Determining Edge Node Real-Time Capabilities

Varun Gowtham, Oliver Keil, Aniket Yeole, Florian Schreiner
Fraunhofer FOKUS
Software-based Networks
Berlin, Germany
{first.last}@fokus.fraunhofer.de

Simon Tschöke
German Edge Cloud
Research Department
Eschborn, Germany
simon.tschoeke@gec.io

Alexander Willner
Fraunhofer FOKUS / TU Berlin
Software-based Networks
Berlin, Germany
https://orcid.org/0000-0002-8615-4902

***Abstract*—The distributed Cloud Computing paradigm is continuously being adopted within the domain of industrial automation. The most distinguishing feature of these Edge Clouds relates to their ability to provide low-latency and even hard real-time services. As infrastructure deployments can be rather heterogeneous in their nature, service providers require precise means for estimating end-to-end application latency behavior, in order to know the performance boundaries that can be met for defining certain Service Level Agreements (SLAs). Although network performance tools exist for many years, mechanisms for assessing hard real-time performance of applications in distributed Edge Cloud environments have not been considered extensively yet. Therefore, we use a built-in feature of the Linux Kernel, the extended Berkeley Packet Filter (eBPF), to measure delays between targeted endpoints in the kernel stack, that enable the user to gain deeper and more accurate measurements of events as compared to generalized approaches (as accurate as eBPF / the time-stamping facility from the kernel). As a result, the real-time behavior of particular Edge Cloud deployments, including its hosted applications, can be profiled in detail by end-users as well as service-providers. Within our evaluation we have monitored a cyclic transmission of packets with a scheduled delay of under 190 µs and measured a round trip time under 2 ms. Future work include profiling the real-time behavior of potentially hosted time-critical applications, such as virtual Programmable Logic Controllers (vPLCs), over real-time networks, such as Time Sensitive Networking (TSN); the extension towards dynamically configured real-time networks; and finally its application to future organic, self-optimizing Ultra-Reliable Low-Latency Communication 6G core networks.**

***Index Terms*—eBPF, Edge Computing, IoT, IIoT, Industry 4.0, NFV, PLC, TSN, vPLC**

I. Introduction

The distributed Cloud Computing paradigm is being adopted in the domain of Operational Technologys (OTs) in the form of Edge computing infrastructures, especially where low latency and real-time capabilities are required. Industrial applications are often time-critical in nature. I.e., an end user relies on the assurance received from the application providers that a designated operation can be completed in a specified time frame. Under this premise, a sub-system constituting other time-critical applications can work in parallel to achieve the overall goal of the industrial process. Time critical applications are addressed and studied in the field of real-time systems, where systems should produce logically correct output under stipulated time constraints. Such time constraints are dictated by the natural evolution of the system states in the environment where a given system is operating [5, 36].

When focusing on Edge-based real-time applications in industrial automation domain, Software-based Programmable Logic Controllers (PLCs) pose considerable challenges in terms of determinism and reliability on a given Edge computing infrastructure. In this context, a virtual Programmable Logic Controller (vPLC) encapsulates functional aspects of PLC as a software package, such that it can run on commodity hardware. This transformation enables the opportunity to bring DevOps practices closer to OT environments. Although PLCs follow standards such as IEC 61131-3 and IEC 61499, implementations are often proprietary technologies, which have existed since decades, delivering reliable and deterministic control characteristics, crucial to industrial processes. Since vPLCs can be hosted on commodity Edge Computing hardware, which may or may not be optimized for industrial applications, there is an inherent need to assess the capability of the execution platform towards hosting vPLCs including underlying real-time networks.

This document draws the requirements for a measurement framework keeping in view the use case provided by end user. More specifically, this includes that (1) the measured process is an industrial application, (2) the industrial application comprises of a controller configured and exercised on private cloud facility provided by an Edge Cloud provider, (3) the controller is able to further control an Input/Output (I/O) through the means of a real-time capable networking technology for example industrial field bus or Time Sensitive Networking (TSN), and (4) the Edge Cloud infrastructure and application providers, as well as final end-users, intend to measure capabilities of an Edge Computing infrastructure, towards hosting time-critical real-time control logic applications. Thus, the requirement specification captures the preferable features of a measurement framework in the direction of extraction of relevant Key Performance Indicators (KPIs) required to assess feasibility of hosting time critical applications, especially for vPLCs.

We provide an overview of the state of the art and related work. The related work is organized using a typical industrial control use case of a computing node that directs a remote

978-1-6654-3326-6/21/$31.00 ©2021 IEEE

I/O controller via a network path. We further divide the use case into 3 specific topics of computing node, network and remote I/O, and present the related work for each identified topic. Although we found monitoring architectures targeting specific domains, there was a lack of a monitoring system that enabled user to select and monitor a specific event and at the same time being user friendly. Our approach, which is referred as MessTool in the remainder of the paper, is designed to not only be able to assess the above specified requirements, but also to make it easier for the user to apply complex profiling approaches for software systems flexibly and reliably. We focus incorporating available open source solutions, so that it is possible for the user to apply the tools and to extend them. As one main contribution of this paper, we provide details about the overall architecture and the implementation of MessTool framework that can profile end-to-end time behavior of distributed edge applications, considering the outlined requirements.

We have evaluated the implementation against currently used server systems, with mock applications that simulate the behavior of a vPLC; an application that periodically transmits a network packet with a maximum delay of 190 µs in the software stack and the network round trip time measured at a preferred point in the operating system with under 2 ms. The framework is used to measure the execution characteristics of the application on the computing system. Furthermore, the framework itself is evaluated to determine the precision and accuracy of the measurements obtained on monitoring real-world use cases.

It is in our view that the designed framework will help Edge Cloud infrastructure and application providers to profile their infrastructure for specific application requirements. Further work may include the measurement and dynamic re-configuration of real-time TSN networks based on user specified intents / requested Service Level Agreements (SLAs) [38]; as well as the adoption of lessons learned for High-Precision Networking (HPN) in the context of future 6G networks. Finally, the time behavior of an infrastructure could be an integral part of its Digital Twin (DT) along with its topology and therefore provide valuable information for use-cases such as capacity planning.

The remainder of the paper is structured as follows. In section II, we give a brief overview of related work in the context of available approaches and tools related to the topics of measurement and monitoring. section II provides an overview of the state of the art and related work. In the subsequent Section III the architecture and implementation of the system is presented. The evaluation results obtained from analyzing the framework on a realistic server infrastructure and mock applications is discussed in Section IV. Finally, we close by giving some conclusions and considerations and describe future work in Section V.

II. Related Work

A. General Approaches

In order to collate the related work, a hypothetical use case that embodies a real-time behavior is presented. Figure 1 shows

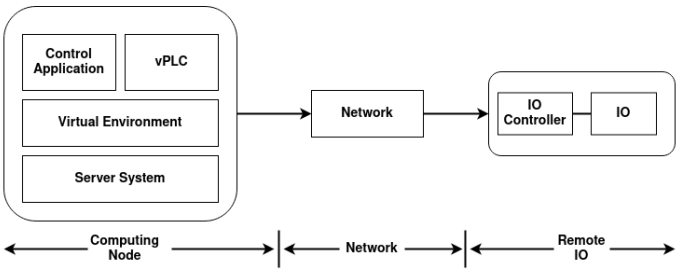

Fig. 1: Industrial Control Use Case.

the industrial control process, where a control application or a software-based PLC executing in a virtual environment on a server system, controls an I/O device through a dedicated communication network. For a given action performed by the I/O, it can be observed that the control application is responsible for providing timely control input to the I/O. Furthermore, the underlying communication network should also be able to satisfy the timing requirements stipulated by the control application; e.g., TSN compliant network.

In order to measure a system, we have to first identify the characteristics of the system, and then administer right strategy to measure a given characteristic. In run-time reflection frameworks, the monitoring system is a collection of logging, monitoring, diagnosis and migration layers [28]. An abstract notion of specifying the system behavior in the form of events, traces and their associated specifications was presented [2].

Several architectures have been proposed for monitoring distributed real time systems. A practical discussion on the importance of run-time monitoring of flight safety critical systems and an architecture is available in [15]. The integration of contrasting approaches of rigorous time-triggered and flexible event-triggered traffic based control, emphasizing the role of deterministic networking for hard-real system was proposed in [12]. *Breadcrumbs* was proposed for monitoring timing constraints of event flows follows a decentralized approach [25]. A non-invasive monitoring system for automotive vehicles was discussed [23]. Real-time systems are required to exhibit both logical and temporal correctness of the produced result. Therefore, the challenge of monitoring distributed real-time systems relies on differentiating between causality of an error due to logical or a temporal incorrectness[39]. A run time environment for monitoring real-time systems by inserting traceable events in the execution program and monitoring the events during run-time was proposed in [7]. The *Brace* architecture was proposed for distributed monitoring of real-time systems in [41]. Furthermore, the knowledge of architecture and functioning of real-time systems has impacts in proposing balanced strategies for monitoring system as discussed in [24]. There is research conducted in the direction of run-time verification of timing constraints, such as [21].

The rest of the section provides related work in targeted areas, ordered as remote I/O controllers, computing node and network as shown in Figure 1.

B. Remote I/O

The monitoring of I/O controllers has been particularly challenging, because of the unavailability of standardized tools to obtain event traces, since most of these devices are proprietary. The scientific literature shows the possibility of monitoring indirectly either by analyzing the control traffic or by actually seeking the digital I/O. A non-invasive monitoring of distributed real-time systems was proposed in [40]. Pellizzoni et al. [31] propose to monitor Commercially Off The Shelf (COTS) device peripherals against their assumed specifications. Hadžiosmanović et al. [19] provide an approach to monitor PLC variable state changes induced as a result of malicious control packets sent via the network to the PLC by an attacker. Kleines et al. [27] provide direct inputs to peripherals of PLC devices from Siemens and test for their performance, by method of analyzing the signals from output peripherals of the PLC. In another approach, Cruz et al. [10] use a remote device deployed along with a PLC to transparently intercept communications messages received by another PLC. Geier et al. [14] present a hybrid monitoring system that uses hardware capabilities such as an Field Programmable Gate Array (FPGA) to monitor systems. An analysis on using PLCs for measuring the size and complexity of PLC programs in different control logic methods has been conducted in [29]. Although this approach was suggested in the context of security, it could also be used for monitoring of remote I/O controllers, as noted in [16, 32].

C. Network

In the field of monitoring of networks, we point to scientific literature relevant to the scope of this manuscript for monitoring metrics such as end to end latency. Chao et al. [6] have tried to measure the precise latency of unidirectional data flows in a non-real time network. Decotignie [11] shows the adaptation of Ethernet to accommodate industrial use cases, providing insights into measuring industrial Ethernet. Prytz [33] has shown a comparison of performance between real-time Ethernet networks EtherCat and PROFINET IRT. Steiner et al. [37] have attempted to explain how the networking worlds of Information Technology (IT) and OT can be brought together with the help of TSN. TSN has been instrumental in bringing real-time networking capabilities nearer to consumers with COTS hardware, which was otherwise managed by vendors offering proprietary products. Furthermore, research conducted in the direction of assessing the capabilities of TSN for real-time networking is promising [8, 13, 17]. Apart from scientific literature available on measuring networking protocols and technologies, several network monitoring tools are available for Linux based machines.

In the direction of extracting timestamp measurements as intended by the proposed measurement tool, it is important to consider time synchronization. In particular, this is important for monitoring distributed systems. Research in the direction of TSN already covers topics of time synchronization, however, important papers related to time synchronization are available [9, 42].

D. Computing Node

In the field of monitoring computing nodes capable of real-time task execution, research has progressed in the direction of the feasibility of monitoring critical tasks. Jevtic et al. [22] propose a multi-level hybrid monitor for hard real-time systems. Bernat et al. [3] introduce the concept of weakly hard real-time systems as opposed to hard real time systems for the purpose of accommodating a flexible treatment of systems. Scholz et al. [35] analyzed performance implication of packet filtering using extended Berkeley Packet Filter (eBPF) [20]. The research has progressed in proposing frameworks and strategies for monitoring real-time tasks in [1, 4, 18, 26, 30, 34].

While it is a challenge to monitor I/O controllers or PLCs, the recent development efforts in the Linux kernel has made it possible to instrument and monitor system events. Many tools have existed and continue to exist to aide developers for the purpose of debugging. The Linux kernel has facilities which can be used to emit events when a particular function or a kernel symbol is executed. Six main data sources are Kernel Probes (kprobes and kretprobe), User Probes (uprobe and uretprobe), Kernel Tracepoints and User Statically-Defined Tracking (USDT). The data source being part of the Kernel infrastructure, enables the user to instrument and retrieve data values from the software stack, which are known to be production safe, because they execute directly in the context of program execution. The Linux Kernel also provides the observer infrastructure that can be used to retrieve data from sources. Tools in that domain are ftrace, eBPF, perf, SystemTap, and LTTng.

Existing tools and frameworks are either non-mutable and restrictive, or are highly specialized for use cases. In the former category, users can obtain measurements readily and there is less room for modification. However, in the latter category, users are open to a plethora of complex tools which need skilled administration to obtain the right measurements.

This paper focuses on measuring the real-time capabilities of an edge server that runs on Linux. Furthermore, the provided framework in the paper enables the user to administer highly specific filter to isolate specific event in the execution of software and, extract contextual and relevant details when the event occurs.

III. MessTool

In order to provide the reader with a supporting structure, this section is divided into three subsections: measurement design, measurement distribution architecture, and component and implementation details. While the first two subsections introduce the general measurement design and each component of the framework individually, the last subsection provides details of how the components work together. Additionally, the main important design constraints (for the components) are provided as well.

A. Measurement Design

Frameworks that target measurement of software systems are usually event based. The propagation of an event is from *Event Sources* to *Event Sinks*. When the event occurs at the source, a measurement framework samples a required metric and records it. This snapshot of the event can be captured at one or more locations in the path traversed by the event, from source to sink.

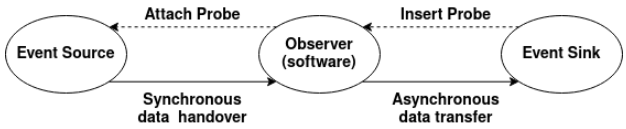

Fig. 2: General design of measurement framework.

Figure 2 consolidates the general design concept of systems that run on a Linux-based operating system. Here, the dashed line arrows depict the flow of configuration events, whenever new probes are brought into the system. Additionally, the solid line arrows depict the flow of probed event data, whenever the probe's attached events are triggered. For a software system, by default, the *Event Source* can be narrowed to the execution of a specific part of code (as instructions executed by the processor(s)). With built-in features of the Linux Kernel, a user is able to register a probe, to a particular location in the execution sequence of the software stack. When the processor executes the monitored code part, the Kernel triggers the registered probe, subsequently handing over the control to a dedicated *Observer*, along with contextual information from the execution.

The *Observer* comprises a set of actions or a function, that can record the contextual data and sample the required metrics. To this end, the *Event Sink* is then responsible to extract the required information and forward it for further processing. The execution window of the *Event Source* and the *Observer* are strictly monitored by the Kernel. The *Event Sink* is a user application, thus clearly defining the roles and boundaries of each component in the design.

B. Measurement Distribution Architecture

This subsection describes the overall architecture of the MessTool, designed as a platform for distributed measurements. MessTool follows a component-based design architecture, where each component has a dedicated role and can be viewed as a micro-service. Therefore, each component exposes its functionality through dedicated interfaces. All of the main components of the MessTool are designed to strictly run on the *Edge Server* site, whereas the main component for user interactions can be deployed more freely, regarding location. The *Collector* takes a mixed role and can be accessed by the user directly, for simplified scenarios.

User Services are provided mainly through:

- Monitoring Manager (MM): Provides the primary interface to the user. This allows a user to specify her intentions of what aspect, of which *Edge Server* is to be measured. Through this interface, the user also receives vital information about the status of her configurations.
- Collector: Provides a simplified interface through which a user can access measured data collected by MessTool.

Figure 3 depicts all main components of the MessTool's architecture and provides an architectural view on the two service categories.

The roles of *Edge Server Services* are detailed below:

- Monitoring Agent: Provides the primary interface to the Monitoring Manager and, with that, acts as its main mechanism to configure required probes, based on to be executed measurements.
- Probe Config Agent: Acts as an Event Sink and provides actual means to insert/remove probes. Since the actual probing mechanism can vary from system to system, this needs decoupling from the Monitoring Agent.
- Aggregator: Provides means to aggregate data from probe data to be able to combine/meld them into a single measurement. On the one hand this allows combining data from multiple probes, on the other hand this also allows to meld multiple data points from a single probe (computing time delays from probe events).
- Collector: collect measurements and provide those to a user or to an externally provided database.

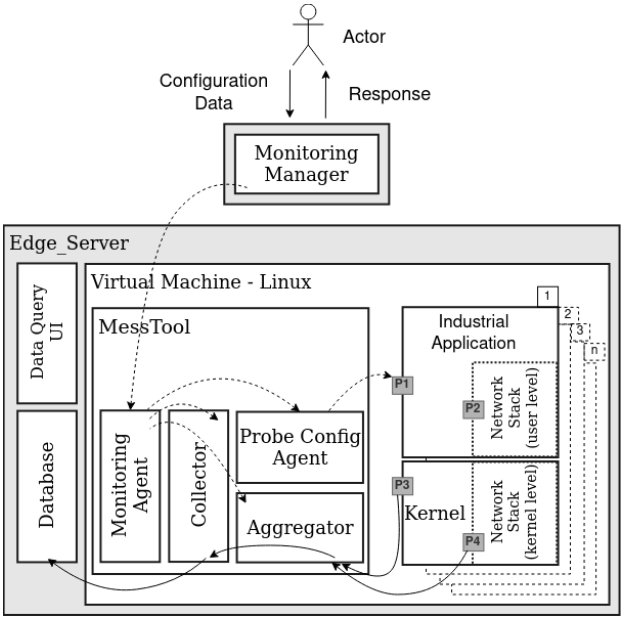

Fig. 3: MessTool Architecture

C. Component and Implementation Details

Figure 3 depicts an example deployment of MessTool components, where all components are placed inside a virtual machine. Additionally, a database (external to MessTool) and its dedicated User Interface (UI) are deployed natively onto the Edge Server system. Other deployment strategies are possible too, such as running all components directly on the Edge Server or even mixed setups. This is the most common one.

In Figure 3 various probes ("P1" to "P4") are configured to allow for measurements regarding a given industrial application that leverages the Network Stack. While "P1" depicts a probe allowing to trace events (functions) in user-level, "P3" allows to trace events inside the execution of the Kernel. Similarly, "P1" and "P2" depict special probes for the network stack, used by that application. Moreover, dashed arrows depict the flow of configuration data from the Monitoring Manager towards a given probe "P1", whereas solid arrows depict the flow of measurement data from probes "P3" and "P4" towards the database.

The user's main interface is provided through the Monitoring Manager. This allows a user to configure new measurement points in the system, query already configured ones, delete or update existing ones. For actual measurement retrieval an externally provided Data Query UI can be accessed by the user (e.g., allowing for a web-based GUI showing measurement plots), though the Monitoring Manager allows also to forward simple measurement queries to the Collector.

Each MessTool component provides two kinds of interfaces (a) for configuration and (b) for (measurement) data exchange. The configuration interface is provided through a REST-based API. A schema file was used to describe each version of each component's configuration API. This allowed for fast prototyping and was necessary also to decide which configuration information can be abstracted at which level of component, using a hands-on approach.

The implementation of configuration API is constraint by the requirement not to interfere with the execution of data transfers on the data exchange interfaces, whereas the implementation of the data exchange interface is constraint by the requirement that a data sink must not interfere with the execution at the data source.

The Collector is used to decouple the reception of measurement data (from the Aggregator) from the requirements/dependencies of the databases. This allows for support of multiple database interfaces and formats but also allows to use – MessTool-internally – specifically designed interfaces for data transfer. Furthermore, the implementation of the Collector is designed modular, in a way that allows for simple extension to new Database formats/interfaces. The Collector also holds additional (configurable) buffers to allow for batching sequences of measurements towards the database. This allows to maximize the efficiency of the connection to the database but also allows for those scenarios with a (temporarily) unavailable database connection.

The Aggregator retrieves event data from one or more probes and aggregates them into a single measurement. Additionally, the Aggregator is enabled to use received event data from probe A to annotate received event data from another probe B – assuming a given causal relationship between both events and that the relation can be determined (dynamically) by the Aggregator. If needed, the Aggregator allows to align event data with different time offsets from multiple probes, as long as the alignment can be determined statically. It should be noted that, while possible, the Aggregator is not designed to batch multiple measurements but provide only a single measurement (from multiple event sources) each time to the Collector.

The Aggregator-to-Collector data interfaces is implemented in a way that the Aggregator (source) is not influenced by the Collector (sink) regarding execution. For the examples as given in Section IV, an Inter-Process Communication (IPC) pipe was sufficient – configured in a way that data will be dropped in overload scenarios. The implementation allows to extend for other mechanisms, like memory-mapped regions, if the scenario demands it.

Finally, the mechanism to extract measurements of monitored events is encapsulated inside the Probe Config Agent (PCA). The PCA leverages two important features exposed by the Linux Kernel. These are, kprobes and eBPF. The kprobes are attachable software hooks provided by the Kernel, to which we can attach certain software blocks. In an event driven Linux Kernel, when an event occurs, specific functions are called by the Kernel to service the event. As a consequence, by monitoring the execution of a function in the Kernel, it is possible not only infer the occurrence of the event but also gain contextual information around the event. To this end, kprobe can be attached to a Kernel function that we wish to monitor and a corresponding eBPF code block can be attached to the hook provided by the kprobe. The Kernel transfers control to the eBPF code block immediately after the Kernel function is called and once the code block completes execution in Kernel context, control resumes to execute the body of the called Kernel function. Therefore, measurements taken in the eBPF code block are the most accurate as they are sampled at the closest proximity possible to the occurrence of an event. Only a dedicated Kernel module may take competing closest measurement. The PCA is responsible in attaching specific filter programs implemented in eBPF to the identified hooks. The eBPF program is designed to take a timestamp at the earliest and further apply filtering mechanism to narrow down to a specific event identified using a combination of parameters available at the moment in the execution context. The measurement data with the annotated timestamp are written into special buffers such that PCA can periodically poll and extract them. PCA then formats the measured data and shares it with the Aggregator.

IV. Evaluation

This section details on the evaluation of use cases using MessTool. As mentioned in Section III, MessTool's PCA leverages kprobe and eBPF functionality offered by the Linux Kernel. A measurement of the monitored event with the closest proximity possible to the occurrence of an event. Since the eBPF code is executed in the context of the kernel, measurements obtained by MessTool are extracted in the kernel thread. Therefore, MessTool as a framework, offers the user to target specific events and eBPF filter programs extract measurements of only the events that matter to the user.

A. Evaluation Setup

The MessTool framework is implemented as a micro-service based architecture, as shown in Figure 4, to analyze real-time applications deployed onto the edge node. That means, all components of MessTool were containerized and communicate over RESTful Application Programming Interfaces (APIs). Moreover, the MessTool operates on all Linux-based systems where the kernel provides native eBPF support. In addition, the Kernel is patched with PREEMPT_RT patch set, making the Kernel fully preemptible by the user application that is assigned a high priority. The execution environment emulates an environment of an edge cloud provider, where user applications are run inside a virtual machine deployed on top of an edge cloud server running on stock hardware.

In this evaluation, we have chosen two use cases that are encapsulated in a task block and are scheduled at highest priority in the preemptible Linux Kernel, such that the use cases enjoy low latency real-time treatment. It is through the interface of system calls, that the Kernel is able to arbitrate the provisioning of the hardware resources to the user applications. Therefore, when a higher priority application invokes a system call, the privilege of high priority is lost because the application now depends on the execution of lower priority Kernel tasks. Since the system calls are a shared interface, the latency is unbounded. This phenomenon is called priority inversion. To emulate such a scenario, we chose to invoke system calls related to transmission of network packets through the shared resource of the Network Interface Controller (NIC), that is shared by all applications on the system. With the MessTool framework we are able to measure the in-host delay and round trip time experienced by the high priority user application at chosen points in the software stack. In this section, for the measurements taken by MessTool, we make the following observations with regard to the precision of the annotated timestamp and the accuracy of the measurement:

1) Accuracy: The accuracy of a measurement depends on how close a timestamp is logged while monitoring the event. By design, eBPF probes are attached and executed in between the point of just after the Kernel function is called and the Kernel function's body. This way, the eBPF code attached to the a designated kprobe, can take a timestamp as early as possible when an event occurs (just after the Kernel function is called). Therefore, the timestamps taken by the probes in MessTool are considered the most accurate measurement feasible from state of the art.

It is to be noted that, to improve accuracy for cyclic applications, a systematic approach for process parameter and system configuration was used. We were using dedicated process priorities and CPU affinity pinning, as well as real-time scheduling (SCHED_DEADLINE). Due to the cyclic nature of applications that were hosted on the edge server, it allowed to minimize the negative influence from other applications hosted on the edge server.

2) Precision: The precision of a measurement depends on the accuracy of the system clock. The MessTool reports the timestamp measurements with nanosecond precision. This level of precision is enabled by the Linux kernel CLOCK_MONOTONIC clock. CLOCK_MONOTONIC is considered to be accurate and immutable as compared to other clock types. Moreover, the nanosecond precision of all timestamps in the MessTool framework is preserved and persisted within the database.

Figure 4 showcases the overall setup of the MessTool deployed within our evaluation setup.

B. Evaluated Use-Cases

This section presents two use-cases. For each use case we show how the MessTool framework can be used to collect required measurements. A summary of respective results is also provided.

1) In-Host Delay: For our first use case, an industrial application controls a remote end device. Control operations are sent to the end device using *UDP messages*. This allows the edge-hosted industrial control application to also gain feedback values from the end device.

The edge-hosted application leverages Linux' networking stack. Therefore, sending out a control message from the edge-hosted application to the end device will trigger system calls in the Linux networking stack. In a matter of speaking, each control message will traverse through the networking stack and, finally, brought to the dedicated NIC. Naturally, there are delays associated with the process getting initiated and for the Linux stack to complete the action. The MessTool framework can be used to measure these delays.

As seen in Figure 5, whenever the control application will send out another control message to the end device (via UDP) the *__sys_sendto* system call is triggered and the control message data is stored in a dedicated buffer within Linux kernel. This data buffer will also contain an embedded sequence number that allows to identify the transmission of a dedicated control message. As the data buffer is exchanged between functions in the software stack, the kernel allocates the socket buffer data structure.

As identified in the figure, the *entry point* marking the start of message transmission is the *__sys_sendto* system call. The *exit function dev_hard_start_xmit* is marking the last known function before the socket buffer leaves the Kernel. The packet is handed over to the NIC for scheduling the packet for its actual transmission.

The difference between timestamps obtained by the MessTool framework, between the events of *entry point* and *exit point*, constitutes the software stack delay unique to each control message.

The monitored user application was scheduled at highest priority of 99 and with scheduling policy SCHED_FIFO and transmits periodic User Datagram Protocol (UDP) messages with 1 s interval. The messages were directed to a remote I/O controller.

Figure 6 shows the histogram of the in-host delay experienced by the user application, where the delay is between the entry event of the calling *__sys_sendto* system call and the exit event of the Kernel calling the function *dev_hard_start_xmit*.

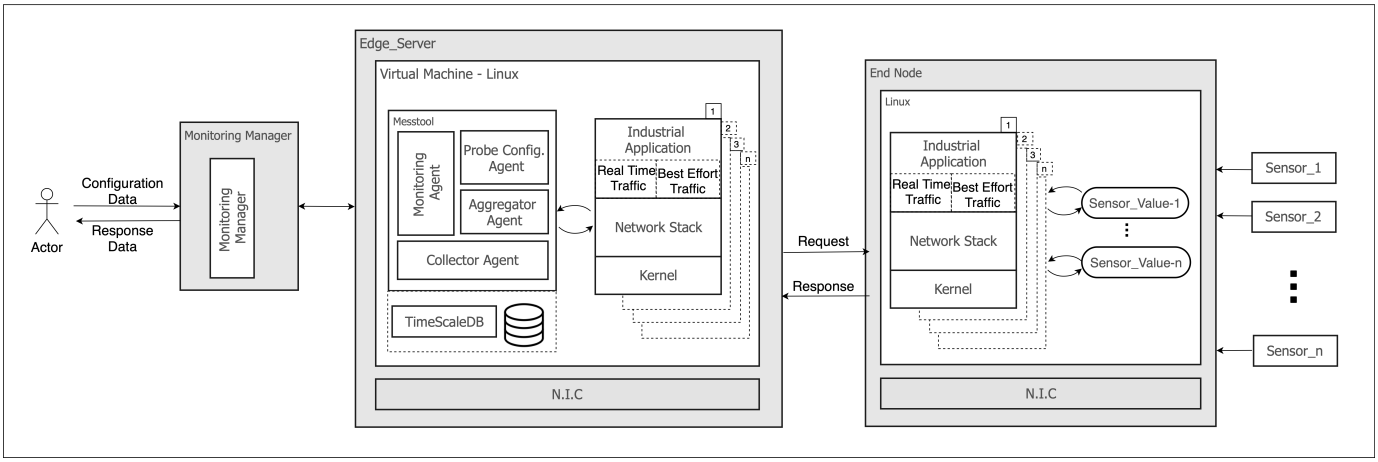

Fig. 4: MessTool: Evaluation Setup.

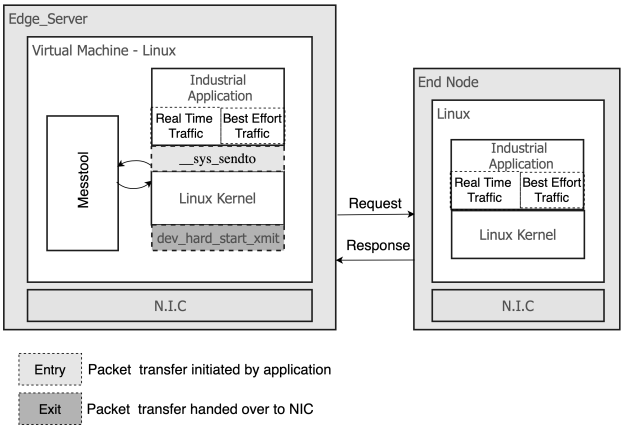

Fig. 5: Use-Case 1: In-Host Delay.

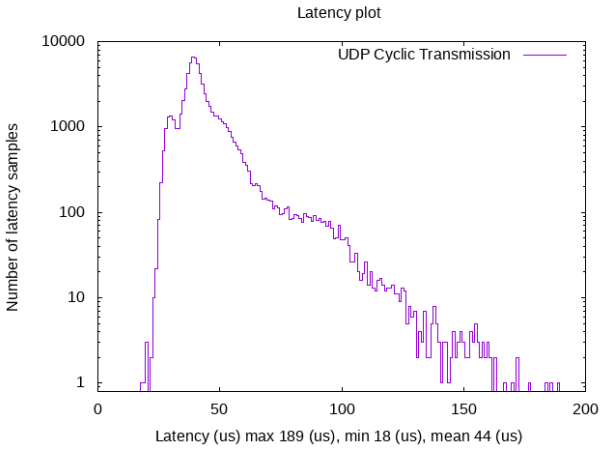

Fig. 6: Histogram of UDP Cyclic transmission run on Virtual Machine

The measurements lasted for over 24 hours and show that the MessTool framework can take measurements for longer duration.

2) Round-Trip Time: Usually, the Round Trip Time (RTT) of a network is measured using the Internet Control Message Protocol (ICMP) protocol. However, more relevant to a message-based remote-control application is the RTT of its actual messages handling, including in-host handling. The RTT can then be seen as the delay between the user process initiating the task in Linux user-space, followed by reporting it to Linux kernel, completing the task. It should be noted that this assumes a periodically executed message exchange.

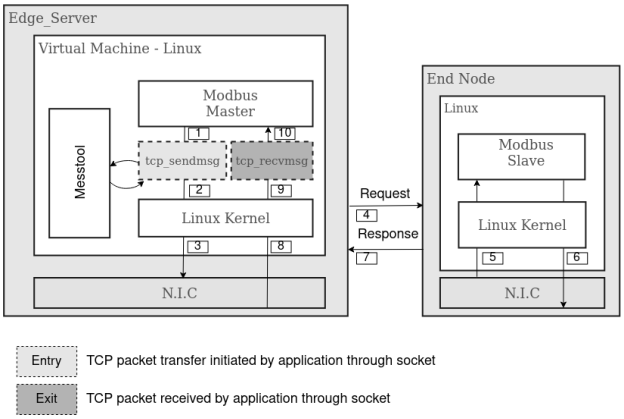

Fig. 7: Use-Case 2: Round Trip Time

Figure 7 shows the use-case diagram for measuring the RTT using the Transmission Control Protocol (TCP); Modbus-TCP to be precise. In this case, the MessTool framework is used for measuring the RTT against application exchanging periodic Modbus-TCP messages. The calculation of the RTT is done by taking the difference between a Modbus-TCP request sent from the master to the Modbus-TCP response received by the master. Since Modbus-TCP relies on TCP, a one-to-one relationship between messages from the master and corresponding responses from the slave, has to be established using context specific fields in the header of the TCP messages. We chose to use the sequence number provided by the Modbus-TCP protocol to establish the one-to-one relationship between request-response

message pairs. The numbers 1 to 10 describe the flow of events (seen globally) regarding the events of message handling.

For this use case, we chose the system calls *tcp_sendmsg* (to send the message from master) and *tcp_recvmsg* (master receiving the message) as entry and exit functions. Similar to the previous use case, necessary information for message identification can be retrieved from the context of each entry and exit event. The periodic cycle duration between successive messages transmitted by the master was set to 1 second and the application was scheduled with priority 99 using the scheduling policy *SCHED_FIFO*.

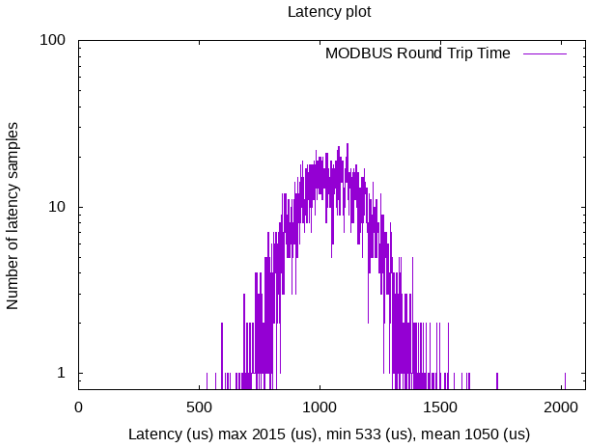

Fig. 8: Histogram of Modbus-TCP RTT on Virtual Machine

Figure 8 depicts the RTT experienced by a Modbus-Master hosted on an edge server.

V. Conclusion and Future Work

We foresee that future industrial infrastructures will be software-defined and that at the same time hard real-time behavior will still be critical in many industrial control use cases. Based on this premise, mechanisms are needed to assess the systems end-to-end time behavior in detail. Therefore, we proposed means to measure performance in distributed Edge Cloud environments exploiting extended Berkeley Packet Filters (eBPF). The main result include a framework that can be utilized to profile in detail the end-to-end time behavior of distributed edge applications. As the timestamps sampled by the probes are as accurate as eBPF / the time-stamping facility from the kernel it provides the most accurate measurement feasible from state of the art. Within our evaluation we have monitored a cyclic transmission of packets with a scheduled delay of under 190 µs and measured a round trip time under 2 ms. The presented work provides potential benefits for cloud and edge infrastructure providers enabling them to profile relevant applications to gain deep insights into their timing performance. This allows for estimating within which boundaries certain KPIs can be held.

Our short-term goal is to profile the end-to-end behavior of various vPLC applications that are deployed over a number of TSN interconnected edge nodes. In medium term, we plan to extend this work towards dynamically configuring the infrastructure ensuring specific SLAs by using a Central Network Controller (CNC) and Centralized User Configuration (CUC). The long-term goals include enabling complex wireless software-based industrial communication infrastructures to comply with real-time requirements, such as within future organic, self-optimizing, TSN-enabled 6G core networks.

Acknowledgment

Research for this paper was partially financed by the German Edge Cloud. We thank our project partners for their contributions and their collaboration to this research work.

References

[1] S. Ahmad et al. “Towards the Design of a Formal Verification and Evaluation Tool of Real-Time Tasks Scheduling of Iot Applications”. In: *Sustainability* 11 (2019).

[2] E. Bartocci et al. “Introduction to Runtime Verification”. In: *Lectures on Runtime Verification*. Ed. by Springer International Publishing. 2018, pp. 1–33.

[3] G. Bernat, A. Burns, and A. Liamosi. “Weakly Hard Real-Time Systems”. In: *IEEE Transactions on Computers* 50 (2001), pp. 308–321.

[4] G. Bruggen et al. *Systems with Dynamic Real-Time Guarantees in Uncertain and Faulty Execution Environments*. in 2016 IEEE Real-Time Systems Symposium (RTSS, 2016.

[5] Giorgio C. Buttazzo. *Hard Real-Time Computing Systems*. Real-Time Systems Series. Springer US, 2011, nil.

[6] I.-C. Chao et al. *Precise latency measurement of unidirectional-data-flow network equipment*. in 2014 IEEE International Frequency Control Symposium (FCS, 2014.

[7] S. E. Chodrow, F. Jahanian, and M. Donner. *Run-time monitoring of real-time systems*. nil: in Proceedings Twelfth Real-Time Systems Symposium, 1991.

[8] I. I. Consortium. “Time Sensitive Networks for Flexible Manufacturing Testbed - Description of Converged Traffic Types”. 2018.

[9] L. Cosart. “Packet network timing measurement and analysis using an IEEE 1588 probe and new metrics”. In: *2009 International Symposium on Precision Clock Synchronization for Measurement*. Control and Communication, 2009.

[10] T. Cruz et al. “Improving network security monitoring for industrial control systems”. In: *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM*. 2015.

[11] J.-D. Decotignie. “Ethernet-Based Real-Time and Industrial Communications”. In: *Proceedings of the IEEE*. vol. 93, 2005, pp. 1102–1117.

[12] S. Einspieler, B. Steinwender, and W. Elmenreich. *Integrating time-triggered and event-triggered traffic in a hard real-time system*. in 2018 IEEE Industrial Cyber-Physical Systems (ICPS, 2018.

[13] N. Finn. "Introduction To Time-Sensitive Networking". In: *IEEE Communications Standards Magazine* 2 (2018), pp. 22–28.

[14] M. Geier et al. "In Situ Latency Monitoring for Heterogeneous Real-Time Systems". In: *2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID*. 2019.

[15] A. Goodloe and L. Pike. *Monitoring Distributed Real-Time Systems: A Survey and Future Directions*. ACM Transactions on Embedded Computing Systems, 2010.

[16] V. Graveto et al. "A Stealth Monitoring Mechanism for Cyber-Physical Systems". In: *International Journal of Critical Infrastructure Protection* 24 (2019), pp. 126–143.

[17] M. Gutierrez et al. "Self-configuration of IEEE 802.1 TSN networks". In: *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA*. 2017.

[18] C. S. V. Gutirrez et al. "Real-time Linux communications: an evaluation of the Linux communication stack for real-time robotic applications". In: 2018 ().

[19] D. Hadžiosmanović et al. "Through the eye of the PLC". In: *Proceedings of the 30th Annual Computer Security Applications Conference on - ACSAC '14*. 2014.

[20] Oliver Hohlfeld et al. "Demystifying the Performance of XDP BPF". In: *Proceedings of the 2019 IEEE Conference on Network Softwarization: Unleashing the Power of Network Softwarization, NetSoft 2019*. 2019.

[21] F. Jahanian, R. Rajkumar, and S. C. V. Raju. "Runtime Monitoring of Timing Constraints in Distributed Real-Time Systems". In: *Real-time Systems* 7 (1994), pp. 247–273.

[22] M. Jevtic, V. Zerbe, and S. Brankov. "Multilevel validation of online monitor for hard real-time systems". In: *in24th International Conference on Microelectronics (IEEE Cat*. nil: No.04TH8716), 2004.

[23] A. Kane, T. Fuhrman, and P. Koopman. "Monitor Based Oracles for Cyber-Physical System Testing: Practical Experience Report". In: *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. 2014.

[24] K. Kavi, R. Akl, and A. Hurson. *Real-Time Systems: An Introduction and the State-of-the-Art*. John Wiley & Sons, Inc p. nil: in Wiley Encyclopedia of Computer Science and Engineering, 2009.

[25] H. Kim et al. "A Decentralized Approach for Monitoring Timing Constraints of Event Flows". In: *in* 2010 (2010), p. 31.

[26] H. Kim et al. "Task-aware virtual machine scheduling for I/O performance". In: *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments - VEE '09*. 2009.

[27] H. Kleines et al. "Measurement of Real-Time Aspects of Simatic/spl Reg/ Plc Operation in the Context of Physics Experiments". In: *IEEE Transactions on Nuclear Science* 51 (2004), pp. 489–494.

[28] M. Leucker and C. Schallhart. "A Brief Account of Runtime Verification". In: *The Journal of Logic and Algebraic Programming* 78 (2009), pp. 293–303.

[29] M. R. Lucas and D. M. Tilbury. "Methods of Measuring the Size and Complexity of Plc Programs in Different Logic Control Design Methodologies". In: *The International Journal of Advanced Manufacturing Technology* 26 (2005), pp. 436–447.

[30] B. Moore, T. Slabach, and L. Schaelicke. "Profiling interrupt handler performance through kernel instrumentation". In: *Proceedings 21st International Conference on Computer Design*. nil.

[31] R. Pellizzoni et al. *Hardware Runtime Monitoring for Dependable COTS-Based Real-Time Embedded Systems*. in 2008 Real-Time Systems Symposium, 2008.

[32] H. Prahofer et al. "A Comprehensive Solution for Deterministic Replay Debugging of Softplc Applications". In: *IEEE Transactions on Industrial Informatics* 7 (2011), pp. 641–651.

[33] G. Prytz. "A performance analysis of EtherCAT and PROFINET IRT". In: *2008 IEEE International Conference on Emerging Technologies and Factory Automation*. 2008.

[34] F. Reghenzani, G. Massari, and W. Fornaciari. "The Real-Time Linux Kernel". In: *ACM Computing Surveys* 52 (2019), pp. 1–36.

[35] D. Scholz et al. "Performance Implications of Packet Filtering with Linux eBPF". In: *in* 2018 (2018), p. 30.

[36] J.A. Stankovic. "Misconceptions About Real-Time Computing: a Serious Problem for Next-Generation Systems". In: *Computer* 21.10 (1988), pp. 10–19.

[37] W. Steiner et al. "Next generation real-time networks based on IT technologies". In: *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA*. 2016.

[38] Rick Sturm, Wayne Morris, and Mary Jander. *Foundations of service level management*. Vol. 13. Sams Indianapolis, IN, 2000.

[39] H. Tokuda, M. Kotera, and C. E. Mercer. "A Real-Time Monitor for a Distributed Real-Time Operating System". In: *ACM SIGPLAN Notices* 24 (1989), pp. 68–77.

[40] J. J. P. Tsai, K.-Y. Fang, and H.-Y. Chen. "A Noninvasive Architecture To Monitor Real-Time Distributed Systems". In: *Computer* 23 (1990), pp. 11–23.

[41] X. Zheng et al. "Efficient and Scalable Runtime Monitoring for Cyber-Physical System". In: *IEEE Systems Journal* 12 (2018), pp. 1667–1678.

[42] N. Zilberman et al. *Where Has My Time Gone?* Cham: in Passive and Active Measurement, 2017.