CLEAVE: Scalable and Edge-Native Benchmarking of Networked Control Systems

Manuel Olguín Muñoz
molguin@kth.se
KTH Royal Institute of Technology
Stockholm, Sweden

Neelabhro Roy
nroy@kth.se
KTH Royal Institute of Technology
Stockholm, Sweden

James Gross
jamesgr@kth.se
KTH Royal Institute of Technology
Stockholm, Sweden

ABSTRACT

As the number of cyber-physical systems rises, it becomes increasingly crucial to study Networked Control Systems (NCSs) combining control communication co-design. This nature of NCSs has led to task-specific approaches to research, creating a dearth of generalizable, repeatable, and scalable experimentation. Further, with the advent of edge computing solutions, it is of paramount importance to explore its relevance in such applications. In this work, we present CLEAVE, a novel, completely software-based framework for repeatable and scalable experimentation in edge native NCSs. Our approach is based on the emulation of physical plants communicating over a real network with software-based controllers. CLEAVE is designed and built for the edge, using Python3 and with full compatibility with industry-standard containerization solutions. Although designed for single-loop emulations, the flexibility afforded by the aforementioned characteristics allow our framework to be adapted to a multitude of complex scenarios.

We validate CLEAVE using an initial implementation of an inverted pendulum NCS. Our results showcase the utility of the tool as a repeatable, extensible, and scalable solution to NCS performance evaluation and benchmarking on the Edge.

CCS CONCEPTS

• **Networks** → *Network experimentation*; *Network performance analysis*; • **General and reference** → *Empirical studies*; *Measurement*; **Experimentation**; *Evaluation*; **Performance**; • **Computer systems organization** → **Sensors and actuators**; Robotic control; *Cloud computing*; *Client-server architectures.*

ACM Reference Format:
Manuel Olguín Muñoz, Neelabhro Roy, and James Gross. 2022. CLEAVE: Scalable and Edge-Native Benchmarking of Networked Control Systems. In *5th International Workshop on Edge Systems, Analytics and Networking (EdgeSys'22), April 5–8, 2022, RENNES, France.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3517206.3526272

1 INTRODUCTION

The number and applications of Cyber-Physical Systems (CPSs) [1] — i.e. systems in which a real, physical mechanism is controlled by a computer — have exploded in recent years. However, this rapid increase in adoption has mostly been limited to industrial contexts. Although CPSs present huge opportunities for all facets of society, they have yet to reach our daily lives in any relevant scale due to their stringent operational requirements. This is about to change, however, as with the advent of novel wireless communication technologies as well as networking paradigms, such as cellular 5G and edge computing [2], consumer-grade CPSs will be made possible. These technologies meet two key requirements of CPSs: real-time capabilities (through extremely low end-to-end latencies), and context- and locality-awareness, and will most likely become the backbone of CPS in the future.

Networked Control Systems (NCSs) [3], a type of CPS wherein multiple networked actuators and sensors form a part of the same automatic control system will benefit from the adoption of these technologies. Depending on the physical system being controlled, NCSs can have stringent timing and reliability requirements for communication that conventional cloud paradigms and cellular networks cannot meet [4]. This necessitates sophisticated tools for the performance evaluation of future system architectures, as well as novel NCS design paradigm.

One the one hand, related literature in NCSs leverages to a large extent theoretical models, at the price of being able to capture networked systems effects only on a coarse level. On the other, there exist several approaches when considering experimental methodologies. One such approach uses setups in which the complete system is built on top of real hardware. This approach is employed in the works of Baumann *et al.* [5] and Cuenca *et al.* [6]; in both of these, the authors implement their approach on physical testbeds. Conversely, other studies choose to instead use completely *simulated* NCS setups. The authors in [7] have opted for such an approach. These studies often employ combinations of physical and network simulation tools trying to capture the complex dynamics of NCSs. Finally, some experimental studies instead employ *virtualized* approaches, in which either (1) a real network interacts with a simulated or emulated control system [8]; or (2) a simulated network interacts with a real control system [9].

As evidenced above, experimental research in NCSs includes varied heterogeneous hardware and software platforms, methodologies and key performance indicators. This, in turn, leads to hardware, software, and methodology fragmentation, as different studies tend to prefer approaches more favored in their respective communities. Furthermore, existing studies tend to focus on individual aspects and components of a system, thus producing results which do not provide a complete image of the NCS. This has caused a gap in knowledge pertaining to the reproducibility and comparison of experimental studies on these systems.

EdgeSys'22, April 5–8, 2022, RENNES, France

ACM ISBN 978-1-4503-9253-2/22/04.
https://doi.org/10.1145/3517206.3526272

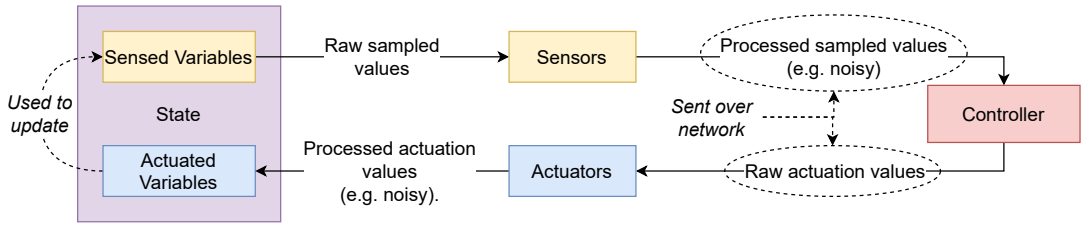


Figure 1: Structure of an emulated Networked Control System in CLEAVE.

Zoppi *et al.* [10] made the first (and to the best of our knowledge, the only) attempt at tackling this challenge in their work. In their work, they proposed a platform called NCSBench, to be used for reproducible benchmarking in NCS. Their methodology utilizes joint knowledge of control, computation, and communication. In their work various architectural elements and the corresponding delays associated with the NCS are modelled. Multiple experimental parameters and certain observable key performance indicators are defined and utilized in the implementation. This work however utilizes a physical LEGO® Mindstorms EV3 Core Set™ based plant for the implementation, preventing instantaneous changes in plant characteristics and component parametrizations. Furthermore, relying on physical objects like an inverted pendulum limits scalability of the experimentation in practice.

We overcome this issue by proposing a completely virtual plant allowing for unparalleled flexibility in changing the plant model and characteristic features of the experiments. In this paper, we present the first fully-software-based framework for scalable and repeatable benchmarking of edge-native NCS. As edge computing begins being adopted by industry, more and more variations have begun to appear in literature. "Near", "far", "core", and "telco" edge describe variations of the original concept and are becoming ubiquitous in new research. While the core idea of edge computing is widely accepted as fundamental for pervasive NCSs in general, understanding the strengths and weaknesses of such different edge concepts is of paramount importance.

Our framework, ControL bEnchmArking serVice on the Edge (CLEAVE), aims to simplify the repeatable and scalable benchmarking of such systems. It is fully virtualized, inspired by our previous work on benchmarking human-in-the-loop applications on the edge [11]. The tool consists of a benchmarking framework and software development kit for the development of emulated physical systems and softwarized controllers. These virtual NCSs can then be deployed on real networks for reparametrizable, repeatable, and reproducible benchmarking of the complete system.

CLEAVE is built using *Python 3.8*, making it highly extensible and able to harness the multitude of already existing user libraries. It is furthermore compatible with container technologies such as *Docker*[1], making it suitable for automated deployment, scaling, and benchmarking on industry-standard edge setups using container orchestration solutions.

The rest of this paper is structured as follows. Section 2 presents the design principles and implementation of the framework. In Section 3, we present a use case scenario which validates the utility, flexibility, and repeatability of CLEAVE. Finally, in Section 4 we conclude this work.

2 THE CLEAVE FRAMEWORK

In this section, we detail our CLEAVE framework for the performance evaluation of Networked Control Systems, with an emphasis on edge deployments. Our design follows a virtualized emulation approach; providing a framework for the real-time *emulation* of physical control systems and the easy implementation of softwarized controllers interacting with *real* networks. CLEAVE is thus a platform with an easy-to-use Application Programming Interface (API) on which a multitude of NCS types can be emulated. This allows for great flexibility, as the core components of the NCS can be switched out while maintaining the realism of the network, the most complex and limiting component of edge NCSs.

The framework is distributed as *free and open-source* software through the *GitHub* organization of the *KTH ExPECA* research group[2]. We also plan to provide a library of NCS implementations, making the tool even more accessible.

2.1 Design and Implementation

The design of CLEAVE attempts to target a broad a set of different control systems as possible, and tries to minimize the number of assumptions made about the systems implemented on top of the framework. These are:

(1) Controllers can be stateful or stateless, but they exclusively control a single plant.
(2) Plants and controllers interact in a "request-response" manner. We currently don't support setups in which the controller generates an actuation command without there first being a request from the plant.
(3) Controller and plant communicate over a TCP/IP network.
(4) The physical state behavior of the plant can be implemented in a discrete-time manner.
(5) Values associated to sensors and actuators can be represented as integers, floating point numbers, booleans, or sequences of bytes.
(6) A plant has zero or more sensors attached to it, with potentially different sampling rates.
(7) A plant has zero or more actuators attached to it.

Figure 1 shows an overview of the conceptual structure and operation of CLEAVE. The *State* implements the discrete-time behavior of the Plant. At the beginning of each time-step, actuated

[1]Docker Engine: https://www.docker.com/

[2]https://github.com/KTH-EXPECA/CLEAVE

variables in the *State* are updated with values obtained from the *Actuator* objects. Conversely, at the end of each time-step, sensed variables are sampled and used to update values associated with *Sensor* objects. These values are processed at the *Sensors* (e.g. to add noise) before being sent over the network to the *Controller*. The *Controller* calculates actuation values and updates the *Actuators* over the network. Finally, the *Actuators* process the actuation values and hold the results until they are read at the next time-step.

Implementation-wise, CLEAVE is built on top of the well-established and stable *Twisted*[3] asynchronous programming and networking framework. The aforementioned core components (*State*, *Controller*, *Sensors*, and *Actuators*) are provided as Abstract Base Classes (ABCs) with core functionality that users must extend for their specific use cases and NCSs. The framework makes no assumptions about the inner workings of the user implementations of these classes other than the interfaces they expose. Users are therefore free to implement arbitrarily complex functionality in these components. On the other hand, CLEAVE handles network communication and metric collection autonomously; users only need to select the desired transport protocol (we do however provide an API for extending the available transports).

Below we present a brief overview of the design and implementation of the core components.

State Base Class: ABC with an abstract method `advance()` which must be extended with the implementation of discrete-time emulation of the plant. The `advance()` method is called by the framework periodically according to the configured emulation update rate, and should perform a single time-step of the discrete-time emulation. Two optionally extensible methods, `initialize()` and `shutdown()` are provided to implement single-fire set-up and teardown procedures.

Controller Base Class: Defines a single abstract method `process()`, taking a mapping of names to values corresponding to the sensed variables of the *State*, and returning a similar mapping corresponding to the actuated variables. This method is called by CLEAVE in an event-based manner whenever samples are received from the plant side; however, we intend to extend this to time-interval-based approaches.

Sensor Base Class: Implements processing of monitored stated variables before sending them over the network to the *Controller*. This base class must be initialized with a property name (corresponding to the monitored *State* attribute) and a sampling rate in Hz, and provides an abstract method `process_sample()` which must be extended with an implementation of the processing performed on each sample.

Actuator Base Class: Performs the processing of actuation values obtained over the network from the *Controller*. It is initialized with a property name corresponding to the name of the actuated *State* attribute, and provides two abstract methods which must be implemented by extending classes: (1) `set_value()`, called by the framework whenever a new value for the corresponding actuated attribute is received; and (2) `get_actuation()`, called on every time-step and must return a value to apply to the actuated attribute.

[3]https://twistedmatrix.com

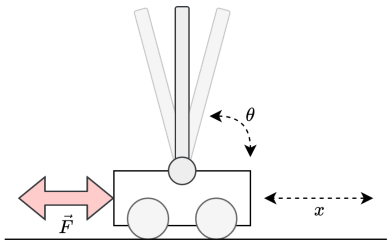


Figure 2: The 2D inverted pendulum system.

2.2 Configuring and Executing an NCS

Once the core components for the emulation have been defined and implemented, they need to be registered with the framework. This is done through Python scripts which define a number of required (and some optional) top-level variables, corresponding to emulation parameters such as the *State*, *Controller*, *Sensor*, and *Actuator* subclasses to use; the emulation update rate; the controller-side host and address; data output directories; *et cetera*. These Python configuration files are then provided as arguments to the `cleave` command line provided by the framework. CLEAVE imports and parses them, and then sets up and executes the emulation.

Although the core design of CLEAVE follows a single-loop approach, multi-loop scenarios can be easily implemented through the combination of multiple CLEAVE instances. The framework does not make any assumptions about the individual components other than the signatures of methods on the ABCs that must be extended and implemented. Combined with the Python emulation configuration scripts, this makes development of more complex scenarios merely a matter of implementing the desired functionality within the framework itself.

2.3 Current Implementations

For the purpose of this work, we have used CLEAVE to implement an emulated *inverted pendulum* control loop (see Figure 2), the goal of which is to balance the vertically free-swinging pendulum by applying horizontal forces on the cart. We have chosen this system as an initial benchmark for its relative simplicity as well as prevalence in the field of automatic control as one of the fundamental examples of linear control.

The inverted pendulum plant *State* is implemented as a real-time discrete-time physical emulation using CLEAVE's API and a 2D physics library[4], updated at a constant 120 Hz (this value is configurable; in this case it corresponds to the maximum achievable stable rate on our available hardware). *Sensors* for the angle, position, angular velocity, and velocity of the *State*, and an *Actuator* for the horizontal force applied to the cart are implemented as "perfect", i.e. values are returned as-is, without any added noise. For the controller side, a proportional-differential strategy is implemented using the framework *Controller* API and the *NumPy* numeric computation library[5]. Plant and controller are then packaged into containers, for ease of orchestration and reparametrization, and to mimic real-world deployment.

[4]Pymunk: http://www.pymunk.org/en/latest/
[5]NumPy: https://numpy.org/

A previously mentioned, we eventually plan to build an open library of emulated control loops.[6] The inverted pendulum plant is merely a proof-of-concept, and CLEAVE could be used to implement much more complex physical systems with even more stringent latency requirements. For instance, the flexibility afforded by the choice of programming language and the abstractions detailed in Section 2.1, allow for relatively straightforward emulation of autonomous drone dynamics. Propeller dynamics could be implemented by *Actuator* objects, gyroscopes by *Sensors*, and the physical behavior of the plant could be modeled using a general purpose 3D physics engine (*e.g.* the Panda3D[7] engine).

3 USE CASE VALIDATION

In this section, we demonstrate the utility of CLEAVE by walking readers through an example use case. With this, we aim to showcase the ability of our framework to provide accurate and repeatable measurements of the performance of NCSs deployed on edge computing infrastructure.

We present the following use case scenario. A simple NCS consisting of an *inverted pendulum* plant controlled by a *proportional-differential* controller is to be deployed on an Edge server. The connection between plant and server is over a WiFi (IEEE 802.11n) link. Moreover, there are a number of video-streaming applications running concurrently on the Edge setup.

We believe this to be an interesting and representative use-case scenario for a number of reasons. Firstly, the inverted pendulum is broadly used as a benchmark in control-system literature, see for instance [5] and [9]. A key reason for this is that such a relatively simplistic system allows for straightforward reparametrization to obtain varied system dynamics, which in turn makes a broader range of experiments possible. Secondly, similar, albeit non-Edge-enabled, setups have been a reality in industry for almost two decades, as evidenced by [3]. Thirdly, although real deployments would most assuredly employ mobile technologies due to the added benefits of mobility and range, the Round-Trip Times (RTTs) offered by these technologies are higher, or at best equal, those offered by WiFi. 4G, for instance, has average RTTs of around a few hundred milliseconds, which is orders of magnitude greater than the values measured in our scenario, and current 5G deployments offer RTT values barely matching the sub-10 ms values we obtain for WiFi. Thus, we argue that using WiFi allows for more experimental freedom, as the system can be tweaked to obtain worse RTTs more akin to those of mobile technologies while still having the option to study best-case scenarios with very low-latencies. Finally, video analytics is one of the main proposed use cases for edge computing [12–14], and thus we foresee edge NCS deployments being deployed in parallel with such applications in the future. However, before deployment can become a reality, key questions such as: "what are the baseline requirements of the NCS?"; "what are the achievable best-case end-to-end latencies in the system?"; and "how does the concurrent video-streaming traffic affect NCS stability?" need to be answered.

[6] At the moment of writing, we are in the process of implementing *model-predictive* as well as *deep neural network* controllers for the inverted pendula plants.
[7] https://docs.panda3d.org/1.10/python/programming/physics/builtin/indexhttps://docs.panda3d.org/1.10/python/programming/physics/builtin/index

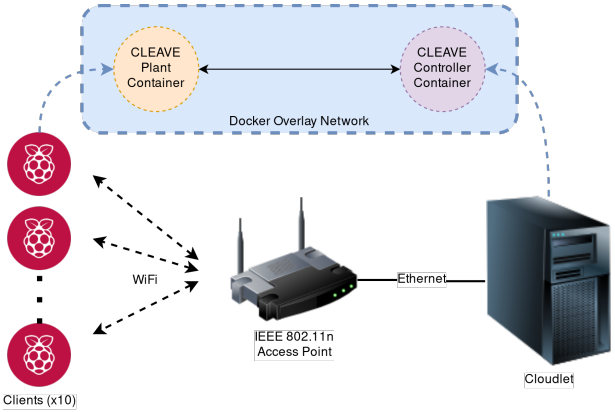


Figure 3: The setup used for our experimentation.

We attempt to study these questions using the experimental setup in Figure 3. CLEAVE is deployed on a testbed consisting of 10 Raspberry Pi 4B clients connected wirelessly to an IEEE 802.11n Access Point (AP). Connected to the Ethernet backbone of this AP is a general-purpose `x86_64` desktop computer acting as a Cloudlet/Edge server.

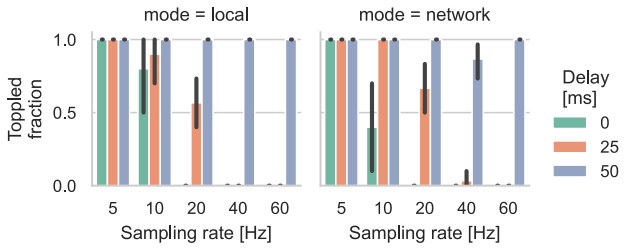
The first step in our case study is to evaluate the baseline performance of the control system, both locally and over the network. We configure CLEAVE to run a single loop with both plant and controller co-located on the Edge server, and we also configure a separate setup with an identical loop executed over the wireless link. For both of these setups, we execute a series of scenarios varying: (1) the sampling rate of the Plant state, setting it to 5, 10, 20, 40, or 60 Hz; and (2) the responsiveness of the Controller, adding fixed delays of 0, 25, or 50 ms after the processing of each sample.

We run repeated executions of each combination of these parameters at least 10 times, for both the networked and "local-only" setups. Each execution lasts for 5 min, during which we collect detailed data on both the state of the controlled system as well as on the data sent over the network. After the initial 10 runs, we identify that setups with sampling rates of 5 and 10 Hz were consistently too unstable to consider, and disregard them in further analyses. We further note that setups with 0 ms additional delay are always stable and thus also disregard them. Remaining setups, i.e. sampling rates $\geq$ 20 Hz and artificial delays $\geq$ 25 ms, are repeated an additional 20 times for better statistical significance.

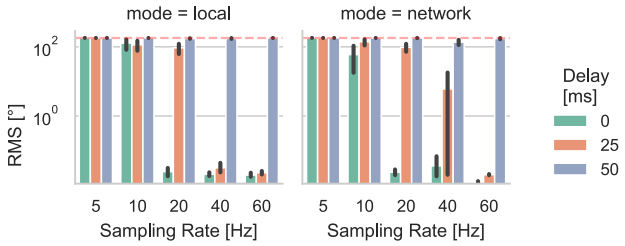
This repeated experimentation and data collection while "zooming in" on particular setups is facilitated by CLEAVE's design. Scenarios are executed automatically in batches using a simple Python script which interacts with Docker through the widely adopted *docker-py*[8] library. This is CLEAVE's first advantage over existing frameworks; it is designed with cloud and edge technology and paradigms in mind, making integration with existing systems convenient.

Figure 4 shows a summary of the results relating to the single-loop scenarios: Figure 4a shows the fraction of plants that toppled in each scenario, and Figure 4b shows the average Root Mean Square (RMS) for the absolute pendula angles. Figure 4c shows

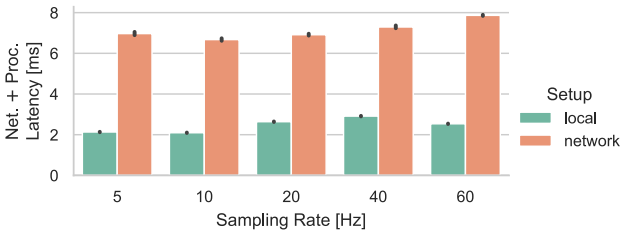
[8] Docker SDK for Python: https://docker-py.readthedocs.io/en/stable/



(a) Fraction of toppled plants.

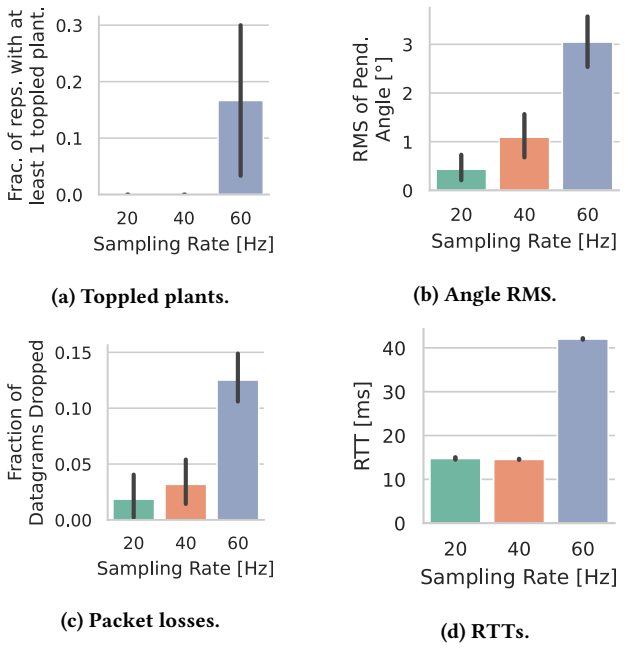


(b) Mean angle RMS; red line indicates $y = 180$.



(c) Mean RTTs.

Figure 4: Baseline results. Error bars indicate 95 % Confidence Intervals (CIs).



(a) Toppled plants.

(b) Angle RMS.

(c) Packet losses.

(d) RTTs.

Figure 5: Multi-loop, resource-constrained setup results. Error bars indicate 95 % CIs in all plots.

average latency due to network and processing (excluding synthetic delays) for both single-loop scenarios. Packet losses were below 0.2 % for all parametrizations of the single-loop scenario over the network, and 0 % for all parametrizations of the local-only setup.

As expected, higher sampling rates tend to correlate with better quality of control; at higher sampling rates the system was able to reach stability at higher RTTs. These initial results already hint at interesting consequences for such an edge-bound NCS deployment. For instance, it is clear from Figures 4a and 4b that network delays can, to a certain extent, be compensated for by increasing the sampling rate of the system. A corollary of this is, conversely, that at lower network latencies NCSs are able to stabilize at lower sampling rates. Adaptive sampling might thus be a viable method for optimizing resource utilization.

Once baselines for single loops in the use case scenario have been established, we can proceed to studying the interaction between the NCSs and the video-streaming applications. We deploy 6 control loops on the experimental setup depicted in Figure 3. On the remaining 4 clients we run the *iperf3*[9] traffic load generator, each generating 6.5 Mbit/s of uplink User Datagram Protocol (UDP) traffic. This emulates the load generated on the network by 1080 p Full-HD video streaming, originating from the clients and terminating in the cloudlet. Based on the baseline results, we execute setups with NCS plant sampling rates of 20, 40 and 60 Hz. Each sampling rate configuration is run for 5 min, and then repeated 30 times to obtain statistical significance. Once again, repetitions of this scenario are executed automatically in batches using a simple Python script and *docker-py*.

Figure 5 shows a summary of the results obtained. Figure 5a shows the fraction of repetitions of each setup in which *at least* one plant failed to maintain stability and toppled. Figure 5b shows the RMS for the pendulum angles for each setup, only considering data from plants that did not topple. Figure 5c shows the fraction of UDP datagrams dropped, averaged over all plants and repetitions per setup. Figure 5d shows the measured end-to-end plant-side RTT, averaged over all plants and repetitions per setup.

These results are interesting in their counter-intuitiveness compared to the single-loop baseline results. While the baselines might lead us to think that higher sampling rates are always better for the stability of control systems, Figures 5a and 5b show this not to be the case for NCS in resource-constrained scenarios. 60 Hz was the least stable configuration, with at least one pendulum toppling in around 16 % of the repetitions, and mean pendulum angle RMS circa 3 times that of the 40 Hz scenario. 40 Hz was in turn the second worst configuration — although it presented no toppled pendula, average angle RMS doubled that of the 20 Hz setup.

Figures 5c and 5d explains these behaviors. Whereas both the 20 and 40 Hz setups show losses well below 5 %, the 60 Hz scenario shows an average of around 13 % of datagrams lost. The differences in RTTs results are equally telling; RTTs for the 60 Hz setup were

[9] iperf3: https://iperf.fr/

on average approx. 3 times those for the 20 and 40 Hz setups. These results stem from the contention for network resources, and hint at important trade-offs system designers will have to take into consideration when designing and developing NCSs for deployment on the Edge. The Edge will be *multi-tenant* and *multi-instance*. NCS deployments will have to be designed with shared resources in mind, and given the complexity of these systems, experimental tools like CLEAVE will be key for their succesful adoption and massification.

4 CONCLUSION

The issue of repeatable and scalable benchmarks has been largely glossed over in NCS literature, as existing experimental research studies tend to implement *ad-hoc* solutions.

In this work, we aimed to tackle this issue through a fully software-based framework for repeatable, reproducible, and easily scalable NCS benchmarking with a particular focus on edge deployment. We argue our approach, CLEAVE, embodies a better solution than previous work for a number of reasons:

(1) Compared to fully physical approaches, such as those used in [5] and [6], our approach allows for greater flexibility and scalability. The aforementioned approaches rely on specialized and sometimes entirely custom-built physical platforms, and although flexible and cheap approaches such as Zoppi *et al.'s* — which uses a LEGO-based physical plant — exist, these still do not reach the level of flexibility afforded by a fully software-based framework. Experimenters still need copies of the hardware, making anything other than small-scale setups unfeasible. In contrast, our approach requires only general-purpose computing platforms, and can be employed by basically anyone with access to a computer. Scalable deployments can in turn easily and cheaply be set up using single-board computers and/or virtual cloud instances.
(2) When compared to simulated approaches such as [7], CLEAVE provides a higher level of realism, in particular with regards to the network segment of the system.
(3) Finally, although it shares much in common with previous emulated approaches such as the one employed in [8], CLEAVE has an advantage by specifically targeting a general-purpose approach using industry-standard, cloud- and edge-native tools and software. The tool can easily be deployed and scaled using widely-used frameworks such as Docker Swarm and Kubernetes.

We validate the utility of this tool through an example use case approximating a proposed edge deployment of inverted pendula control loops co-located with video analytics services. We argue such a use case represents a realistic scenario and appropriate benchmark for the tool, since (1) the inverted pendulum plant is ubiquitous in NCS research; (2) similar setups exist in real-world industrial use; and (3) video analytics has long been proposed as a "killer app" for edge computing. Our results showcase the ability of the framework to extract relevant metrics relating to the stability of the control system, as well as on the performance of the underlying network link. We believe CLEAVE represents an important step towards enabling inexpensive and low-complexity scalable research for real-world deployment of edge-bound NCSs.

There is still, however, work to be done. We are extending the number of plant and controller implementations on the framework, with the goal of creating an open library of NCSs to share with the community. At the moment, the interactions of CLEAVE and tools such as Docker are still quite superficial. Our goal is to achieve a much tighter integration, e.g. by providing the toolkit as pre-packaged container images. Finally, the validity of the results obtained by the framework will have to be verified through more thorough, realistic scenarios than what we have been able to show in this work. In particular, we intend to perform large-scale experimentation targetting 5G cellular deployments, as this technology is set to become the backbone of edge networks in the near future.

ACKNOWLEDGEMENTS

This research has been partially funded by (1) the VINNOVA Competence Center for Trustworthy Edge Computing Systems and Applications (TECoSA) at KTH Royal Institute of Technology; and (2) the Swedish Foundation for Strategic Research (SSF), through grant number ITM17–0246.

We also wish to thank S. S. Mostafavi, V. N. Moothedath, M. Alsakati, O. Ferm, S. Vojcic, and A. Bhattacharya for their support and valuable contributions to this work.

Finally, we would like to thank the reviewers for their insightful comments and suggestions, which greatly helped us improve this work.

REFERENCES

[1] Ragunathan Rajkumar et al. "Cyber-physical systems: The next computing revolution". In: *Proceedings of the Design Automation Conference*. 2010, pp. 731–736.
[2] Mahadev Satyanarayanan. "The Emergence of Edge Computing". In: *Computer* 50.1 (2017), pp. 30–39.
[3] Rachana Ashok Gupta and Mo-Yuen Chow. "Networked Control System: Overview and Research Trends". In: *IEEE Transactions on Industrial Electronics* 57.7 (2010), pp. 2527–2535.
[4] Shaohua Wan et al. "Efficient computation offloading for Internet of Vehicles in edge computing-assisted 5G networks". In: *The Journal of Supercomputing* 76.4 (2020), pp. 2518–2547.
[5] Dominik Baumann et al. "Evaluating Low-Power Wireless Cyber-Physical Systems". In: *Proceedings of the First IEEE Workshop on Benchmarking Cyber-Physical Networks and Systems (CPSBench'18)*. 2018, pp. 13–18.
[6] Ángel Cuenca et al. "Periodic Event-Triggered Sampling and Dual-Rate Control for a Wireless Networked Control System With Applications to UAVs". In: *IEEE Transactions on Industrial Electronics* 66.4 (2019), pp. 3157–3166.
[7] Yehan Ma et al. "Optimal Dynamic Scheduling of Wireless Networked Control Systems". In: *Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems*. Association for Computing Machinery, 2019, 77–86.
[8] Yu Wang et al. "Inverter-Based Voltage Control of Distribution Networks: A Three-Level Coordinated Method and Power Hardware-in-the-Loop Validation". In: *IEEE Transactions on Sustainable Energy* 11.4 (2020), pp. 2380–2391.
[9] O.R. Natale et al. "Inverted pendulum stabilization through the Ethernet network, performance analysis". In: *Proceedings of the 2004 American Control Conference*. Vol. 6. 2004, 4909–4914 vol.6.
[10] Samuele Zoppi et al. "NCSbench: Reproducible Benchmarking Platform for Networked Control Systems". In: *Proceedings of the 17th IEEE Annual Consumer Communications Networking Conference (CCNC'20)*. 2020, pp. 1–9.
[11] Manuel Osvaldo J. Olguín Muñoz et al. "EdgeDroid: An Experimental Approach to Benchmarking Human-in-the-Loop Applications". In: *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*. ACM, 2019, pp. 93–98.
[12] Ganesh Ananthanarayanan et al. "Real-Time Video Analytics: The Killer App for Edge Computing". In: *Computer* 50.10 (2017), pp. 58–67.
[13] Shanhe Yi et al. "LAVEA: Latency-Aware Video Analytics on Edge Computing Platform". In: *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. Association for Computing Machinery, 2017.
[14] Junjue Wang et al. "Bandwidth-Efficient Live Video Analytics for Drones Via Edge Computing". In: *Proceedings of the Third IEEE/ACM Symposium on Edge Computing (SEC)*. Association for Computing Machinery, 2018, pp. 159–173.