to be sufficiently synchronized, which is achieved by a NTP server. Still, slight clock drifts might be possible. It is also important to note that additional computation steps of the microcontrollers might further increase the total latency. For both experiments, the resulting differences are shown in **Figure 6** and **Table 2**. While the mean and median latency are acceptable, especially considering the connection over Wi-Fi, large outliers can be detected. Still, the results validate the use of a sample rate of the PI controller and the logging of 2 Hz.

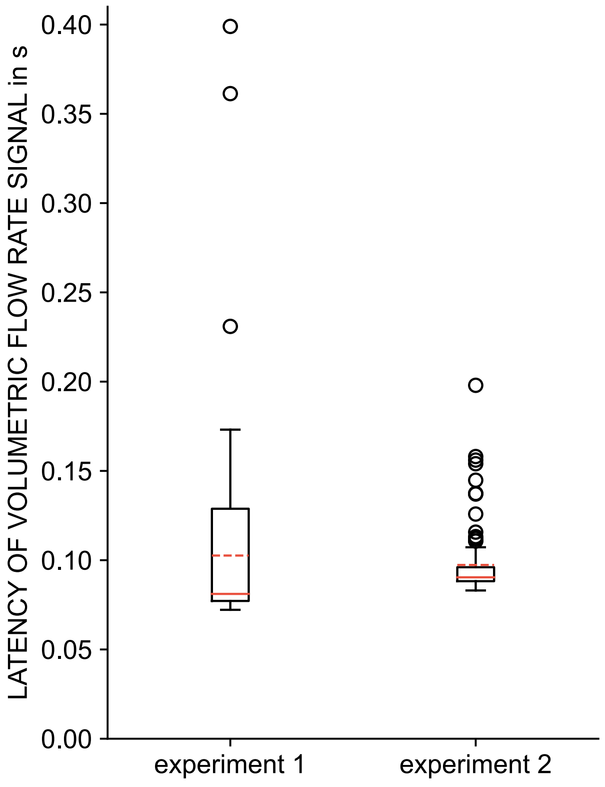


**Figure 6:** Boxplots of the difference between server and sourceTimestamps of the volumetric flow rate as a measure of latency. Sample size = 100. Maximum whisker length = 1.5 · IQR (Interquartile range). The dashed lines represent the mean and the solid lines the median of the data.

**Table 2:** Sample statistics of the difference between server and sourceTimestamps of the volumetric flow rate as a measure of latency in seconds. Sample size = 100.

| Experiment | Mean | Median | Maximum | (Corrected) standard deviation |
|---|---|---|---|---|
| 1 | 0.103 | 0.081 | 0.399 | 0.048 |
| 2 | 0.097 | 0.090 | 0.198 | 0.019 |

## 5. CONCLUSION

In this work, we introduced a framework for the validation of distributed control algorithms. To physically represent the agents, microcontrollers with appropriate peripherals are used. The communication between the agents is achieved by Wi-Fi and OPC UA. In this context, a OPC UA server is assigned to each agent. Also, an additional OPC UA client is assigned to an agent for each agent that it needs to communicate with. The additional integration of a central computer facilitates logging and supervising of the distributed control. The framework was assessed using a test rig, which represents the water distribution in a residential building. In the test setup, a pump agent employs a PI controller to reach a target volumetric flow rate. As the pump does not have direct access to a flow rate sensor, it needs to acquire the information from an upstream valve agent. This is achieved by subscribing to the value changes of the volumetric flow rate node in the valve agent's OPC UA server address space. Though the results show a high fluctuation of the sensor readings and harsh outliers in the communication latency, the control task can in general be achieved satisfactorily.

To tackle the latency of the communication, Ethernet connections are a natural alternative to Wi-Fi. These in turn pose additional questions, e.g., regarding data transmission topologies.

As pointed out by the manufacturer of the microcontrollers, the integrated ADCs are sensitive to noise, which explains the high fluctuations of the volumetric flow rate readings. To mitigate this issue, the manufacturer suggests the usage of bypass capacitors, which could be examined in a next step.

The developed framework is readily scalable to more sophisticated distributed control tasks. The server-client principle of the agents allows new agents to enter the network in a simple "plug-and-play" manner. However, further considerations have to be taken into account. For one, additional clients pose additional memory requirements and computational complexity for the microcontrollers. This might be unfeasible for systems, where an agent has to be able to communicate with a large number of peers. In such settings, OPC UA PubSub is worth considering. OPC UA PubSub is especially relevant for one-to-many, many-to-one or many-to-many communication settings, but does not reliably ensure the data transmission. Looking at the different variants of distributed control, particularly the market mechanism, a reliable one-to-one communication might however be important.

Especially the more sophisticated distributed control methods require richer information models than those that are considered in this work. In a first step, predefined node sets from the OPC Foundation might be used. Further research could aim at developing an information model to fully describe agents, irrespective of their type. Taking the market mechanism as an example again, the information model should also be capable of mapping complex communication patterns, such as negotiations, to make the framework universally applicable.

## ACKNOWLEDGMENT

The authors would like to thank Manuel Rexer and Niklas Puff for their valuable input on designing the hardware solution. Results were obtained in project no. 22514N/1, funded by the German Federal Ministry for Economic Affairs and Climate Action (BMWK) approved by the Arbeitsgemeinschaft industrieller Forschungsvereinigungen "Otto-von-Guericke" e.V. (AiF).

## REFERENCES

[1] European Union (2023) Directive (EU) 2023/1791 of the European Parliament and of the Council

[2] Logan KT, Stürmer JM, Müller TM et al. (2022) Comparing Approaches to Distributed Control of Fluid Systems based on Multi-Agent Systems

[3] Müller TM, Knoche C, Pelz PF (2022) From Design to Operation: Mixed-Integer Model Predictive Control Applied to a Pumping System. In: Trautmann N, Gnägi M (eds) Operations Research Proceedings 2021. Springer International Publishing, Cham, pp 318–324

[4] Logan KT, Stürmer M, Müller TM et al. (2023) Multi-Agent Control of Fuid Systems – Comparison of Approaches.

[5] Müller TM, Leise P, Lorenz I-S et al. (2021) Optimization and validation of pumping system design and operation for water supply in high-rise buildings. Optim Eng 22:643–686. https://doi.org/10.1007/s11081-020-09553-4

[6] Espressif Systems ESP32-S3-DevKitC-1. https://docs.espressif.com/projects/esp-idf/en/latest/esp32s3/hw-reference/esp32s3/user-guide-devkitc-1.html. Accessed 03 Feb 2024

[7] OPC Foundation OPC Unified Architecture. https://reference.opcfoundation.org/. Accessed 04 Feb 2024

[8] open62541 open62541 documentation. https://www.open62541.org/doc/master/. Accessed 04 Feb 2024

[9] Pfrommer J, Ebner A, Ravikumar S et al. (2018) Open Source OPC UA PubSub Over TSN for Realtime Industrial Communication. In: 2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA).

[10] John S. Rinaldi OPC UA Client vs. Server. https://www.rtautomation.com/rtas-blog/opc-ua-client-vs-server/. Accessed 04 Feb 2024

[11] Amazon Web Services FreeRTOS. https://www.freertos.org/fr-content-src/uploads/2018/07/FreeRTOS_Reference_Manual_V10.0.0.pdf. Accessed 05 Feb 2024

[12] cmbahadir opcua-esp32. https://github.com/cmbahadir/opcua-esp32. Accessed 05 Feb 2024

[13] Pallets Flask. https://flask.palletsprojects.com/en/3.0.x/. Accessed 05 Feb 2024

[14] Highsoft AS Highcharts. https://www.highcharts.com/. Accessed 05 Feb 2024

[15] FreeOpcUa opcua-asyncio. https://github.com/FreeOpcUa/opcua-asyncio/?tab=readme-ov-file. Accessed 05 Feb 2024