Enabling Wireless Network Support for Gain Scheduled Control

Sebastian Gallenmüller*, René Glebke†, Stephan Günther*, Eric Hauser*,
Maurice Leclaire*, Stefan Reif‡, Jan Rüth†, Andreas Schmidt§,
Georg Carle*, Thorsten Herfet§, Wolfgang Schröder-Preikschat‡, Klaus Wehrle†
*Technical University of Munich, {gallenmu,guenther,hauser,leclaire,carle}@net.in.tum.de
†RWTH Aachen University, {glebke,rueth,wehrle}@comsys.rwth-aachen.de
‡Friedrich-Alexander University Erlangen-Nürnberg, {reif,wosch}@cs.fau.de
§Saarland Informatics Campus, {andreas.schmidt,herfet}@cs.uni-saarland.de

Abstract

To enable cooperation of cyber-physical systems in latency-critical scenarios, control algorithms are placed in edge systems communicating with sensors and actuators via wireless channels. The shift from wired towards wireless communication is accompanied by an inherent lack of predictability due to interference and mobility. The state of the art in distributed controller design is proactive in nature, modeling and predicting (and potentially oversimplifying) channel properties stochastically or pessimistically, i. e., worst-case considerations. In contrast, we present a system based on a real-time transport protocol that is aware of application-level constraints and applies run-time measurements for channel properties. Our run-time system utilizes this information to select appropriate controller instances, i. e., gain scheduling, that can handle the current conditions. We evaluate our system empirically in a wireless testbed employing a shielded environment to ensure reproducible channel conditions. A series of measurements demonstrates predictability of latency and potential limits for wireless networked control.

CCS Concepts • **Networks** → *Network experimentation*; • **Computer systems organization** → *Embedded and cyber-physical systems.*

Keywords edge computing, networking, control, latency-awareness, gain scheduling, reproducible wireless measurements

ACM Reference Format:
Sebastian Gallenmüller, René Glebke, Stephan Günther, Eric Hauser, Maurice Leclaire, Stefan Reif, Jan Rüth, Andreas Schmidt, Georg Carle, Thorsten Herfet, Wolfgang Schröder-Preikschat, Klaus Wehrle. 2019. Enabling Wireless Network Support for Gain Scheduled Control. In *2nd International Workshop on Edge Systems, Analytics and Networking (EdgeSys '19), March 25, 2019, Dresden, Germany.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3301418.3313943

1 Introduction

Control methodologies for physical processes, ranging from self-stabilizing inverted pendulums, over cruise control in cars, up to maintaining the safe operation of large-scale chemical reactors, were built assuming the underlying hardware and communication networks behave highly predictable. Shielded, single-purpose wired connections, as well as synchronous networks [14], enable controllers to assume that—if not for the failure of some equipment—their control decisions are both based on the most recent available sensor readings and executed by the actuators at a specific, predefined time in the future.

Recent advances in communication and systems architecture, however, have begun to challenge the practicability of this approach. Instead of operating a cyber-physical system in isolation, a plethora of embedded systems communicate, coordinate, and collaborate. A key change in the system architecture is to migrate control logic to an *edge* component [16] that has a global view on all individual systems, and hence allows for more efficient control algorithms, sophisticated data processing, and also serves as a bridge to cloud systems while keeping the communication latency low [3, 13]. As a side effect of this trend, the communication architecture shifts from dedicated wires to wireless connections for multiple reasons [1]. First, the growing number of devices disallows for expensive point-to-point connections. Second, future smart systems that are scattered over a large area render wired links impractical. Third, wireless communication is an essential requirement for mobile systems such as autonomous vehicles. While wireless networks allow building cost-efficient, large, and mobile systems, they also make connections less predictable—signals may degrade due to interference, resulting in information delay or loss. Controllers unaware of such situations may assume wrong system states, causing processes to run out of control with potentially disastrous consequences.

Gain scheduling breaks down the controller for a non-linear problem to a linearized *family of controllers*, and selects the controller most closely matching the current operating conditions—a proven, viable method for adapting to varying operation conditions [8]. Applications of gain scheduling include autopilots and flight control systems, or engine control in cars [12], and the approach has long been shown to guarantee stability, robustness, and performance, especially for slowly changing operating conditions [15]. In 2004, Tzes and Nikolakopoulos presented a set of controllers for a wirelessly controlled system based on the idea of gain scheduling [19], and various studies thereafter also applied gain scheduling to controllers facing network delay [17]. While the control theory community has proposed numerous methods for modeling the delays exhibited by (wireless) communication channels (cf. [17] for an overview), to the best of our knowledge, little to no work has been done in taking actual *measured* artifacts of wireless communication into account. We believe that live monitoring of network behavior, e. g.

ACM ISBN 978-1-4503-6275-7/19/03…$15.00
https://doi.org/10.1145/3301418.3313943

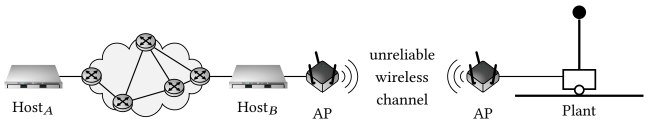

Figure 1. Networked control system topology

network stack or driver behavior, can *enable the network conditions to be more tightly integrated into the set of operating conditions in gain scheduling*, and thus allow the designers of control algorithms to integrate knowledge about the network's current *and upcoming* behavior into their design process.

In this paper, we explore the possibility of integrating live feedback from a wireless communication network into the gain scheduling approach to controller generation from a technical point of view. We start by illustrating a typical networked control system (Section 2) and present related work on control over wireless networks (Section 3). We then design and implement a prototypical framework that enables the live monitoring of delays and channel conditions of a wireless network based on the combination of an instrumentable real-time transport protocol (Section 4) and an automated shielded testbed for wireless communication (Section 5). Afterwards, we provide an empirical evaluation of our approach to the collection of channel state information (Section 6) before reasoning on the optimization potential for gain scheduled control approaches based on our method.

2 System Model

Figure 1 shows a physical system or plant attached to a network via a radio link. Two hosts A and B are connected to the network differing in their distance (number of hops) to the plant. Adding a control application, running on either of the hosts, turns this into a networked control system. The plant itself and the inherent properties of the actual control process determine the requirements for the network connection, such as maximum delay, number of exchanged messages, and maximum packet loss.

We propose to place the control application as close as possible to the controlled plant to reduce potential network delay or avoid network bottlenecks. In case of the system in Figure 1 we would prefer Host$_B$ over Host$_A$ for running control application due to the lower distance from the plant. Despite shortening the link between the host and the plant, network behavior may change over time especially if wireless links are involved. Gain scheduling allows us to react to these changes by selecting a control application which fits best to the current operating conditions of the underlying network. In this work, we do not provide a complete network control system that dynamically adapts to rapidly changing network conditions and solves all issues of networked control systems in general. We rather focus on two important aspects of such a system: First, we investigate a protocol equipped with in-band live-monitoring features that can be used to collect the information necessary for gain scheduling (Section 4). Second, we evaluate how this protocol behaves on a wireless link, through a series of reproducible network experiments (Section 6). Due to the unreliable nature of wireless links, this part of the network connection is the most challenging component of the control system.

3 Related Work

Our work focuses on the synergies and challenges when combining network and control in networked cyber-physical systems. A recognized area of problems (cf. [5]) analyzes real-time capabilities of different QoS schemes in IEEE 802.11 wireless systems and finds that the mechanisms are unsuitable even in low-demand situations. To this end, research proposes various cross-layer interaction schemes to overcome some of these challenges. Nakashima et al. [9] propose utilizing the deterministic nature of TDMA multi-hop wireless systems to account for delay and round trip propagation times in their control design. Nikolakopoulos et al. [10] utilize gain scheduled control with the help of the wireless hop count in a multi-hop wireless network to schedule different controllers. Xia et al. [20] use properties of the wireless PHY layer together with deadline missratios of the control application to design a cross-layer adaptive feedback scheme that dynamically adapts to the channel. While these works share the ideas to utilize cross-layer synergies, the assumptions on the wireless channels and their operating principles are often idealized and only simulated. In contrast, our work utilizes commodity wireless adapters in an actual testbed to investigate the effects of the interplay of communication and control for gain scheduled systems. Especially, the periodicity of control traffic promises valuable insights into the synergies of communication and control. For instance, Tian and Tian [18] develop a Markov model for real-time periodic traffic together with the IEEE 802.11 MAC layer and compare it to NS-2 simulations. While such models are valuable to control, it is not known how well these models map to real-world systems where the interplay of hardware alone may alter theoretic constraints. Therefore, our work strives to make the first step towards a fully reproducible real-world analysis of networked control and communication to foster the applicability of gain scheduled control to wireless control systems.

4 Design and Implementation

We create a run-time support system to enable gain scheduling in wireless networks of cyber-physical systems. As a basis, we use the openly available *Predictably Reliable Real-time Transport* (PRRT)[1] protocol [11], which provides partial reliability and in-order delivery, and at the same time allows making statements about the timing characteristics. The timing behavior is influenced by requirements an application has with respect to, for instance, the maximum tolerable latency. Thereby, one controller instance designed for a specific latency can communicate this requirement to the run-time system and the protocol.

Traditionally, only the control application—but not the network stack—is aware of latency requirements that are a constraint of the physical process it is designed to control. IP-based control applications can choose between two services: First, they can use a fully reliable transport protocol, such as TCP or QUIC, which retransmits messages even if the latency demands cannot be met any longer. Second, it can use an unreliable transport protocol, namely UDP, which does not retransmit at all, even if latency demands would allow it. PRRT allows an application to use a service that is *in between* these two extremes, providing *partial reliability* with *predictable timing*. Using the PRRT stack, the application passes its latency requirements to the network stack, which can handle retransmissions respecting latency requirements. If latency requirements of

[1] http://prrt.larn.systems

a message cannot be met any longer, PRRT discards it—thereby avoiding waste of time and energy.

Naturally, there are operating conditions that do not allow for a fulfillment of these constraints, e. g. meeting a 1 ms end-to-end deadline on a wireless link with 5 ms propagation time. In these cases, PRRT makes this issue transparent to the application, letting the application pick a remediation, e. g. triggering emergency routines or adapting its general control strategy. In the case of gain scheduling, this *notification* is the latest point in time to switch to a different controller instance that can handle the current conditions. A more efficient way to trigger the controller switch is to continuously probe the run-time system for a change in observed latencies or to register an event handler.

As long as the operating conditions allow for PRRT to fulfill its requirements, PRRT uses two techniques to do this in a reliable and predictable way: (a) error control and (b) a combination of congestion and rate control.

Error control is implemented as a block-based hybrid ARQ scheme, so PRRT aggregates multiple packets to a block. The packets themselves are sent out as fast as they arrive and proactive redundancy is sent as soon as a block is filled. Afterwards, reactive transmissions of redundancy are triggered if no acknowledgments arrive within a round-trip time plus processing margin. The arrival of a sufficient number of data or redundancy packets for a block allows reconstructing all packets of the block, e. g. previous sensor readings or actuator inputs. While the relevance of any sensor reading older than the latest is zero for Markovian controllers, our solution targets controllers where either (a) there is no Markovian model and the history is important (e. g. to detect temperature trends) or (b) the controller is fitting such a model during operation. Using error control, the protocol can optimize resilience under the given latency constraints.

Congestion and rate control minimize queuing that would lead to excessive delays by controlling both the amount of data in-flight as well as the rate of packets. This combined approach aims to avoid both self-induced as well as contention-based queueing delays, a well-known problem of loss-based TCP congestion control [7].

The controller-supplied latency constraint is further used as a deadline for messages, i. e., messages that have already exceeded the deadline or are going to exceed it with certainty are not processed further. Thereby, perturbations of the end-to-end latency that lead to a single packet not arriving in time do not impede subsequent packets that can still make the deadline. Additionally, the `recv()` calls have a *receive_window* parameter to filter packets that are ready to be delivered, namely those that expire in between *now* and *now* + *receive_window*.

This timing-awareness within PRRT enables our run-time system to select controller instances dynamically depending on the current operating conditions. The cooperation between the transport layer and the control application is hence symbiotic: the transport protocol provides timing measurements for controller selection while the control application dynamically reconfigures latency and, indirectly, error control parameters.

PRRT measures the network round-trip time using an algorithm similar to NTP by including timestamps in its metadata as well as feedback packets and compensating for processing time. Simultaneously, PRRT tracks the current data rate by estimating the delivery rate on the sender side, leveraging a mechanism presented in a recent IETF draft [4]. The implemented congestion control follows the design of BBR [2], including adaptations of recent fixes in the Linux kernel code for TCP-BBR. This congestion control, together with rate control through packet pacing, aims to avoid queueing at all stages of the communication, minimizing latency and jitter.

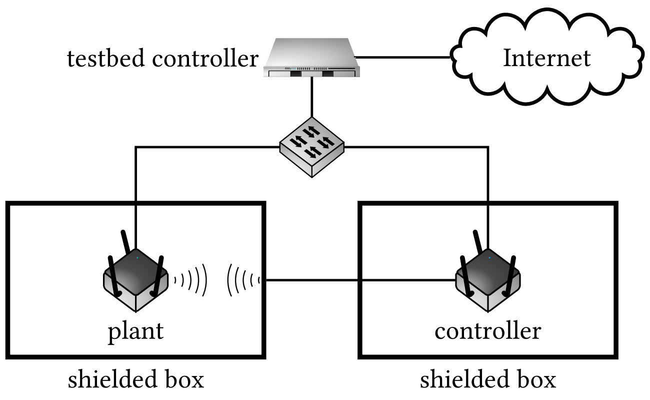

Figure 2. Simplified testbed setup

In summary, our run-time support system exploits PRRT, a partially reliable and latency-aware transport protocol, for gain scheduling. It enables control applications to adapt dynamically to the currently faced communication latencies. Simultaneously, the controller selection allows the transport protocol to minimize the error rate as well as jitter.

5 Testbed and Measurement Setup

Towards comparable benchmarks of our run-time system at different settings, a reproducible test environment is essential. Since wireless networks are affected by many different factors such as noise, networks on the same or neighboring channels, fading channel conditions, and radar detection, it is challenging to guarantee comparable conditions across different measurements. Note that we use this testbed setup to ensure reproducibility of our evaluation runs, but it is not required to operate our system.

To allow comparisons between benchmarks of PRRT at different settings, we use the setup depicted in Figure 2 consisting of two wireless test nodes (*plant* and *controller*) placed in shielded boxes. The antenna port of the controller is connected to a shielded coaxial cable that is connected to an antenna placed in the plant's shielding box. An air gap between that antenna and the antenna of the plant within the shielded box ensures constant channel conditions resembling an undisturbed wireless network allowing for repeatable wireless conditions across all measurements. We use IEEE 802.11g (54 Mbit/s) in ad-hoc mode and generate PRRT packets with a constant sampling interval. The test nodes use Debian Linux (linux-4.8) and are equipped with AMD GX-412TC CPUs (4 × 1 GHz "Jaguar" cores), Qualcomm Atheros AR958x IEEE 802.11abgn wireless network adapters, and Intel I210 NICs.

Both nodes are connected via Ethernet to a *testbed controller* which is connected to the Internet for remote testbed operation. The testbed uses the pos framework [6] to execute the network experiment. It orchestrates a set of measurements by performing the following steps:

1. After each measurement run, the test nodes are completely reset by power cycling and booting a live system via PXE

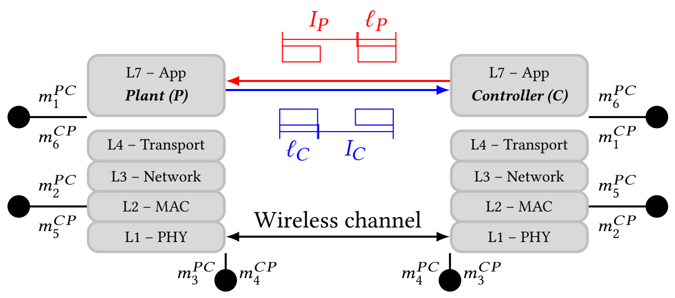

Figure 3. Stack with timestamping vantage points

from the testbed controller, eliminating any residual effects of the previous measurement run such as the firmware of wireless devices being initialized with unwanted settings.

2. When the nodes are booted, clocks are synchronized once using the Precision Time Protocol (PTP) utilizing the hardware support of the I210 NICs. This is crucial to obtain comparable timestamps on both nodes. Starting with a deviation below 1 µs after synchronization, the clocks' deviation does not exceed 10 µs after a single test run of 2 min.
3. When all preparations are finished, the test nodes are ready to execute the actual measurement software.

The software works with two independent threads: the first thread only transmits and receives packets, while the second thread captures packets using *libpcap*. This architecture ensures that packets are processed as soon as they are received to keep the timestamps as accurate as possible.

In order to evaluate PRRT, we integrated it into our measurement software, which allows recording timestamps at various locations throughout the protocol stack, as shown in Figure 3. Considering the direction from plant to controller (denoted as *PC*), we obtain the timestamps

- m_1^{PC} when the app transmits a message,
- m_2^{PC} when a frame becomes visible by libpcap at the transmitting node, i. e., before it is actually transmitted,
- m_3^{PC} when the *echo frame*[2] including the sender's radiotap header becomes visible at the transmitting node,
- m_4^{PC} when a frame is received,
- m_5^{PC} when the received frame becomes visible through libpcap at the receiving node, and
- m_6^{PC} when the measurement software receives a message.

The radiotap header thereby contains various information about how a frame has been transmitted, e. g. the chosen transmit rate. The same holds for the reverse direction (denoted as *CP*). Using these timestamps we can derive delays that are difficult to determine under ordinary circumstances. For instance, $t_d^{PC} = m_6^{PC} - m_1^{PC}$ is the one-way delay from plant to controller, which is all but symmetric for wireless networks. This one-way delay is of particular interest for the evaluation of PRRT as it allows to verify whether or not datagrams are within the defined receive window. Similarly, the delay $m_3^{PC} - m_2^{PC}$ is primarily influenced by the media access time, which can be determined precisely when the serialization time of frames is known.

[2]By *echo frame* we refer to the frame including the radiotap header provided by a wireless card's driver when a frame has been transmitted successfully.

Table 1. Parameters of the delay measurement set

Parameter	Minimum	Maximum	Steps
Packet-to-packet time (I)	1 ms	10 ms	5
Payload size (ℓ)	20 B	1400 B	5
PRRT target delay	1 ms	10 ms	5
PRRT receive window	0.1 ms	2 ms	6

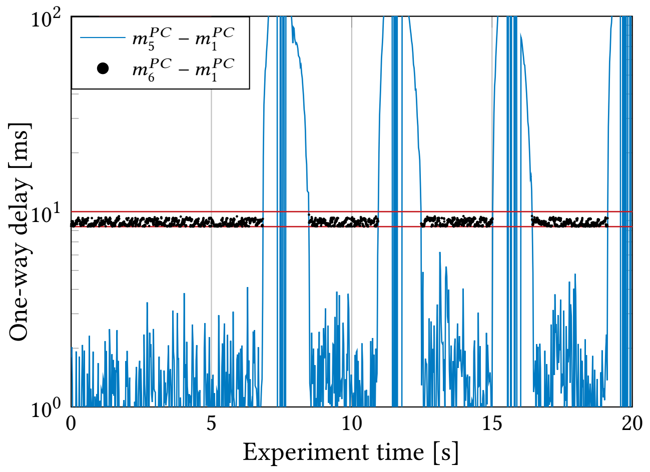

Figure 4. 1 ms packet-to-packet time, 20 B payload size, 10 ms target delay, 1.62 ms receive window

In case PRRT is not able to deliver a packet within the desired interval due to delays on the wireless channel or within the operating system, the packet is discarded. If a packet is discarded for that reason on the way from the plant to the controller, the timestamp m_6^{PC} is missing as PRRT dropped the respective packet due to a violation of the desired time interval. The same holds for m_6^{CP} when a packet in the reverse direction is dropped. The timestamps in between m_1 and m_6 are useful to investigate the channel's characteristics or influences of the OS independently from PRRT. They can aid in comprehending why PRRT, for instance, was not able to deliver specific packets in time. The delay $t_{proc\_s} = m_2^{PC} - m_1^{PC}$ gives an insight how long PRRT needs to process packets from the application and hand it over to the network stack. Correspondingly, the delay $t_{proc\_r} = m_6^{PC} - m_5^{PC}$ shows how long packets are delayed before being handed back to the application.

After a measurement run has finished, the files from the test nodes containing the timestamps are copied to the testbed controller and the parameters of the test run are logged.

6 Evaluation

We investigate the behavior of PRRT through a series of measurements using a combination of four configuration parameters: the packet-to-packet time (I) specifies the sampling time of a control process, the payload size (ℓ) sets the data transmitted by the control process, the PRRT target delay defines the time data should arrive at the control process, and the PRRT receive window defines a grace period (cf. Section 4). Table 1 contains the values for the measurement parameters. All possible combinations result in 750

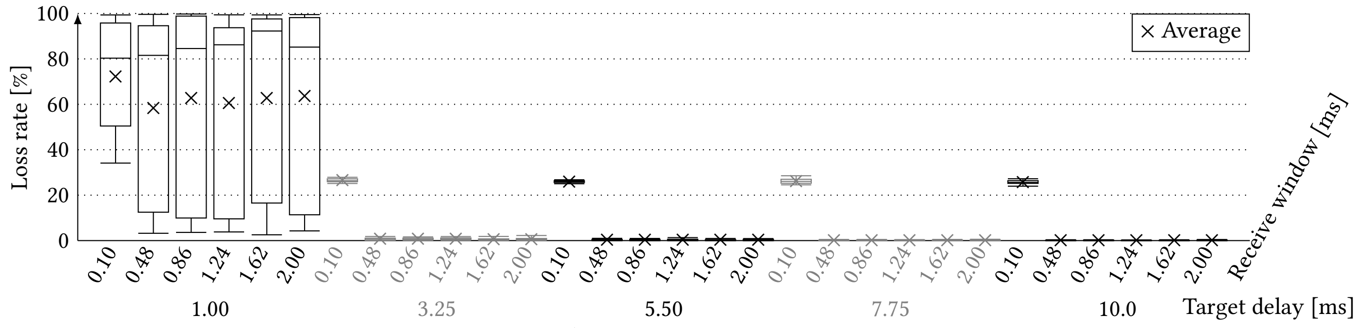

Figure 5. Measurements of PRRT with varying target delay and receive windows

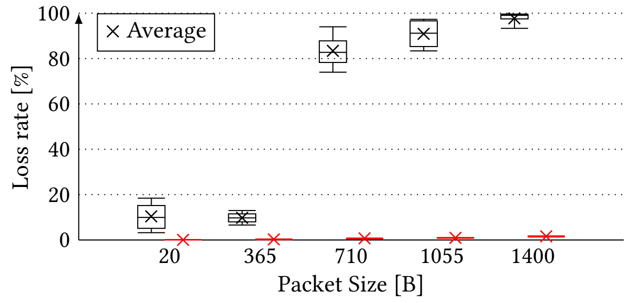

Figure 6. Measurement of PRRT for different packet sizes, receive window of 480 µs, target delay of 1000 µs (black) and 3250 µs (red)

distinct measurement runs. For the measurements, we only consider one-way delays as our environment creates almost symmetric channel conditions in both directions due to shielding both test nodes from interference with other networks.

Packet-to-packet time: Figure 4 shows a single measurement run with a sampling time of 1 ms, a payload size of 20 B, a receiving window of 1.62 ms, and a target delay of 10 ms as a time series over 20 s with different delays on a logarithmic ordinate. The delay from the sending application to the network stack of the receiving node ($m_5^{PC} - m_1^{PC}$) is depicted as the blue line and the delay to the receiving application ($m_6^{PC} - m_1^{PC}$) as the black scatterplot. The receive window is visible as upper and lower bounds (red horizontal lines). If the delay to the receiving network stack already violates the target delay, as seen in Figure 4 at approximately 7 s, the packet can obviously not arrive in time. In such a case PRRT will not deliver the packets to the application which manifests as gaps in the scatter plot.

A closer investigation of the delays between network stack and driver ($m_3^{PC} - m_2^{PC}$) reveals an increase during the periods of packet loss. Further investigations of the packet delay ($m_4^{PC} - m_3^{PC}$) show that the (lower-layer) transmission time between the NICs may exceed the investigated packet-to-packet time of 1 ms. This leads to a buffer overload scenario on the NICs as their buffers cannot be drained fast enough, which results in delay violations on the application layer. This disposition for overloading is a property we cannot solve within PRRT. However, PRRT can provide the means to detect such situations. Based on that information, gain scheduling can select a different controller instance that operates at a lower sampling frequency to allow stable control performance in such cases.

PRRT target delay and receiving window: Figure 5 shows the loss rate over a series of measurements with varying target delay and receive windows. Loss rates for a target delay of 1 ms have a median above 80 %, which can be attributed to the target delay being too close to the lower layer transmission delay. A narrow receive window also has an influence on the loss rate: for 0.1 ms, the loss rate reaches 25 % independent of the target delay. For wider receive windows and target delays above 3.25 ms, the loss rate stays below 3 %. These results indicate that the achievable target delay has the highest impact on the loss rate. However, even if high target delays are combined with narrow receive windows, packet loss may still remain high. If the packet loss rates are unacceptable for a controller instance, gain scheduling can switch to another controller instance respecting both target delay and receive window sizes.

Payload size: To investigate the influence of the payload size on the loss rate, we pick two examples from Figure 5 for a closer investigation. Figure 6 shows an example for a receive window size of 0.48 ms and a target delay of 1 ms in black. Only for low payloads of 20 B and 365 B a loss rate of below 20 % can be achieved. For higher payloads, the loss rate steeply rises to over 80 %. The second example in Figure 6 shows the scenario with a receive window of 0.48 ms and a target delay of 3.25 ms in red. There, the packet size has only little influence on the loss rate, rising up to 1.75% in the worst scenario using 1400 B packets. As the packet size has only little influence on the loss rate—compared to the previous parameters—gain scheduling should consider packet size as a minor input factor.

7 Conclusion

Migrating control applications into edge computing systems allows for the cooperation of cyber-physical systems. This paper considers these edge controllers utilizing wireless communication links for mobile cyber-physical systems, introducing unpredictability with respect to latency, jitter, and error rate into the control process.

Our novel approach combines two techniques for counteracting the negative influence of the wireless links—gain scheduling and a link monitoring system. Our system monitors the channel properties at run-time, utilizing the PRRT protocol and cooperates with the control application to select the best available controller instance, i. e., gain scheduling. Measurements investigate four main levers—sampling frequency, target delay, receiving window, and packet size—influencing the loss rates on wireless connections and identifying potential bottlenecks. Our fully automated evaluation procedure uses shielded boxes to create reproducible results in a well-defined environment for wireless measurements. We identify sampling frequency and target delay as the most important impact factors on loss rate as long as receive windows are not chosen too narrow. Packet size only has limited influence on loss rates.

This work has demonstrated the applicability of our run-time system to control problems. As future work, we plan to evaluate the impact of the run-time system and the automated controller selection on the *quality of control* of realistic control applications. Particular attention will be put on the dynamics caused by switching the controller instance and how this switching can be synchronized to the channel dynamics. In this work, we conducted our measurements only under optimal conditions for the wireless network. Future measurements will involve a more realistic environment investgating the effect of interferences, channel occupation, and other effects that lead to varying latencies. Furthermore, we plan to examine the impact of our approach on *resource-constrained* edge controllers, where only a limited set of controller instances can be allocated to a single node at the same time and additional controller instances are downloaded on-demand from a large-scale backend system, e. g. a cloud.

Acknowledgments

The German Research Foundation (DFG) supported this work within the priority programme 1914 *Cyber-Physical Networking (CPN)* under grants CA 595/71, HE 2584/4-1, SCHR 603/15-1, and the REFLEXES project.

References

[1] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. 2012. Fog Computing and Its Role in the Internet of Things. In *MCC 2012*. ACM, 13–15.

[2] N. Cardwell, Y. Cheng, C. Gunn, S. Hassas Yeganeh, and V. Jacobson. 2016. BBR: Congestion-based congestion control. *ACM Queue* 14, 5 (2016), 50.

[3] Y. Chen, Q. Feng, and W. Shi. 2018. An Industrial Robot System Based on Edge Computing: An Early Experience. In *HotEdge 18*. USENIX Association.

[4] Y. Cheng, N. Cardwell, S. Hassas Yeganeh, and V. Jacobson. 2017. Delivery Rate Estimation - IETF DRAFT.

[5] R. Costa, P. Portugal, F. Vasques, C. Montez, and R. Moraes. 2015. Limitations of the IEEE 802.11 DCF, PCF, EDCA and HCCA to handle real-time traffic. In *INDIN 2015*.

[6] S. Gallenmüller, D. Scholz, F. Wohlfart, Q. Scheitle, P. Emmerich, and G. Carle. 2018. High-Performance Packet Processing and Measurements (Invited Paper). In *COMSNETS 2018*. Bangalore, India.

[7] J. Gettys and K. Nichols. 2011. Bufferbloat: Dark buffers in the Internet. *ACM Queue* 9, 11 (Nov. 2011), 40:40–40:54.

[8] D. Leith and W. Leithead. 2000. Survey of gain-scheduling analysis and design. *Internat. J. Control* 73, 11 (2000), 1001–1025.

[9] K. Nakashima, T. Matsuda, M. Nagahara, and T. Takine. 2017. Cross-layer design of an LQG controller in multihop TDMA-based wireless networked control systems. In *PIMRC 2017*. 1–7.

[10] G. Nikolakopoulos, A. Panousopoulou, A. Tzes, and J. Lygeros. 2005. Multi-hopping Induced Gain Scheduling for Wireless Networked Controlled Systems. In *CDC 2005*. 470–475.

[11] S. Reif, A. Schmidt, T. Hönig, T. Herfet, and W. Schröder-Preikschat. 2018. Δelta: Differential Energy-Efficiency, Latency, and Timing Analysis for Real-Time Networks. In *ECRTS RTN 2018 (RTN)*. ACM SIGBED, Barcelona, Spain, 6.

[12] W. Rugh and J. Shamma. 2000. Research on gain scheduling. *Automatica* 36, 10 (2000), 1401 – 1425.

[13] M. Satyanarayanan. 2017. The Emergence of Edge Computing. *IEEE Computer* 50, 1 (Jan. 2017), 30–39.

[14] A. Schimmel and A. Zoitl. 2010. Real-Time Communication for IEC 61499 in Switched Ethernet Networks. In *ICUMT 2010*. IEEE, 406–411.

[15] J. Shamma and M. Athans. 1991. Guaranteed properties of gain scheduled control for linear parameter-varying plants. *Automatica* 27, 3 (1991), 559 – 564.

[16] W. Shi and S. Dustdar. 2016. The Promise of Edge Computing. *IEEE Computer* 49, 5 (May 2016), 78–81.

[17] S. Srinivasan, M. Vallabhan, S. Ramaswamy, and Ü. Kotta. 2013. Adaptive LQR controller for Networked Control Systems subjected to random communication delays. In *ACC 2013*. 783–787.

[18] G. Tian and Y. Tian. 2010. Markov Modelling of the IEEE 802.11 DCF for Real-Time Applications with Periodic Traffic. In *HPCC 2010*.

[19] A. Tzes and G. Nikolakopoulos. 2004. LQR-output feedback gain scheduling of mobile networked controlled systems. In *ACC 2004*, Vol. 5. 4325–4329 vol.5.

[20] F. Xia, L. Ma, C. Peng, Y. Sun, and J. Dong. 2008. Cross-Layer Adaptive Feedback Scheduling of Wireless Control Systems. *Sensors* 8, 7 (Jul 2008), 4265–4281.