Plug and Produce for Industry 4.0 using Software-defined Networking and OPC UA

Basavaraj Madiwalar, Ben Schneider, Stefan Profanter

***Abstract*—The manufacturers are in quest for flexible and agile production facilities capable of accommodating changes to product specification. The need for flexible production facilities is stemming from the desire for customized products and fluctuating market trends. Industry 4.0 impels for adaptable manufacturing plants by employing intelligent devices and advanced communication technologies. The complexity of the configuration process determines the adaptability of production facilities to accommodate changes to the production process.**

We propose a systematic integration process and multi-level production system using Software-defined Networking (SDN) and OPC Unified Architecture (OPC UA) to reduce the configuration complexity to a Plug and Produce level. OPC UA, as a service-oriented middleware, provides the tool-set for semantic modeling and automatic device discovery. However, due to the multicast nature of the OPC UA discovery mechanism, the existing approaches require intelligence at the device level to select the desired device to connect to. In contrast, our proposed solution shifts the intelligence to a centralized SDN controller to route multicast traffic to facilitate device discovery. The combination of an SDN controller and OPC UA discovery enables the integration of new devices by adding more intelligence to the device discovery.

I. Introduction

The increasing demand for customized products and shorter product life cycles is shifting the focus of the manufacturing process from mass production towards mass customization [1]. The manufacturing facilities should be adaptable to accommodate changes to the product specification. Therefore, the search for ideas to create flexible manufacturing facilities is gathering greater interest.

The flexibility aspect of a manufacturing system largely depends on the configuration effort needed to commission and restructure the production facility. The current production facilities employ proprietary automation technologies and communication protocols optimized for high performance with a narrow area of applications. As a consequence, the current highly automated production facilities deemed inflexible due to high configuration effort required to set up the production facility. In a flexible manufacturing facility, the integration of new devices shall be comparable to hot-plugin of USB devices to a PC system. To achieve such a level of Plug and Produce, the system should detect the newly plugged device and configure it to perform the production task. The existing research work provides solutions for automated device discovery or device configuration [1, 2]. However, they don't offer a common solution for both the device discovery and configuration process. In this work, we make use of open-source communication technologies such as Software-defined Networking (SDN) and Open Platform Communications Unified Architecture (OPC UA) to define a multi-level Plug and Produce system which detects and configures new devices.

The proposed multi-level Plug and Produce system utilizes the SDN flow programmability and the device discovery features of OPC UA to reduce the configuration effort. In our proposed solution, the process of integrating new devices is split into two phases: Discovery and Configuration. In the Discovery phase, a new device discovers and registers with its control entity to receive control parameters. OPC UA facilitates the Discovery phase by defining the local discovery servers utilizing multicast DNS (mDNS) and DNS Service Discovery (DNS-SD) protocols. The combination of mDNS and DNS-SD is primarily used in the enterprise environment for automatic device discovery. Even though mDNS offers automated device discovery, we showed in our previous work that selecting the correct controlling device requires pre-configured information [3]. The pre-configuration requirement defeats the purpose of Plug and Produce systems.

As a solution, we propose to make use of the SDN controller as an intelligent network control device to filter and forward mDNS messages from the correct control device. In the configuration stage, the registered device is configured with the help of higher level manufacturing services, to perform the production task. Thereby, both SDN and OPC UA reduces the configuration effort to the Plug and Produce level. Apart from the reduced configuration effort, the SDN controller also helps to reduce the mDNS multicast traffic in the network. Further, the SDN can help to realize the deterministic Ethernet networks[4].

II. Background

A. Software-defined Networking and OpenFlow

The data packet forwarding functionality of network devices is split into functional planes termed as: the control plane and data plane. The control plane determines the forwarding behavior of data packets traversing through the network device. The data plane performs simple data packet forwarding tasks based on the instructions provided by the control plane. In non-SDN networking, each device in the network owns a dedicated control plane and data plane. As a result, the forwarding and traffic shaping behavior is determined by the device control plane using complex distributed routing protocols. As a consequence of the distributed control plane, the global

B. Madiwalar is with Technische Universität München, Munich, Germany
B. Schneider, S. Profanter are with fortiss, An-Institut Technische Universität München, Munich, Germany. Correspondance should be directed to basavaraj.madiwalar@tum.de

configuration and management of the network device is more challenging.

SDN separates the device control and data plane by migrating the control plane to a centralized network control device: an SDN controller. Additionally, it uses open-standard interfaces for communication between the centralized control plane and data plane. The OpenFlow protocol is one such open-standard interface, which enables programming of network device forwarding information [5]. Thereby SDN and OpenFlow facilitate the programmability of the device forwarding behavior and enable a higher flexibility in network configuration in contrast to non-SDN networks.

OpenFlow implements the packet processing pipeline in network devices using a series of flow tables and group tables. The flow table holds the flow rules to represent packet header matching constraints. The group table can be used to implement complex forwarding behavior such as multicast forwarding. Additionally, it defines messages which the SDN controller can use to add, delete, or modify flow rules and group constructs.

B. OPC UA - Middleware for Industrial IoT

OPC UA is a standardized communication middleware providing semantic modeling features for its information model. Since the wire protocol of OPC UA is standardized, our proposed combination with SDN can be used independent of the used OPC UA stack. One strength of OPC UA is its semantically described information model which can be used to provide additional information to connected clients. Besides of the protocol and information model, OPC UA also defines two discovery services to discover other OPC UA instances in the network: Local Discovery Services (LDS) are used to find instances in the same subnet, while Global Discovery Services (GDS) can be used across different subnets.

In this paper we focus on LDS and combine it with SDN features to enable Plug and Produce systems. An extension of LDS is LDS with Multicast Extension (LDS-ME): it uses mDNS broadcast messages carrying DNS-SD information to identify OPC UA instances. These broadcast messages are sent through the whole network, such that every OPC UA LDS-ME will receive this information. This allows adding new OPC UA devices to the network without pre-configuring the OPC UA counterparts. A more detailed explanation of LDS-ME can be found in [3].

The mDNS message broadcast leads to some issues: first of all, the network is flooded by broadcast messages for any new OPC UA device which is added to the system. Secondly, if there are multiple OPC UA devices in the network, the newly plugged device needs to have some built-in logic to decide to which control device it needs to register. This control device may vary, depending on the factory setup.

Therefore an intelligent network is required which automatically routes the mDNS packets to the correct control devices, and therefore shifts the logic from the device level to a more centralized component. In our proposed system this centralized component is the SDN controller which is connected to the manufacturing service bus or other higher-level components.

III. Related Work

The need to transform flexible manufacturing facilities in order to cope with changing customer demands was recommended around 20 years back. There are several attempts being made to improve the flexibility of the production facility utilizing auto-configuration methods.

Durkop et al. [6] proposes an auto-configuration process for Real-Time Ethernet(RTE) system using OPC UA. The auto-configuration process is split into discovery and configuration. When a device is plugged in, the OPC UA server on the device registers with an OPC UA discovery server on the IO controller device. The IP address of the discovery server is distributed using the options field in Dynamic Host Control Protocol (DHCP) messages. However, the work doesn't make use of OPC UA discovery process as the OPC UA discovery services were published later in the year 2015. Moreover, it is not evident how the DHCP server is able to map the new devices if there are multiple IO controllers.

The Device Profile for Web Services (DPWS) another middle-ware technology discussed alongside OPC UA. Authors in [2] uses DPWS to automate the integration of field devices. Similar to OPC UA, it supports the client-server architecture and web-based discovery services. The client can make use of Web Service technologies to execute services on the servers. DPWS uses service description file to describe the service information of a server application. The major limitation of using DPWS is that a client application needs to posses the server service description file.

Henneke et al. [8] extend the OPC UA device discovery process with SDN and Network Function Virtualization. The authors focus on limitations of OPC UA local discovery services in networks with multiple subnets (i.e., OPC UA GDS). The solution offers network-wide discovery services by utilizing an SDN controller and OPC UA GDS as a Virtual Network Function (VNF). The solution reduces multicast traffic and enables the discovery of OPC UA server instances across the network.

IV. Functional Requirements of Plug and Produce Systems

A flexible manufacturing facility should support the following functional requirements for the realization of a Plug and Produce system:

- **Skill as a Service:** A device in a flexible manufacturing facility should offer its skill as a service for other devices. Besides, devices should provide self-description of their capabilities [9].
- **Automatic Device Discovery:** A device should be able to discover other devices in the system and their services without any network specific pre-configuration.
- **Standard Communication Protocol:** The devices should communicate using standardized communication proto-

cols for interoperability with devices from different vendors [10].

- **Flexible Network Infrastructure:** The network infrastructure should be flexible and dynamically configurable. It should accommodate new devices or topology changes without the need to configure network devices manually.

OPC UA satisfies the first three requirements. Using an OPC UA server application and its address space one can model the functionality and data associated with different components in an industrial automation system. Additionally, OPC UA supports the standard Ethernet Protocol as recommended by the Reference Architecture Model for Industry 4.0 [10]. The use of standard Ethernet across the layers of manufacturing facility eliminates the need to deploy and configure multiple communication protocols.

SDN promotes programmable networks and hence caters for the requirement of flexible and dynamically configurable networks. The SDN controller with a consolidated view of the network topology enables the identification of the automation device network location. Therefore, we combine OPC UA and SDN in our proposed solution, in order to fulfil all the requirements mentioned above. Here the role of the SDN controller is not limited to configuring flow rules. It is used as the intelligent network control device to supervise the automated device discovery process. The subsection V-B provides more specific details about the role of the SDN controller in our setup.

The subsequent sections describe the components of the proposed Plug and Produce system consisting of OPC UA client-server applications and the SDN network infrastructure. Additionally, the implementation details of the device discovery and configuration process are discussed.

V. Design of a Plug and Produce System

This section is split into two parts covering the details of components providing the production functionality and the SDN infrastructure interconnecting components. Subsections further describe the role of each one of these components in the device discovery and configuration process.

A. Components of a Plug and Produce System

In this work, we model the production process by utilizing the concept of hierarchical OPC UA servers described in our previous work [3]. The hierarchy of OPC UA servers helps to divide the complex production process into sub-tasks. Fig. 1 shows a model of the proposed Plug and Produce system with a hierarchy OPC UA servers. The hierarchy of OPC UA servers helps to abstract the capabilities of devices to higher level control components.

The workstation in Fig. 1 implements a subtask utilizing a set of devices offering specific functionality. The coordinator on the workstation controls the devices on the workstation by implementing the execution logic of the subtask. To detect devices on the workstation, the coordinator uses an instance of an OPC UA LDS-ME server. A device on the workstation models the functionality of field devices such as a sensor or a conveyor. The device uses an OPC UA server to implement the functionality of a field device. When a device is plugged in, it discovers and registers with the coordinator on the workstation by implementing a subset of mDNS responder services.

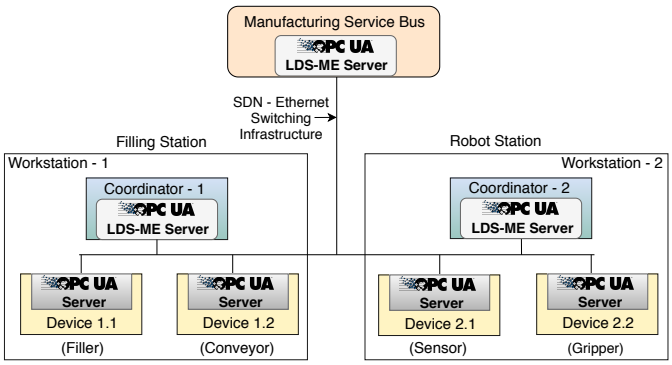

Fig. 1. Components of the proposed Plug and Produce system. The Manufacturing Services Bus configures the coordinator components on workstation. The coordinator component orchestrates devices on the workstation.

The Manufacturing Service Bus (MSB) is responsible for detecting and configuring coordinators on workstations. It acts as a centralized communication component and provides the requesting coordinator with the information about other available coordinators. The MSB uses an instance of an LDS-ME server and listens for mDNS announcements from LDS-ME servers on coordinators.

B. Software-defined Networking Infrastructure

Fig. 2 depicts the proposed network infrastructure for the Plug and Produce system consisting of the SDN controller and OpenFlow switches. The SDN controller with the help of the network controller services configures the OpenFlow switches to handle data-traffic between devices, coordinators, and the MSB.

Additional to the general flow rule configuration, the SDN controller supervises the device discovery process. In the OPC UA multicast based discovery process, devices with an OPC UA server announce their services using mDNS multicast announcement messages. A device receiving an announcement message, responds with its implemented services using an mDNS multicast response message. Lets assume, that we use non-SDN Ethernet switches instead of OpenFlow switches. The non-SDN switches flood mDNS multicast packets to all the ports except the ingress port. As a result, all devices and coordinators in the subnet receive the mDNS announcement and respond to it. Therefore, the state of the art approach proposes to pre-configure new devices with the designated coordinator information, in order to filter the received mDNS responses [3]. However, in this work, we eliminate the need for such pre-configuration by using an SDN controller and programmable OpenFlow switches. The SDN controller stores and forwards mDNS responses only from correct coordinator to newly plugged-in devices.

The SDN controller uses the skill information of a plugged-in device, the network topology information, and the automa-

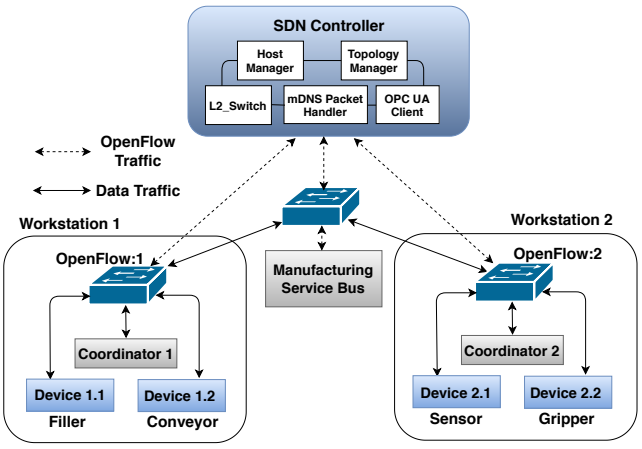

Fig. 2. SDN Infrastructure for Plug and Produce System.

tion process plan to identify the correct coordinator response messages. It uses an OPC UA client instance to obtain the new device skill information. We propose to use a dedicated OpenFlow switch per workstation. Using a dedicated switch, helps the SDN controller to identify the physical location of devices and to distinguish devices with similar skills on different workstations. Additionally, the SDN controller uses of the topology manager service to gather the switch ID for the devices. It then compares the device switch ID with the switch ID and workstation map to determine the physical location of the newly plugged device. An example of a switch ID and workstation map is shown in Table I.

An administrator is responsible to provide the SDN controller with an automation plan. The automation plan is a simple map of the workstation and skills provided by devices on the same workstation. An example of a workstation and skill map is shown in Table II. Using the combination of skill information, topology information and automation plan, the SDN controller identifies the coordinator for the newly plugged-in devices. Thereby, the SDN controller eliminates the need to pre-configure devices with their control entity information. The combination of OPC UA discovery services and the centralized SDN controller's ability to direct the mDNS multicast traffic helps new devices to discover and register with their control entities.

We make use of the OpenDaylight SDN controller framework to implement the network control services. These network control services provide mechanisms to configure OpenFlow flow-rules and process mDNS multicast announcements. The next section provides the step-by-step details of the device discovery and configuration process using the SDN controller and OPC UA LDS-ME.

VI. Device Discovery and Configuration

In our proposed Plug and Produce system, the process of integrating new devices is split into device discovery and configuration stages. In the discovery phase, the new device use the mDNS responder services to announce its addition to the network. The SDN controller uses the announcement messages

TABLE I
Map of switch ID and Workstation.

Switch-ID	Workstation-ID
OpenFlow-1	Workstation-1
OpenFlow-2	Workstation-2

TABLE II
Map of Workstation and Skill set.

Workstation-ID	Skill Set
Workstation-1	Conveyor, Filler
Workstation-2	Sensor, Gripper

to detect new devices and helps them to find and register with their desired control component. In the configuration stage, the control component configures the registered device with its control parameters. In our proposed approach, the discovery phase is nontrivial as compared to configuration stage. The combination of the SDN controller and OPC UA discovery servers facilitate the discovery phase. The next subsections describe the details of the steps involved in the discovery and configuration of new devices.

A. Device Discovery

The subsection provides details about the messages exchanged between the devices with OPC UA servers and the SDN controller during the discovery phase. Figure 3 depicts the order of messages exchanged between the SDN controller and devices implementing specific skills. As a first step, the SDN controller and OpenFlow switches are powered on. In addition, the OpenFlow switch on all the workstations are configured to forward the unmatched mDNS packets to the SDN controller. The details of messages exchanged between the coordinator, devices, and the SDN controller are as follows:

1) The Coordinator device is powered-on before powering any other devices on the workstation. The LDS-ME server on the coordinator sends mDNS announcement messages with Resource Records (RRs) of the OPC UA server. The OpenFlow switch forwards the announcement messages to the SDN controller for further processing. The mDNS packet handler module in the SDN controller collects the discoveryURL information from the RRs and passes it to the OPC UA client module.
2) The OPC UA client performs an OPC UA read request for the skill information using the provided discoveryURL. The SDN controller uses OpenFlow PacketOut messages to instruct the OpenFlow switch to forward packets via the port connecting the newly plugged Coordinator device.
3) The OPC UA server on the coordinator returns responds with the skill information. The SDN controller decides the next processing steps based on the collected skill

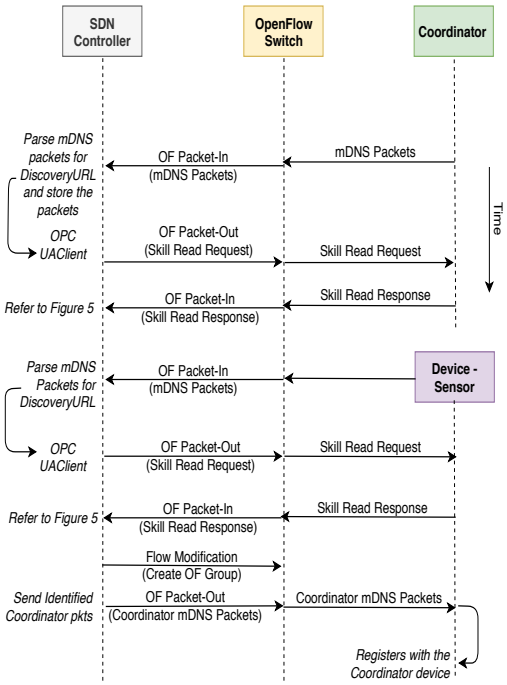

Fig. 3. Sequence of messages exchanged with the SDN controller when a device joins the network.

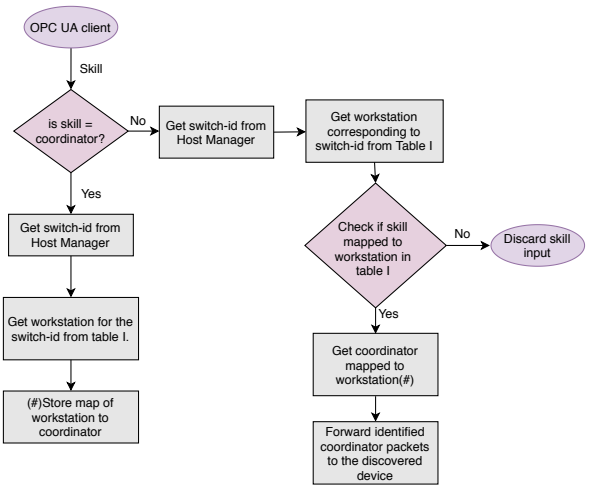

Fig. 4. The flow diagram of an SDN controller's device skill processing method.

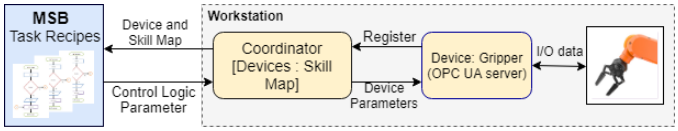

Fig. 5. An example device configuration process for a plugged-in gripper.

information. Figure 4 shows the flow-chart of the SDN controller's skill processing method.

4) In the next step, the device offering specific skill (or functionality) is powered-on. The OPC UA server on these devices sends mDNS packets with the RRs of the OPC UA server implementing the skill. The OpenFlow switch forwards these mDNS packets to the SDN controller for further processing.
5) Similar to point 2, the SDN controller parses the mDNS packets for the discoveryURL of the OPC UA server and uses the OPC UA client module to perform a read request for the skill information.
6) the OPC UA server on the device responds with the skill information. Figure 4 shows how the SDN controller processes the obtained skill information.
7) Upon identification of the correct coordinator device, the SDN controller forwards the buffered coordinator mDNS packets (point 2) to the newly plugged-in device. Additionally, the SDN controller configures the OpenFlow group and OpenFlow flow-rule to implement the multicast group consisting of coordinator and devices assigned to the mapped coordinator.
8) The OPC UA server uses the information from the received mDNS packets and registers with the coordinator.

B. Device Configuration

We use the device configuration process defined in [1]. Figure 5 shows an example of the new device configuration process after the successful registration with a coordinator. The gripper object is discovered and registers with its designated coordinator device as described in the previous subsection. The OPC UA server on the coordinator spawns an OPC UA client instance to gather the registered device skill information. Further, the coordinator device informs the MSB with a list of registered devices along with their skill information. The MSB contains the coordinator device control logic parameters and execution order in the form of task recipes. The task recipes represent the machine configuration derived from the product specification. The OPC UA server on the coordinator reuses the already created client instance to configure control parameters of the registered devices.

VII. Evaluation

The previous sections present the Plug and Produce system details and a systematic approach to discover and configure new devices using SDN and OPC UA. The reduced configuration effort improves the flexibility of manufacturing facilities. The flexibility of a system is a qualitative parameter and its interpretation depends on the system under question. Therefore in this section, we first define the flexibility aspects of the proposed Plug and Produce system and compare the proposed SDN based Plug and Produce system against the non-SDN based approach specified in [3].

A. Flexibility Aspect of a Plug and Produce System

The integration of SDN and OPC UA is inevitable as OPC UA alone is not sufficient to achieve Plug and Produce [3]. Our previous work with OPC UA and non-SDN network infrastructure showed that devices require pre-configured information to select the correct control entity [3]. The ability of the

SDN controller to store and forward mDNS multicast packets eliminates the need for this kind of pre-configuration. Hence, the SDN infrastructure plays a greater role in comparison to OPC UA to improve the system flexibility. Therefore, we use [11] as a reference to draw the definition of flexibility for our proposed Plug and Produce system. The definition of flexibility varies depending on which component of the network as a system improves system's flexibility. For example, flexibility can be represented by the scalability of the network, the ability to react to topology changes and the ease of network configuration. In our proposed Plug and Produce system the term flexibility refers to the ease of adding new devices to the system with minimum or no configuration effort and without affecting the process under control. It is hard to quantitatively compare two systems using flexibility as a parameter. Therefore, we have considered alternative parameters for a comparison and discuss them in the next subsection.

B. Comparison of Parameters

The literature survey suggests that the cost incurred to improve the flexibility of a system can be a parameter for comparing the network design choices [11]. Therefore, we are considering a system cost parameter, that is common for both the SDN and non-SDN based Plug and Produce system.

The mDNS protocol as a distributed name resolution protocol uses cache maintenance strategy. As a result, the mDNS responder service should flush discovered services whenever the state of the system or network interface changes. Therefore, devices using OPC UA discovery services have to rediscover and register with its control entity. In production systems with distributed control systems the unavailability of a part of the system could hamper the production quality. Moreover, a flexible system should handle changes to the system state in a timely manner. Therefore, we choose the time to register as the cost parameter for the comparison and, its definition is as below.

Time to Register: The time required for a device to rediscover and register with its control entity.

The use of an SDN controller to facilitate the device discovery by controlling the flow of mDNS traffic creates several additional benefits. The non-SDN approach forces devices to process a large number of mDNS multicast packets (i.e., through the flooding of multicast packets). Moreover, the field devices are embedded devices with limited resource capacity and network interfaces with limited buffer size. The large multicast traffic can affect the timely processing of time critical control packets. However, due to unavailability of real embedded devices and limitations of virtual network emulator, we could not evaluate the effect of mDNS traffic on limited buffer-size and packet processing delay on real hardware. As an alternative, we are comparing the number of multicast packets received by devices in our proposed SDN based Plug and Produce system against the non-SDN approach.

mDNS multicast packet count: The number of mDNS multicast packets received by devices with OPC UA servers in the SDN and non-SDN based Plug and Produce system.

The testbed setup and measurement process for both the Time to Register and mDNS multicast packet count experiments are largely similar. Therefore, the we provide a common description for the testbed and measurement process.

C. Testbed Setup and Measurement Process

The subsection provides details about the testbed used for the measurement of time to register and mDNS multicast packet count parameters. We use Containernet as the network emulator to create a virtual network infrastructure with OpenVswitches [12]. We use Containernet's Python APIs to create the virtual network topology and run OPC UA client-server applications. Additionally, we use Pyshark2 Python libraries to count mDNS multicast packets on each host interface.

Figure. 6 shows the sample virtual network topology used for the measurement of the time to register and mDNS multicast packet count experiments. The network topology consists of OpenVswitches connected in a linear fashion to verify the SDN controller's ability to limit the multicast traffic to devices on the same workstation. The same network topology with OpenVswitches configured to operate in normal-mode for the measurement of non-SDN based approach. Each workstation contains three devices with an OPC UA server modeling the skill of real-world objects and a coordinator with an LDS-ME server. All workstations contain devices with the same set of skills to demonstrate the SDN controller's ability to distinguish devices with same skills on the basis of the topology information. The SDN controller and virtual network topology run on two different test PCs. Both test PCs equipped with Intel i7 processors@3.70GHz, 8GB of RAM running Ubuntu 16.08. The test PCs are connected via a 100Mbps Ethernet network interfaces.

The measurement process is automated by a Python script and begins by starting the SDN controller and configuring the SDN controller with the automation plan as shown in Table I and II. Next, we create a virtual network for the given number of devices. The OPC UA server and LDS-ME server applications are mapped to the Docker Containers acting as devices and coordinators. Using the Python APIs, the OPC UA server applications on coordinator and devices are started. The OPC UA server applications are designed to log the time after a successful registration with the respective coordinator devices. The registration process is repeated for 100 iterations to ensure that statistics of the sample can reliably represent the population parameters such as mean and standard deviation. The measurement process for the non-SDN based approach is largely similar to SDN based approach. The OpenVswitch are configured to operate in learning switch mode and the OPC UA server applications is started by providing the hostname of the coordinator device as the argument.

D. Discussion of Measurement Results

This subsection provides the quantitative comparison of the Time to Register and mDNS multicast packet count values for the SDN based and non-SDN based experiments.

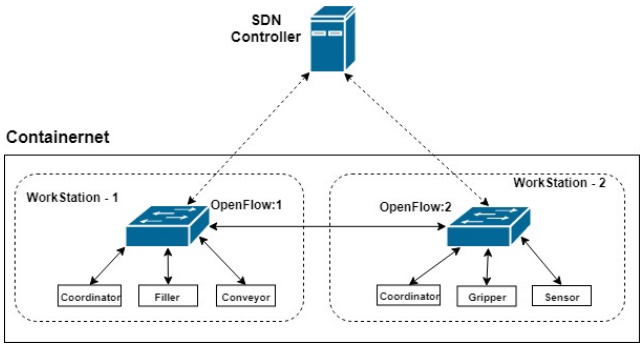

Fig. 6. Example network toplogy for the measurement of Time to Register and mDNS multicast packet count. The network topology shown here contains only 8 devices. However, for the measurement we extend the network topology with up-to 48 OPC UA servers by adding more workstation units.

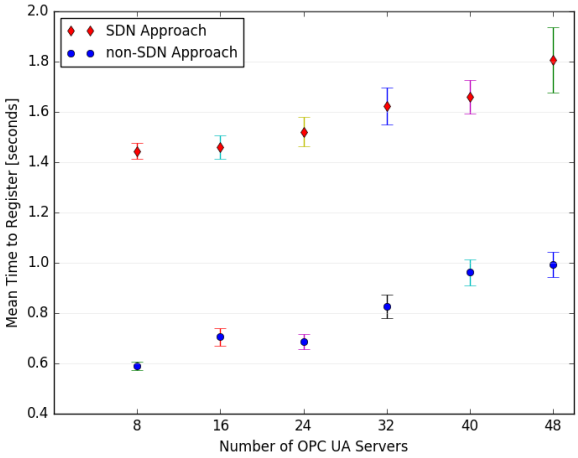

Fig. 7. The plot compares the time to register values for devices in the SDN based Plug and Produce system against non-SDN based approach The measurement performed by adding different of OPC UA servers to the network at the same time.

1) Time to Register: Fig. 7 compares the mean time to register value for a device in the SDN based approach against the non-SDN based approach. The X axis shows the number of OPC UA servers joining the network at the same time and Y axis depicts the time to register values in seconds. Fig. 7 shows that the average time to register values for both the approaches increases when more OPC UA servers joins the network at the same time. The time to register for SDN based approach increases approximately by 76ms for an additional eight OPC UA servers joining the network. Similarly, the time to register values for non-SDN based approach increases by 80ms for an additional eight OPC UA servers joining the network at the same time. A device in the SDN based approach requires approximately one second more than a device in the non-SDN based approach to discover and register with the control entity.

Different components in both the approaches contribute to the increasing time to register. A device with an OPC UA server sends out nine mDNS multicast packets when it joins the network. As a result, the SDN controller has to process twice the number of mDNS packets for every additional OPC UA server in the network. In the case of non-SDN approach, every device in the network needs to process double the amount of mDNS multicast packets when an additional OPC UA server joins the network. Furthermore, the OPC UA server applications are single threaded applications. As a result, the time to register values increases for both approaches. Moreover, the SDN controller needs to query devices for their skill information. All these factors contribute to the higher time to register values for the SDN case. We conclude that it is necessary to store the discovered service information for use across reboots in order to reduce the time to register for the SDN approach and that the network should offer a high degree of availability.

2) mDNS Multicast Packet Count: The subsection compares the number of mDNS multicast packets received by devices in the SDN and non-SDN approach. Figure 8 compares the average number of mDNS multicast packets received by devices in both approaches. The Y-axis shows the mDNS multicast packet count and the X-axis shows the number of OPC UA servers in the network. A device in the non-SDN approach needs to process approximately 140 more mDNS multicast packets for every additional 8 OPC UA servers in the network. Whereas, a device in the SDN based approach processes an average of 48 mDNS multicast packets in networks with a different number of OPC UA servers. Moreover in the SDN case, the number of mDNS multicast packets received, depends only on the number of OPC UA servers on the same workstation. Therefore, the mDNS multicast packet count for the SDN based approach remains constant as we add new workstations to increase the number of OPC UA servers.

From the above discussion, it is evident that when new workstations are added to the network, the devices in the non-SDN based approach need to process a constantly rising number of mDNS multicast packets. This can heavily affect the industrial communication system in production, as such systems often require deterministic End-2-End (E2E) delay bounds which are influenced by the mDNS configuration traffic. It is important to note that OPC UA server applications are single threaded applications and processes call from network stack in sequential order. Therefore, the large mDNS multicast traffic increases the load on the OPC UA server. As a consequence, it can lead to non-deterministic E2E delays. Therefore, we conclude that the SDN based approach handles mDNS multicast traffic at a constant rate even with an increasing amount of joining OPC UA servers, whereas the mDNS packet count constantly increases for a rising number of joining OPC UA servers for the non-SDN approach.

VIII. Conclusion and Future Work

Our work demonstrates that it is possible to create a Plug and Produce manufacturing system by leveraging the capabilities of SDN and OPC UA. The SDN controller requires device skill information and an automation-plan to determine the control entity for new devices. It doesn't need any device

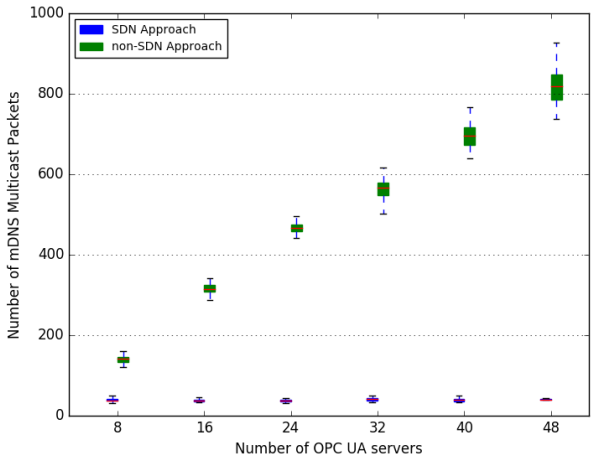

Fig. 8. The plot compares the distribution of number of mDNS packets received by a device in SDN based Plug and Produce system against the non-SDN based approach.

specific data like MAC addresses or hostnames. As a result, even if we replace a defective device with a new device, the SDN controller can still help new device to discover its control entity. Our evaluation showed that the proposed SDN-based approach is scalable in terms of transmitted multicast packets on the network when more OPC UA servers are added to the system. Additionally, the SDN controller's ability to communicate with OPC UA servers in the network, can create more interesting use-cases in machine to machine communication. The use of platform-independent OPC UA as a middle-ware, enables communication across layers of the manufacturing facility and among devices from multiple vendors. The support of standard Ethernet by SDN and OPC UA helps to replace vendor-specific field-bus protocols. Thereby, the combination of OPC UA and SDN serves to improve the interoperability and reduces the overall system cost.

The proposed SDN controller logic to determine control devices requires further work to cater for more complex automation scenarios. The MSB application can act as the SDN controller northbound application and make use of semantic reasoning to solve more complex use cases. We further plan to investigate how SDN and OPC UA functions in heterogeneous communication networks.

Acknowledgment

We like to thank Amaury Van Bemten for providing suggestions for the integration of the OpenDaylight SDN controller and OPC UA client application.

References

[1] K. Dorofeev, C.-H. Cheng, M. Guedes, P. Ferreira, S. Profanter, and A. Zoitl, "Device adapter concept towards enabling plug&produce production environments," in *Proceedings of the IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2017.

[2] S. Hodek and J. Schlick, "Ad hoc field device integration using device profiles, concepts for automated configuration and web service technologies: Plug&play field device integration concepts for industrial production processes," in *International Multi-Conference on Systems, Signals & Devices*, 2012.

[3] S. Profanter, K. Dorofeev, A. Zoitl, and A. Knoll, "OPC UA for plug & produce: Automatic device discovery using LDS-ME," in *Proceedings of the IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2017.

[4] S. B. H. Said, Q. H. Truong, and M. Boc, "Sdn-based configuration solution for ieee 802.1 time sensitive networking (tsn)," *ACM SIGBED Review*, vol. 16, no. 1, pp. 27–32, 2019.

[5] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[6] L. Dürkop, J. Imtiaz, H. Trsek, L. Wisniewski, and J. Jasperneite, "Using opc-ua for the autoconfiguration of real-time ethernet systems," in *2013 11th IEEE International Conference on Industrial Informatics (INDIN)*. IEEE, 2013, pp. 248–253.

[7] L. Dürkop, J. Imtiaz, H. Trsek, and J. Jasperneite, "Service-oriented architecture for the autoconfiguration of real-time ethernet systems," in *3rd Annual Colloquium Communication in Automation (KommA)*, 2012.

[8] D. Henneke, A. Brozmann, L. Wisniewski, and J. Jasperneite, "Leveraging opc-ua discovery by software-defined networking and network function virtualization," in *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*. IEEE, 2018, pp. 1–4.

[9] D. C. M. Hartmut Rauen, "Industrie 4.0 communication guideline based on opc ua," VDMA Industrie 4.0 Forum, Fraunhofer Application Center Industrial Automation (IOSB-INA), Tech. Rep., 2017.

[10] M. Hankel and B. Rexroth, "The reference architectural model industrie 4.0 (RAMI 4.0)," *ZVEI, April*, 2015.

[11] W. Kellerer, A. Basta, P. Babarczi, A. Blenk, M. He, M. Klugel, and A. M. Alba, "How to measure network flexibility? a proposal for evaluating softwarized networks," *IEEE Communications Magazine*, no. 99, pp. 2–8, 2018.

[12] M. Peuster, H. Karl, and S. van Rossem, "Medicine: Rapid prototyping of production-ready network services in multi-pop environments," in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov 2016, pp. 148–153.