IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 5 September 2022, accepted 29 October 2022, date of publication 1 November 2022, date of current version 8 November 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3218774

RESEARCH ARTICLE

DynNetSLAM: Dynamic Visual SLAM Network Offloading

PETER SOSSALLA[1], JOHANNES HOFER[1], JUSTUS RISCHKE[1,2], CHRISTIAN VIELHAUS[1], GIANG T. NGUYEN[1,3], MARTIN REISSLEIN[4], (Fellow, IEEE), AND FRANK H. P. FITZEK[1,3], (Senior Member, IEEE)

[1]Deutsche Telekom Chair, 5G Laboratory Germany, Technische Universität Dresden, 01062 Dresden, Germany
[2]Dr. Ing. h.c. F. Porsche AG, 70435 Stuttgart, Germany
[3]Centre for Tactile Internet with Human-in-the-Loop (CeTI), Technische Universität Dresden, 01062 Dresden, Germany
[4]School of Electrical, Computer, and Energy Engineering, Arizona State University, Tempe, AZ 85287, USA

Corresponding author: Martin Reisslein (reisslein@asu.edu)

This work was supported in part by the Audi AG, German Federal Ministry of Education and Research (BMBF), through the Project 5G Insel under Grant 16KIS0956K; in part by the Software Campus R-SLAM under Grant 01IS17044; in part by the German Research Foundation (DFG) through Germany's Excellence Strategy EXC 2050/1 – Project ID390696704 – Cluster of Excellence Centre for Tactile Internet with Human-in-the-Loop (CeTI) of Technical University of Dresden.

ABSTRACT Existing Visual Simultaneous Localization And Mapping (vSLAM) approaches that offload the complex self-localization computations from mobile robots over a wireless network to edge computing are limited to static offloading, i.e., the offloaded computation tasks are offloaded permanently. However, wireless networks are inherently dynamic and may excessively delay the transmissions between a mobile device and the edge during periods of poor wireless network quality, e.g., from fading or temporary obstructions. We propose and evaluate Dynamic Visual SLAM Network Offloading (DynNetSLAM) to dynamically adapt the vSLAM computation offloading according to the measured wireless network latency. As groundwork towards developing DynNetSLAM, we first enhance the existing state-of-the-art vSLAM approaches through judicious parameter settings and parallel map updates to enable the tracking of common fast vSLAM data sets. We introduce an offloading latency threshold along with a safe zone and a hysteresis around the threshold to control the dynamic offloading. Our extensive evaluations with public vSLAM data sets indicate that DynNetSLAM with the hysteresis substantially reduces the probability of track loss events compared to the state-of-the-art ORB-SLAM2 approach for processing statically on the mobile device and the enhanced static Edge SLAM. Also, DynNetSLAM nearly attains the low absolute position error and only slightly increases the CPU utilization compared to the enhanced static Edge SLAM.

INDEX TERMS Accuracy, edge computing, latency, offloading, reliability, visual simultaneous localization and mapping (vSLAM), wireless network.

I. INTRODUCTION

Reliable and flexible localization is essential for autonomous systems, such as robots and self-driving vehicles [2], [3], [4], [5], [6]. An increasingly popular localization method is Simultaneous Localization and Mapping (SLAM) [7]. SLAM extracts properties of the environment from different data sources and compares with previous recordings to perform localization. Visual Simultaneous Localization And Mapping (vSLAM) [8], [9], [10], [11] uses images from an optical camera. The visual image data can also be acquired together with depth data, e.g., with RGB-D (red, green, and blue color with depth sensing) cameras [12], [13].

The vSLAM computations are complex and the need for real-time localization leads to high hardware requirements. In self-driving vehicles, as well as in mobile robots, powerful processors and graphics cards are used to provide the necessary computing power. This extra hardware and the resulting added weight on the mobile device increase the energy consumption. Since mobile systems are often powered by

The associate editor coordinating the review of this manuscript and approving it for publication was Christos Anagnostopoulos.

batteries, higher energy consumption leads to a shorter range.

A. MOTIVATION FOR SLAM NETWORK OFFLOADING

In recent years, the offloading of computationally intensive processes from mobile devices to cloud systems has become an important research topic [14], [15]. Particularly the trend of edge cloud computing [16], [17], [18], [19], [20], [21] to perform computation as close as possible to the end user, i.e., at the edge of the network, has enabled the offloading of latency-critical applications, such as SLAM. With the upcoming usage of private 5G networks [22], virtualized robot control [23] with edge computing has gained traction. The study [24] demonstrated that the increased computing power supplied by edge computing speeds up the SLAM processing and reduces the computing load on the mobile end device. Recent studies [25], [26], [27] have demonstrated that components of the SLAM method can be reliably offloaded to the network. However, little reference is made to a main characteristic of wireless networks: a fluctuating quality of the communication channel. This fluctuation affects the throughput and latency of the network connection between the mobile device and the computing resource. Increasing latency has an impact on the responsiveness of the system. A high network latency can make offloading unreliable and even negate the offloading benefits such that an offloading system with high network latency performs worse than a system without offloading.

B. CONTRIBUTIONS AND STRUCTURE

Since the conditions in wireless networks change over time, it is reasonable to design the offloading dynamically. Therefore, we introduce Dynamic Visual SLAM Network Offloading (DynNetSLAM) to dynamically offload SLAM components depending on the network latency. For this purpose, the current wireless network latency is monitored and according to the measured network latency, compute-intensive SLAM processes are offloaded to the edge or executed on the mobile device.

We first provide background on vSLAM for generalist readers and review the existing static vSLAM edge offloading approaches in Section II. In Section III, we model the delays of the static Edge SLAM [25] offloading and define the SLAM performance metrics. As groundwork towards the development of DynNetSLAM, we then enhance the existing state-of-the-art static Edge SLAM system by judiciously setting key parameters and by parallelizing the map update processing in Section III-B. Also, we examine the impact of the mobile device-edge Round Trip Time (RTT) latency on the reliability of the enhanced static Edge SLAM in Section III-C.

Based on this groundwork, we present in Section IV the novel DynNetSLAM system that dynamically offloads SLAM computations to the edge depending on the current communication latency. The RTT latency of the connection is measured in real time and used to decide whether the computation should be performed on the mobile device or on the edge. DynNetSLAM can operate with different offloading strategies. Specifically, we evaluate four different offloading strategies that operate with or without a safe zone or hysteresis around an offloading latency threshold.

The evaluations in Section V employ two different sequences of the publicly available data set [28], which is commonly used for vSLAM benchmarking. We evaluated the reliability of the localization, i.e., the probability of track loss events that bring the localization to a halt, the localization accuracy, as well as the CPU usage on the mobile device. Results demonstrate that dynamic offloading can overcome the reliability loss of Edge SLAM in the case of fluctuating wireless network latency. This allows DynNetSLAM to provide higher reliability than running solely on the mobile device or constantly offloading the compute-intensive functions to edge computing. We then summarize the results and outline future work directions in Section VI.

II. BACKGROUND & RELATED WORK

A. BACKGROUND ON SLAM

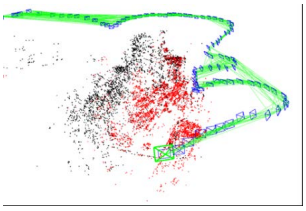
The goal of Simultaneous Localization and Mapping (SLAM) [7], [29] is the autonomous localization of a mobile robot in space. SLAM is especially interesting for indoor environments [6], where satellite-based radio navigation, such as Global Positioning System (GPS) [30], do not work due to the blocked signals from the satellites. The robot uses properties of the environment, i.e., corners or edges, to orientate itself and to simultaneously create a map. This allows the robot to estimate its position even without prior information about its environment.

To capture the properties of the environment, called *features*, various sensors can be used. Typical examples for sensors are Light detection and ranging (Lidar), monocular cameras, or RGB-D camera. Depending on the sensor type, a distinction can be made between Laser SLAM [31] and vSLAM [10]. While Laser SLAM is based on planar data from a laser range finder, vSLAM uses data from optical cameras. In vSLAM, these can also be refined with depth images [32], which leads to highly accurate 3D maps. Due to the higher number of sensor data types, more information can inherently be captured by 3D visual cameras than by 2D laser scanners. In addition, 3D visual cameras, such as Stereo and RGB-D cameras, have become more affordable in recent years and are therefore increasingly used in mobile robots [13]. The additional information and the increased usage of 3D cameras motivates us to focus on vSLAM.

1) VISUAL SLAM (vSLAM)

When a color or depth image arrives at the vSLAM system, features are searched for in the image. A *frame*, which describes the detected features in the image, is shown at the left in Figure 1a. The size and position of the features are compared with previous frames. Based on the detected changes, the SLAM system then attempts to estimate a change in location and orientation referred to as a pose. The process

(a) RGB-D input stream with detected features matches.



(b) Created map points (dots), keyframe poses (blue rectangles), current keyframe map points (red dots) and covisibility graph (green lines).

FIGURE 1. (a) Frame with feature matches and (b) created map of sequence `freiburg3_long_office_household_2` of dataset [28].

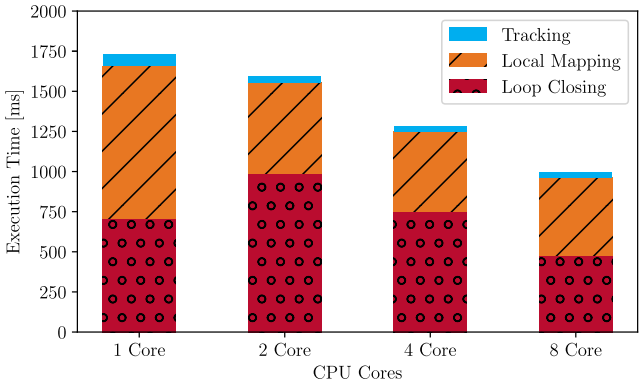


FIGURE 2. Average computing delay of the three ORB-SLAM 2 modules as a function of the number of CPU cores. The total computing delay was reduced by 48 % by increasing the number of cores from one to eight.

of detecting the pose from corresponding frames is referred to as *visual odometry* [33]. The process of feature extraction and matching as well as the visual odometry is referred to as *Tracking*.

Using the estimated position change, the frame with features is then used to insert new map points into the 3D map, referred to as *Mapping*. The created map can be represented in different variants [34]. A typical variant is the storage in a graph [35], [36], where the pose is stored together with the identified features. Figure 1b shows a map with all created map points and frame poses. In addition, a covisibility graph is created that contains information about which frames observe similar map points. The map is then used in the further process to estimate the position. Clearly, errors can already occur with the estimation of the position change. During the subsequent SLAM processing, the error can propagate and lead to a drift. One possibility to prevent this drift is the so-called *Loop Closing*. Loop closing checks whether one has already visited the respective place. If so, the map previously created as a graph is searched again and subsequently improved.

Over the last few years, many systems have been proposed for vSLAM [10]. The graph-based ORB-SLAM [37] and RTAB-Map [38] vSLAM approaches are widely accepted. We have adopted ORB-SLAM for this study, because it is widely used and has high accuracy and stability [39].

2) ORB-SLAM

ORB-SLAM [37] can operate in real-time, in outdoor and indoor environments and originally used monocular cameras as image sources. ORB-SLAM2 [40] was introduced with the support of stereo and RGB-D images. Our proposed system for offloading SLAM is based on ORB-SLAM2. Therefore, we briefly describe the relevant ORB-SLAM2 modules: *Tracking*, *Local Mapping (LM)*, and *Loop Closing (LC)*.

- **Tracking:** In the Tracking Module, features are extracted and then matched to perform an initial pose estimation [41]. For feature extraction, the eponymous Oriented FAST and rotated BRIEF (ORB) [42] is used, which is based on the keypoint detector Features from Accelerated Segment Test (FAST) and the visual descriptor Binary Robust Independent Elementary Features (BRIEF). After the initial pose estimation, a decision is made whether to declare the current frame a so-called *keyframe*. Deciding whether a keyframe contains enough new information to be considered suitable for insertion into the map is based on several criteria [40]. If a new keyframe is detected, it will be forwarded to the local mapping module.
- **Local Mapping (LM):** In the LM module, the keyframe and its corresponding estimated pose are saved in graph-based structures. The found features within the keyframe are inserted as *map points* with their corresponding estimated coordinates. After additional optimization, unnecessary keyframes and map points are detected and deleted.
- **Loop Closing (LC):** In parallel to the LM, the LC module is executed. Its function can be divided into two parts: Loop Detection and Loop Correction. The loop detection checks for each keyframe whether the mobile device has already visited this location. Loop correction includes the optimization of already existing map points.

B. OFFLOADING SLAM COMPUTING

The processing of SLAM is computationally and memory intensive [24], especially LM and LC. To further illustrate the influence of the computing power on the SLAM processing, we plot the average measured processing times of the ORB-SLAM2 modules as a function of the number of available CPU cores in a Virtual Machine (VM) in Figure 2. We immediately observe that the total processing time for all the modules together decreases substantially with an increasing number of CPU cores. This can be attributed to the implementation of the modules in multiple threads, whereby a parallel execution of some processes in several CPU cores can be exploited effectively. Since the different modules partly access shared data structures, they can block each other. By reducing the execution times, the shared data structures are blocked for a shorter period of time, further reducing the execution time. Semenova et al. [43] present a quantitative

analysis of the system challenges of ORB-SLAM2. They analysed the calculation times of the different ORB-SLAM2 modules on two computers with different processing powers. Semenova et al. [44] motivate the modular implementation of Edge SLAM. They demonstrated that a lower computing power leads to a lower keyframe creation rate. A low keyframe rate has a negative impact on the reliability of the vSLAM system, since, for example, not enough matches can be established between the frames during rotational movements.

This motivates us to offload the processing to a computationally more powerful instance. Additionally, we observe from Figure 2 that the tracking module has the lowest computation delay in comparison to the LM and LC modules. Hence, we infer that offloading the LM and LC modules can substantially reduce the computing times. Also, the transmission of raw camera data prior to the execution of the tracking module would require a large bandwidth in the communication channel [45], while the transmission of detected ORB features requires less bandwidth.

C. RELATED WORK ON CLOUD AND EDGE ROBOTICS

Moving tasks from mobile robots to cloud systems is a widespread topic in research and industry [14], [15], [46], [47], [48]. A disadvantage of the computation of conventional cloud systems is the network latency between the cloud and the robot. This latency encompasses the network interface or switch processing, queuing, transmission, and propagation latencies for the wireless connection as well as for the wired connection to the cloud computing center. If centralized cloud computing services are used, the latency depends on the distance and forwarding in the public network. Therefore, it would be desirable to at least reduce the latency between the wireless network and the control system.

Edge Computing [20] is based on the concept of bringing the resources as close as possible to the user, i.e., to the edge of the network. This can reduce the latency caused by passing through corporate or public networks. In addition, fluctuations in latency outside the user's own control can be mitigated. This makes the use of edge computing interesting for robotics or autonomous driving and thus attracts the interest of academia [49], [50], [51].

Most studies on offloading tasks of mobile robots relate to object recognition and navigation functions. For object recognition, Fan et al. [52] exploit edge computing for the collaboration between a robot arm and a mobile robot, whereby the calculation necessary for navigation is performed on the edge. Fan et al. demonstrate that static offloading can reduce the computation time for the object recognition. Dynamic offloading for object tracking was studied in [53]. The offloading decision depends on the current computing load produced by the corresponding number of features that have to be tracked. In [53], the tracking of objects was offloaded. The computation is offloaded when the workload on the device increases to a level in which the quality of the tracking drops. Spatharakis et al. [54] present offloading strategies for navigation functions of mobile robots that aim to maximize the utilization of the computing resources in the edge.

Another advantage of using edge computing for mobile robots is the use of a global map for multiple robots, which improves navigation. This global map can be collaboratively created and updated by multiple robots [55]. The global map can then be shared and updated depending on the current task and environment of the individual robots. Especially for 3D maps, which are characterized by large memory and computation requirements, a sophisticated management of the maps is useful, as proposed in EdgeSharing [56] by Liu et al.

The use of edge computing was also extended for drones [57], [58], which can be considered as flying robots. Since drones usually have very limited resources on the device, due to strict constraints on weight and power supply, they benefit greatly by offloading tasks. Hayat et al. [57] demonstrate the advantages of offloading feature detection and tracking for trajectory planning for drones. They compare operation with full, partial, or no offloading. For their setup, full offloading achieves the shortest processing times under the assumption that the necessary network bandwidth is available. Ahmad et al. [58] demonstrate a system to reconstruct 3D models based on Lidar on the edge. The Lidar data is first compressed and then uploaded for processing. The study [58] highlights the potential of offloading computationally intensive processes from computationally weak devices to a more powerful edge cloud. The majority of the study [58] is focused on tracking, i.e., feature detection and extraction. The outsourcing of further SLAM elements has been investigated in the studies reviewed in the following subsection.

D. RELATED WORK ON OFFLOADING SLAM

We have illustrated in Figure 2 that increased computing resources greatly accelerate the processing of the computationally intensive vSLAM tasks. This demonstrates the potential benefits of outsourcing SLAM to an edge cloud. Huang et al. [59], [60] introduce an algorithm to offload 2D Laser SLAM, whereby the GMapping 2D SLAM system [61], [62] is utilized. They demonstrate that they can speed up processing and create joint maps on the edge by using the data of different robots. Kamburugamuve et al. [63] also build on GMapping to speed up SLAM by parallelizing the processing on multiple machines. Fukui et al. [64] offload GMapping of multiple robots to a central cloud system. Sarker et al. [65] present a layer-based architecture to offload 2D laser SLAM to a cloud or edge cloud. They demonstrate a working prototype and compare the energy consumption of two offload setups. Dey et al. [66] describe an offloading framework to optimize the energy consumption and processing time for feature matching for 2D Laser SLAM. Gouveia et al. [67] introduce the idea to offload the SLAM computing workload to multiple robots. Gouveia et al. show that sharing the data with more robots or computing the data by multiple robots in a more powerful central instance can reduce the

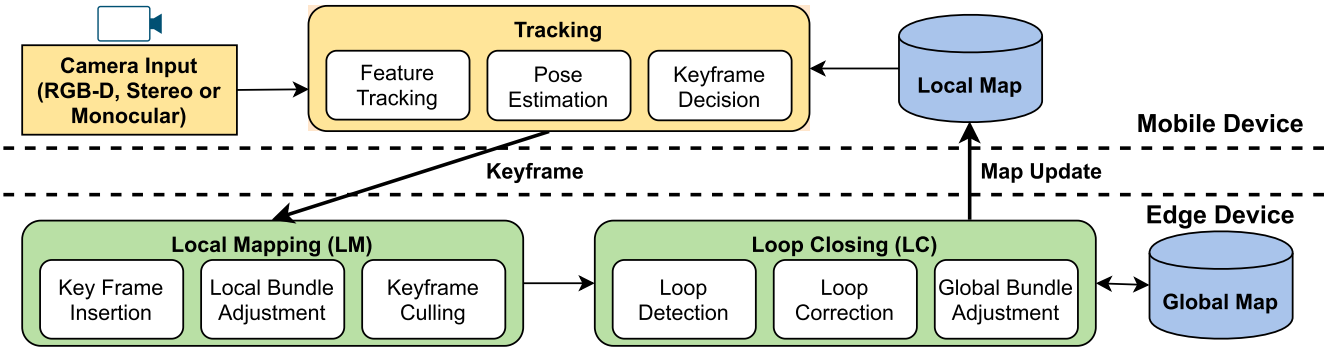


FIGURE 3. Simplified system structure of Edge SLAM with important modules, adapted from [25].

computing times. Algorithms have been proposed to manage the different Quality of Service (QoS) requirements when offloading 2D SLAM in a wireless network, see e.g. [68] and [69]. Another interesting application is the offloading approach for an acoustic SLAM with reflectors [70].

The C^2TAM study [71] provides an experimental setup to offload RGB-D SLAM based on the PTAM [72] map management framework into the cloud, whereby the focus is on consumed bandwidth and mapping frame rate. Benavidez et al. [73] propose saving features used by vSLAM in a cloud database. For this purpose, a cloud framework is presented and the feasibility is demonstrated with a proof-of-concept.

Ben Ali et al. introduced Edge SLAM [25], wherein the LM and LC modules from ORB-SLAM2 are offloaded to the edge. They demonstrated that Edge SLAM can offload vSLAM without loosing accuracy in comparison to the original ORB-SLAM2 on a mobile device. In their work, they also show the potential to reduce the memory and CPU consumption on the mobile device. Wright et al. [27] present an implementation of computing SLAM on the edge in an automotive context. They offload the loop closure and leave the tracking as well as the mapping on the device. Xu et al. [26] introduce edgeSLAM in which the LM and LC modules are offloaded, similar to [25]. The edgeSLAM study [26] focuses on the detection of temporal objects, such as pedestrians, in images. Here an offloading strategy is proposed, which defines an upper and lower limit of frames from when or until when a keyframe should be selected. These limits are defined by five different states. The states are selected based on the network latency and the bandwidth. Specifically, the computationally intensive modules are not dynamically offloaded, but the decision thresholds for the keyframes are dynamically adjusted. Xu et al. [74] also present a framework for collaborative vSLAM, in order to reduce the resource overhead and scheduling issues between multiple agents and the edge. To synchronize the maps of the different entities, each mobile device tracks system operations that modify, delete, or add map data. These system operations are transmitted to the edge, which applies all collected system operations to its own global map and additionally compresses them.

Cui et al. [75] propose a system to create semantic maps based on ORB-SLAM2. The system is based on edge computing and YOLOv3 feature extraction [76]. Tang et al. [77] introduce a coordinator to schedule the task offloading between a vehicle and an edge cloud. They use their proprietary SLAM system to offload the feature extraction and matching and speech recognition based on Deep Learning (DL). The optimization goal is to minimize the power consumption. For evaluation, the utilization on the mobile device and the edge cloud for the tasks was measured. Spatharakis *et. al* [78] present offloading strategies for localization and path planning of mobile robots. The offloading decisions depend on the expected pose estimation accuracy, path planning difficulty, and the available edge resources.

A variety of approaches for offloading SLAM have been proposed. Advantages in processing speed have been shown, but little emphasis has been placed on comparing reliability. Most of the existing SLAM offloading studies focus on offloading 2D laser SLAM. Offloading studies on vSLAM demonstrate the advantages of offloading to computationally more powerful systems. We consider [25], [26] to be the candidates with the greatest potential, since an offloading without the mapping module [27] would leave a large computing load on the mobile device. In [25] and [26], the impact of latency on reliability was not examined. However, results from [27] indicate a strong negative effect of high networking latencies on the SLAM reliability and accuracy. Solutions introduce systems to dynamically offload the computation depending on available bandwidth [54], or available resources on the edge [78], or mobile devices [67]. An offloading depending on the network latency, a parameter that directly influences the quality of the SLAM process, has not yet been presented. Therefore, in this work we present our system to dynamically offload depending on the network latency.

III. ENHANCING STATIC EDGE SLAM

A. MODELLING STATIC EDGE SLAM

Before we introduce dynamic offloading depending on latency, we model the static offloading of Edge SLAM, introduce enhancements, and evaluate the network latency impact. The main idea of Edge SLAM is to offload the computationally intensive tasks of SLAM to a resource which has

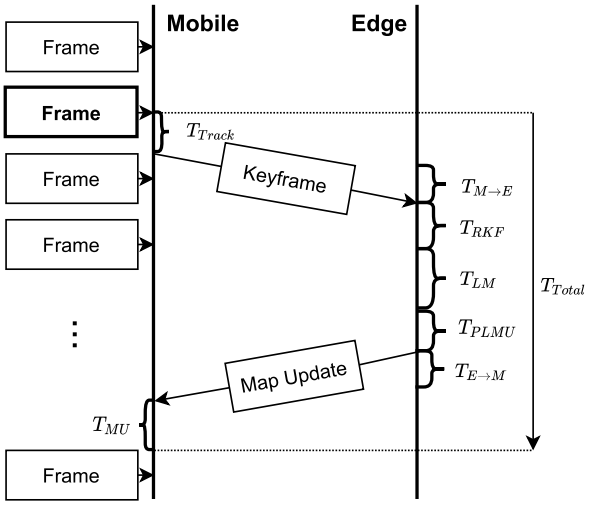


FIGURE 4. Workflow of Edge SLAM with corresponding delays.

higher computing power. These task are typically the LM and LC modules since they require high computing and memory resources. Figure 3 shows a schematic system overview of Edge SLAM with the three modules (tracking, LM and LC) and relevant processes running in these modules.

1) MODEL OF PROCESSING DELAYS

The tracking, LM, and LC modules use complex algorithms that add a computing delay to the overall process. Figure 4 illustrates the interactions between the mobile device and the edge. Delay T_{Total} denotes the time interval between the capturing of a frame on the mobile device and the time when the local map update is processed. The total delay

$$T_{\text{Total}} = T_{\text{Mobile}} + T_{\text{Edge}} + T_{\text{NW}}$$

is composed of computing on the edge T_{Edge} and mobile device T_{Mobile} as well as network delays T_{NW} between the two entities. Whereby T_{NW} encompasses the delays incurred for processing (in the network cards and switches), queuing, transmission, and propagation, i.e., for the entire two-way (round-trip) network transport.

The computing delay

$$T_{\text{Mobile}} = T_{\text{Track}} + T_{\text{MU}}$$

consists of the tracking delay T_{Track} and the integration of the local map update in the local map T_{MU}. Upon capturing a frame, the feature tracking module is called. T_{Track} encompasses the time for detecting the features in the frame, creating a first pose estimate, deciding if the frame has enough information to be declared a keyframe, and if so, serializing the data structures for further routing to the edge. After processing in the edge, the mobile device receives back a part of the global map, which is then integrated into its local map to complete a so-called local map update. The delay T_{MU} is caused by the integration of the local map update.

Receiving the keyframe T_{RKF}, the local mapping T_{LM} and the preparation of the local map update T_{PLMU} add up to the

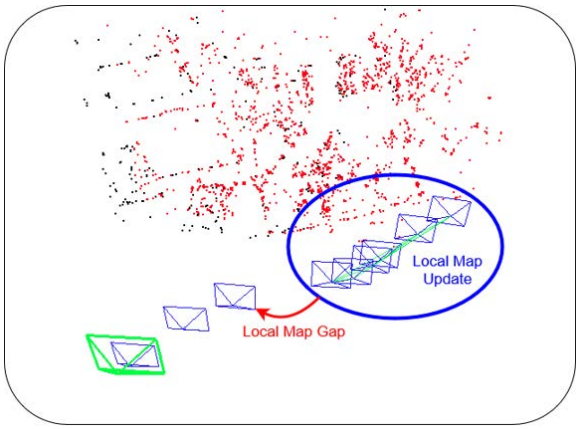


FIGURE 5. Illustration of local map gap.

edge processing delay

$$T_{\text{Edge}} = T_{\text{RKF}} + T_{\text{LM}} + T_{\text{PLMU}}.$$

When receiving a keyframe, first the data needs to be deserialized. The extracted keyframe is evaluated according to conditions outlined in [25] in order to determine whether the keyframe gives enough new information and should therefore be inserted into the global map. Furthermore, it is checked if the LM module is currently busy and these steps take time T_{RKF}. Afterwards, the LM process starts, which requires time T_{LM}. T_{LM} includes running Bundle Adjustment (BA) and keyframe culling, which are described in detail in [37]. After the LM process, the update is prepared for the mobile device, which causes the delay T_{PLMU}.

The exchange of data between the edge and the mobile device adds the round-trip network delay T_{NW}

$$T_{\text{NW}} = T_{\text{M}\to\text{E}} + T_{\text{E}\to\text{M}}.$$

$T_{\text{M}\to\text{E}}$ denotes the delay for sending the keyframes from the mobile device to the edge. $T_{\text{E}\to\text{M}}$ represents the delay for the transmission of the local map update from the edge to the mobile device. For a wired connection, the network delay mainly depends on the propagation delay, which increases with distance, and the delay added by the switches on the transmission path. Queuing delay may also be incurred, for example in case of congestion of a connection. However, queueing rarely occurs for over-provisioned bandwidth. In wireless networks, however, packet losses due to low signal quality can trigger retransmissions which in turn can cause increasing or fluctuating latency.

2) KEY PERFORMANCE METRICS

a: RELIABILITY

For mobile systems, a reliable localization is essential. In SLAM systems, however, localization can be interrupted. An interruption occurs if the mobile system can no longer estimate its position. Especially with regard to autonomous driving or in a factory environment with employees, an interruption can have fatal consequences. Therefore, the occurrence of the so-called *track loss* event [79] should be

(a) 23 s (b) 24 s (c) 25 s

FIGURE 6. Feature matches in a turn from 23 s to 25 s of sequence `office` of dataset [28].

prevented. One reason for a track loss is the processing of the frames with a low rate by the SLAM system. During fast movements or in a curve, it is possible that not enough common features are found in two successive keyframes to estimate the position change. Particularly rotational movements, as illustrated in Figure 6, can quickly lead to situations where not enough feature matches are found. ORB-SLAM2 has a built-in relocalization function, which is based on the assumption that, similar to loop closing, a location has been visited before. Since in mapping usually the same locations are not passed through more than once, no relocalization can be performed in mapping scenarios [79]. Even if a previously recorded area is visited again and a relocalization can be performed, the map is not updated between the track loss event and the relocalization.

Figure 5 illustrates a so-called Local Map Gap. Such a Local Map Gap arises on the mobile device when accepting a local map update. As described in Section III-A1, T_{Total} describes the time between the last keyframe sent to the edge and the acceptance of the corresponding local map update on the mobile device, see Figure 4. The keyframes that are used in the tracking module on the mobile device within T_{Total}, are not taken into account for the new local map, since receiving a local map update will delete the existing local map of the mobile device. For this reason, after a local map update, there can be a significant gap between the newest information within the local map and the new incoming frames. This can reduce the number of data points that are available for tracking new frames. If too much relevant map information is lost, track losses can occur. The low frame processing rate due to low computational power on the mobile device motivates the offloading to computationally powerful edge systems. The negative impact of network latency, on the other hand, motivates the dynamic offloading depending on network latency.

We measure the occurrence of track losses empirically by repeating sequence runs 100 times with the same initial conditions. We present the occurrence of track losses in two ways. The first way is to measure the cumulative number of occurrences of track loss events over time. The second way is the track loss ratio, which is the relative proportion of the number of runs in which a track loss occurred to the total number (100) of sequence runs.

b: ACCURACY

The main goal of the SLAM algorithm is to estimate the position of a mobile system in space. Accordingly, one of the most important SLAM performance metrics is the accuracy of the localization. To evaluate the SLAM accuracy, the estimated position is compared with the real position, called *ground truth*. The difference in space (x, y, and z coordinates) is specified with the Absolute Trajectory Error (ATE). We use the Root Mean Square Error (RMSE) of the x, y, and z values of the ATE for our measurements. For accuracy measurements, we only evaluate successful runs of the test sequence in which no track loss events occurred.

c: CPU AND MEMORY UTILIZATION

Mobile devices are often powered by batteries and thus have a limited energy supply. Therefore, the goal is to keep the energy consumption low. Since SLAM requires a high computing power, one possibility would be to equip the mobile devices with powerful hardware. This has several disadvantages: (i) The resulting higher energy consumption reduces the battery life. For a mobile robot, the higher energy consumption shortens the range and operating time. (ii) The extra weight also causes a shorter range, since more energy is needed for acceleration. (iii) Higher required computing power also increases costs to provide each mobile system with additional hardware. Due to the limited capacity of the mobile system, smaller computing systems are often used, such as a Raspberry Pi. These systems are often based on processors with limited single-core performance and limited memory. Another factor is the main memory on mobile systems. This is in the single-digit gigabyte range for space and cost reasons. As shown in [25], the memory required to save the map grows continuously and may thereby exceed available memory.

As illustrated in Figure 2, higher computational power reduces the computational time of the individual modules. Thus, with a higher computational power, the time between the keyframes can be reduced and thus the reliability can be increased.

3) EVALUATION SETUP

a: TESTBED

To investigate SLAM offloading, we use a testbed with two different systems. A Raspberry Pi 4 with an eight-core processor and 4 GB RAM is chosen as a realistic mobile device since these systems are often used in mobile robotics. A Fujitsu K102-A100 with an Intel i7-6700T CPU and 16 GB of RAM was selected as the edge device. Although the Raspberry Pi 4 also has multiple cores, the Fujitsu PC that acts as an edge has higher processing power. Both systems are directly connected with each other via Ethernet in our testbed. An average RTT of 0.25 ms was measured with *ping*. To emulate different latencies, we use the Linux *NetEm* tool [80] to delay packets by a prescribed delay. Both systems use the Ubuntu 18.04. LTS operating system.

b: DATA SETS

In order to ensure reproducibility and comparability of the SLAM system evaluations, we utilized the common

TUM RGB-D dataset [28]. We use the test sequences `freiburg3_long_office_household_2` (abbreviated as `office`) and `freiburg_desk` (abbreviated as `desk`). The sequences `office` and `desk` have a duration of 87 s and 99 s, respectively. To ensure comparability, we truncated the `desk` sequence at 87 s. The sequence frames are played back using the Robot Operating System (ROS) [81]. ROS is a framework for robots that provides a variety of software libraries for operating a wide range of robot types. For the replaying of the benchmark datasets we use the so-called *rosbag* module. The benchmarking datasets were truncated from the original 30 Frames per Second (FPS) to 10 FPS. This was necessary to make sure that the Raspberry Pi 4 can replay the sequences in realtime via the ROS interface.

B. JUDICIOUS PARAMETER SETTINGS AND PARALLEL LOCAL MAP UPDATE FOR STATIC EDGE SLAM

1) ENHANCEMENTS

During the test of the Edge SLAM system [25], we noticed that it could not run sequences of the TUM RGB-D benchmark dataset [28] without track losses in the majority of the runs. In conventional Edge SLAM [25], a self-recorded data set was used, in which a robot moves at a average translational velocity of 0.085 m/s. On the other hand, the commonly used `office` sequence [28] has an average translational velocity of 0.249 m/s, i.e., almost three times faster than the data set in [25].

In order to enable the use of the faster public datasets, we first optimized Edge SLAM. We have identified two parameters that strongly influence the performance of Edge SLAM. One of them sets the maximum number of features n_{ORB} that are extracted per frame. The other is the number of keyframes n_{KF} that are bundled and sent as a local map update from the edge to the mobile device. The bundling of keyframes was introduced to reduce the required network bandwidth between the mobile device and the edge. Additionally, we found that the execution time of the local map update procedure on the mobile device could be reduced with parallelization. We reduced the number n_{KF} of keyframes used for local map updates from the conventional 6 to 1. In the original Edge SLAM [25], the old map was serially deleted with each update, a new map was created and filled with the new data. This caused an unnecessary delay, which we have eliminated with our process parallelization.

2) PERFORMANCE IMPLICATIONS

a: RELIABILITY & ACCURACY

Figure 7 shows the track loss ratio of ORB-SLAM2 without offloading and enhanced Edge SLAM as a function of the maximum number n_{ORB} of features per extracted frame. In ORB-SLAM2, the main reason for track loss is frame dropping due to a blocked tracking module. Because of the high computational load of the mobile device, it can happen that the processing of individual frames in the tracking module takes longer, which means that new incoming frames are not accepted. Figure 7 indicates that decreasing n_{ORB} does decrease the ORB-SLAM2 track loss ratio, since a lower n_{ORB} decreases the processing times.

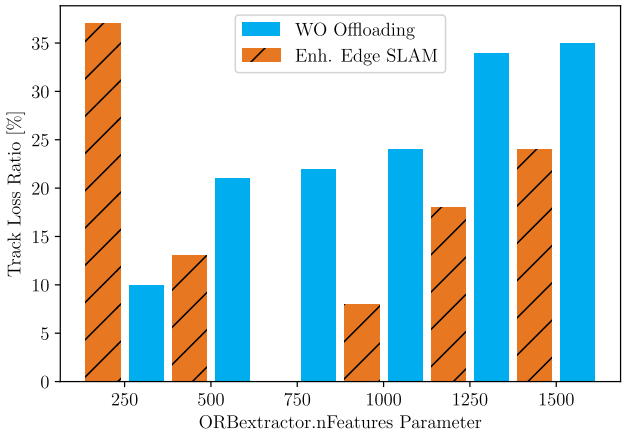


FIGURE 7. Track loss ratio as a function of maximum number n_{ORB} of extracted features per frame for conventional ORB-SLAM2 (without (WO) Offloading) on the mobile device, and enhanced Edge SLAM ($n_{\mathrm{KF}} = 1$, parallel local map update).

TABLE 1. Computing times [ms] of the different SLAM components, track loss ratio, and accuracy as ATE RMSE for conventional Edge SLAM ($n_{\mathrm{KF}} = 6$, $n_{\mathrm{ORB}} = 1000$, serial local map update) and enhanced Edge SLAM ($n_{\mathrm{KF}} = 1$, $n_{\mathrm{ORB}} = 750$, parallel local map update).

Parameter	Conventional	Enhanced
T_{Total} [ms]	904 ± 13.1	426 ± 12.6
T_{Track} [ms]	61 ± 33.5	61 ± 34.2
T_{MU} [ms]	581 ± 61.3	210 ± 69.3
T_{RKF} [ms]	33 ± 12.1	26 ± 10.6
T_{LM} [ms]	138 ± 61.3	107 ± 33.4
T_{PLMU} [ms]	92 ± 24.7	21 ± 10.5
Track Loss Ratio [%]	78	0
ATE RMSE [cm]	1.3 ± 0.2	1.3 ± 0.2

In Edge SLAM, the processing time is already reduced by offloading modules. However, local map gaps lead to track losses. With increasing n_{ORB}, T_{Total} increases, which leads to increased local map gaps and track losses. However, setting n_{ORB} too low can mean that not enough features are taken into account for matching. The default value in Edge SLAM as well as ORB-SLAM2 is $n_{\mathrm{ORB}} = 1000$. We can see in Figure 7 that for $n_{\mathrm{ORB}} = 750$, no track losses occur in Edge SLAM. The adapted $n_{\mathrm{ORB}} = 750$ and $n_{\mathrm{KF}} = 1$ as well as the introduced parallelization lead to significant reliability improvements, which are due to the reduction of T_{Total}.

Table 1 shows the average processing delays and corresponding standard deviations of the individual T_{Total} components for the conventional static Edge SLAM and the enhanced static Edge SLAM. We can see that all delays, except for tracking, are greatly reduced. The improvements have the largest relative influence on T_{PLMU}, i.e., the processing time of the local map update preparation, with a reduction of 77 %. In total, we can reduce the average computation delay T_{Total} by 52 %. Table 1 also indicates that the ATE RMSE accuracy (which was only considered from runs without track loss, for a fair evaluation) is nearly unchanged by enhanced static Edge SLAM. Therefore, we infer that with

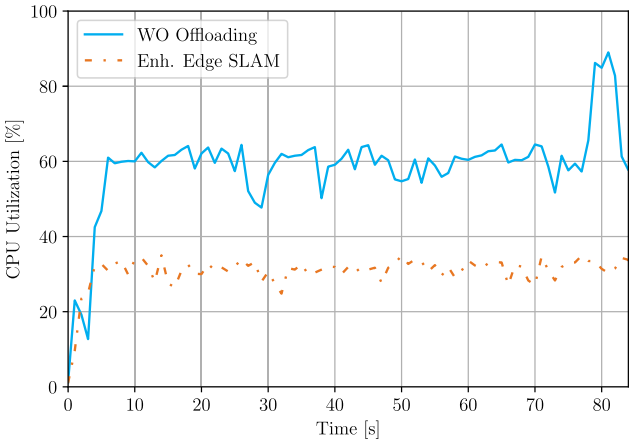


FIGURE 8. Mobile device (Raspberry Pi 4) CPU utilization as a function of time in sequence `office` [28] for conventional ORB-SLAM2 without offloading and for static enhanced Edge SLAM; 0 ms RTT latency for enhanced static Edge SLAM. The spike of the without offloading curve around 80 s is from the ORB-SLAM2 loop closing module.

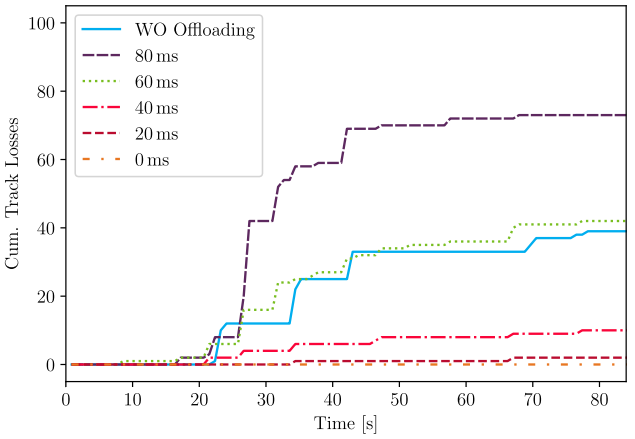


FIGURE 9. Cumulative number of track losses over 100 `office` sequence runs as a function of time for conventional ORB-SLAM2 on mobile device without (WO) offloading and for enhanced static Edge SLAM (Tracking on mobile device; LM and LC on edge) for different round-trip RTT network latencies (0 ms, 20 ms,. . ., 80 ms).

substantially fewer track losses (track loss ratio is reduced from 78 % to 0 %) and nearly the same ATE RMSE, the enhanced static Edge SLAM system is more stable.

b: CPU UTILIZATION

Figure 8 compares the mobile device (Raspberry Pi) CPU utilization levels of conventional ORB-SLAM2 vs. enhanced Edge SLAM. The average CPU utilization over a full run of the `office` test-sequence when running all ORB-SLAM2 modules on the mobile device is 58.5 %. If LM and LC are offloaded to the edge, the average CPU utilization on the mobile device drops to 30.6 %. Mobile device CPU utilization is relevant for two reasons. On the one hand, less mobile device CPU load provides more resources for the calculation of other important functions, such as route calculation or security functions. On the other hand, a higher CPU load leads generally to a higher power consumption [82], which lowers the range of the mobile device.

C. INFLUENCE OF LATENCY

As we observed in Figure 2, more computing power can speed up the SLAM processes. In the case of high network delays $T_{\mathrm{M}\to\mathrm{E}}$ and $T_{\mathrm{E}\to\mathrm{M}}$ caused by increased network latencies, we investigate the trade-off between an increased network latency and the advantage of more computing power at the edge. We assume symmetric links, i.e., we apply an egress latency on the mobile and on the edge device. We always specify the latency as the RTT, i.e., twice the respective added (one-way) egress delay. We measure the reliability, i.e., the occurrences of track loss events. Figure 9 shows the cumulative distribution of the number of track loss events over time from 100 runs. The track loss ratio of the SLAM system depends on the sequence, the computing capacities, and for Edge SLAM also on the network latency. We use this type of plot of track losses over time, to better observe the reasons of track loss. As described in Section III-A2.a, the reasons can be movements in the sequence or a high T_{Total} that result in local map gaps.

TABLE 2. Track loss ratio (cumulative number of track loss events at the end of the sequence for 100 runs) as a function of the network RTT for conventional and for enhanced static edge SLAM.

RTT	0 ms	20 ms	40 ms	60 ms	80 ms
Track Loss Ratio Conv. [%]	78	79	82	92	100
Track Loss Ratio Enh. [%]	0	2	10	48	73

In Figure 9, we can see that the track loss occurrences start to increase at about 22 seconds into the sequence in the case of running ORB-SLAM2 on the mobile device. This is due to an increased angular velocity at 22 s in the test sequence, as described in Section III-A2.a. The computation power on the mobile device is too low for keeping the processing rate high enough to establish sufficient matches between the keyframes for the increased angular velocity. The improved Edge SLAM without artificially added network latency (0 ms) can survive the increased angular velocity without track losses due to higher processing rate, which is due to higher computing power.

Table 2 shows the track loss ratio for the conventional Edge SLAM and for the enhanced Edge SLAM for different network RTT latencies. The enhancements help to gain more resilience for an increasing network latency, but the advantage of Edge SLAM still decreases for increasing network RTT latencies due to increasing local map gaps. In particular, we observe from Table 2 that the occurrence of track loss events starts to increase substantially at 40 ms, a typical RTT latency in 4G networks [83].

This demonstrates that first we can achieve a higher reliability by lowering the T_{Total} by offloading to a more powerful edge system. Second, we can see that the reliability gains diminish with rising network latency. Therefore, we propose to dynamically offload the SLAM computations depending on the current latency in the network. In the following sections, the enhanced version of Edge SLAM is referred to as *Static Edge SLAM*.

IV. PROPOSED DynNetSLAM APPROACH

As demonstrated in the preceding section, the offloading of SLAM processing has benefits in terms of reliability, accuracy, memory and energy consumption. However, we also discovered the negative influence of latency on the reliability and accuracy. This can be an issue, if the latency is high and fluctuates. In reality, the latency can rise and fluctuate depending on the channel conditions. Typical reasons can be interference, fading, and absorption. As a result, packet losses occur and retransmissions become necessary, which add delays to the network transmission. To cope with these dynamics, we propose an adaptive system that dynamically offloads the corresponding SLAM processes depending on the current network condition.

A. DynNetSLAM OVERVIEW

The underlying idea of Dynamic Visual SLAM Network Offloading (DynNetSLAM) is to monitor the current network state and adjust the offloading decision accordingly. We discovered two issues in Section III:

- Increased processing power can reduce the processing delay and thus increase reliability. Therefore, moving the computationally-intensive processes to the better-equipped edge is beneficial. The modules identified as most computationally intensive are LM and LC.
- A higher delay in the connection leads to a pronounced decrease in reliability. Hence, offloading is not beneficial if the transmission to the edge increases the delay substantially.

This leads to a classic trade-off scenario. Since we may experience increased latency in a wireless transmission, adding dynamic adaptation to the decision process is preferable. Therefore, we introduce DynNetSLAM, the dynamic offloading of the LM and LC modules depending on the network latency. Figure 10 shows the DynNetSLAM concept. Depending on the RTT, DynNetSLAM decides whether the LM and LC modules are executed in the edge or on the mobile device.

B. DESIGN GOALS & CONSIDERATIONS

A goal is to reduce the computing load on the mobile devices. Our main objective is to increase the SLAM reliability and accuracy by exploiting more powerful computing in the edge. We want to detect and recognize autonomously when the LM and LC modules should be offloaded to the edge. We aim to prevent that the advantage of the high computing power at the edge gets lost due to the influence of the RTT latency of the network.

To deal with these trade-offs, we design the DynNetSLAM system according to the following principles:

1) The network latencies $T_{\mathrm{M}\to\mathrm{E}}$ and $T_{\mathrm{E}\to\mathrm{M}}$ negatively affect the reliability of static Edge SLAM. Therefore, the latency should be constantly monitored.
2) A reasonable latency decision threshold should be determined: For latency values below the latency threshold, the offloading of the LM and LC modules to the edge leads to a higher reliability. Therefore, we take the latency as the criterion for the offloading decision.
3) The transition between the two system states, namely computing the modules on the edge or on the mobile device, should run smoothly so that there are no processing downtimes.

C. DynNetSLAM DESIGN

To meet the defined goals, we design a system that enables dynamic offloading of the LM and LC modules depending on the network latencies between the edge and mobile device $T_{\mathrm{E}\to\mathrm{M}}$ and vice versa $T_{\mathrm{M}\to\mathrm{E}}$.

1) DECISION THRESHOLD & STATES

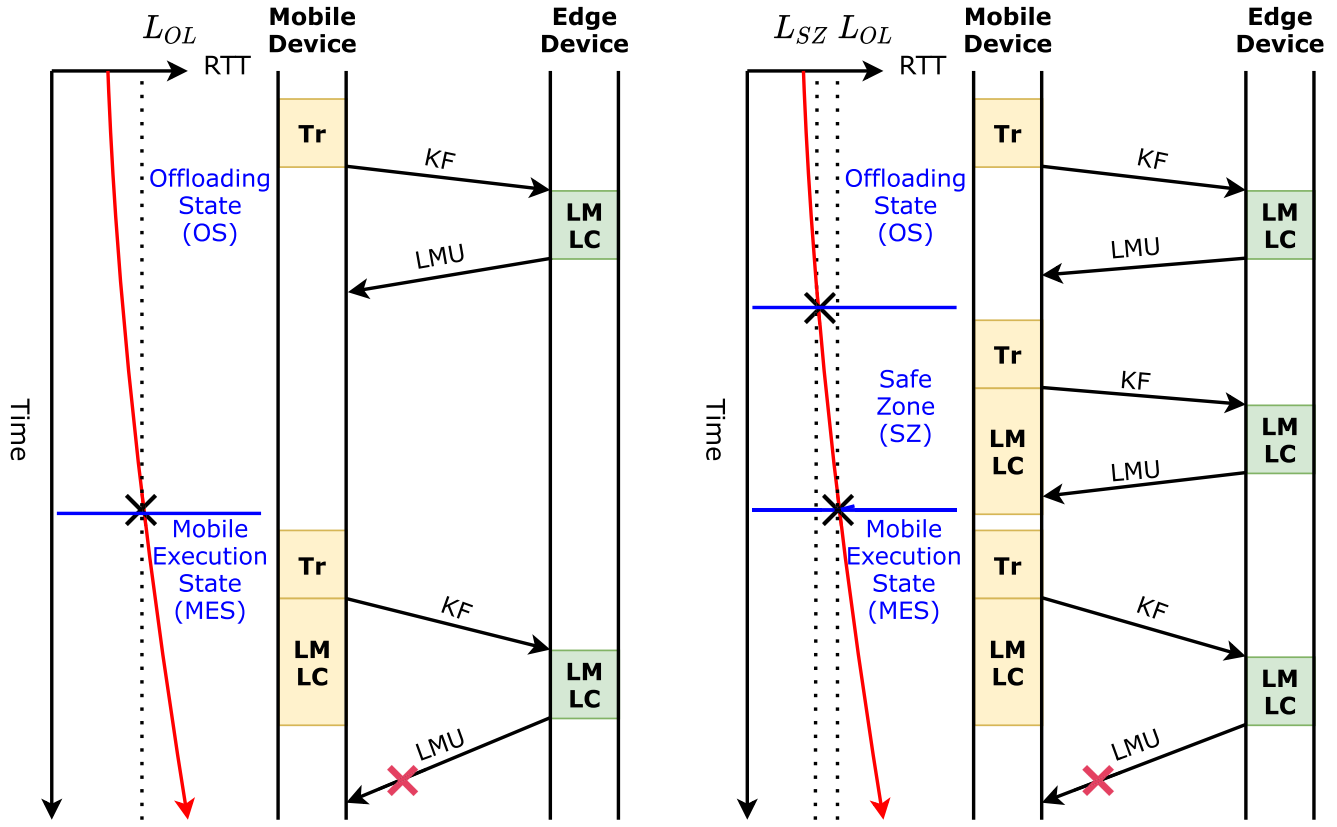
As shown in Figure 9, an increasing latency degrades the Edge SLAM performance by increasing the track loss ratio. To perform dynamic offloading reliably, we set the decision threshold depending on the latency. We define a static decision threshold L_{OL} as *Offloading Latency Threshold*.

Initially, we define two states that are illustrated in Figure 10a: In the *Mobile Execution State (MES)*, all SLAM modules are computed on the mobile device. In the *Offloading State (OS)*, the LM and LC modules are offloaded to the edge. The system switches between the states depending on L_{OL}. If the measured latency, i.e., RTT, is higher than L_{OL}, then DynNetSLAM switches the state to mobile execution (MES). If the measured RTT is lower than L_{OL}, then DynNetSLAM switches to offloading (OS).

2) SAFE ZONE STATE

The synchronization of the mobile device's map and the edge occurs at a fixed periodicity, e.g., every 5 seconds. Due to fast mobile device movements, which can be detected by observing the keyframe creation rate, the mobile device can decline local map updates in order to prevent track losses caused by large local map gaps. However, the map on the mobile device becomes less accurate as time passes without local map updates. If L_{OL} is exceeded during a phase of fast mobile device movement, the inaccurate local map on the mobile device must be used as the basis in the following MES. This can result in inaccurate pose estimations. To prevent the inaccurate calculation based on an outdated map, we introduce a so-called *Safe Zone (SZ)*.

The idea behind the SZ is that when the latency is close to the decision threshold, then the mobile device should start to optimize its own map in order to minimize inaccuracies that can arise from local map update rejections due to fast movements. Therefore, we introduce a third state, namely the SZ state, and a corresponding latency threshold L_{SZ}, with $L_{\mathrm{SZ}} < L_{\mathrm{OL}}$, as illustrated in Figure 10b. If the measured latency had previously been low enough that the edge was used to calculate the LM and LC modules, but there is now an increase in latency, then the SZ is entered at L_{SZ}. In the SZ, the LM and LC modules are started on the mobile device, to proactively prepare for the case that



(a) Without Safe Zone (SZ)

(b) With Safe Zone (SZ)

FIGURE 10. Illustration of DynNetSLAM without and with Safe Zone (SZ): Depending on the measured RTT network latency, the state changes between Offloading State (OS), Mobile Execution State (MES), and SZ. Depending on the current state, the Tracking (Tr), LM, and LC modules are executed on the mobile device or the edge. In the SZ, the local map update from the edge is still accepted on the mobile device.

no local map updates are accepted due to fast movements and thus no edge-supported map optimization can take place. If no fast movements occur, then the local map updates are accepted normally as in the OS, discarding the mobile device optimizations. The SZ has the advantage that a more accurate map is available on the mobile device in the case of fast movements due to the earlier start of the mobile device map optimizations.

3) HYSTERESIS CONTROL

A fixed state transition decision threshold, however, can be problematic if the measured latency fluctuates around the decision threshold. These fluctuations can lead to many state transitions despite only small changes of the measured variable. A typical example is a heating system, where the furnace starts up at a prescribed temperature decision threshold. Each furnace start-up leads to increased fuel consumption as well as wear-and-tear. If the temperature fluctuates around the threshold, it would lead to a excessively frequent switching on and off of the furnace.

A common solution is to use a hysteresis as a control function. The hysteresis introduces an offset in both directions of the temperature decision threshold to prevent excessively frequent state transitions. We intend to prevent unnecessary state transitions in our dynamic offloading of SLAM modules as well. Despite the previously introduced SZ, the excessively frequent transitions would likely increase the track loss ratio. Therefore, we introduce a hysteresis control with width B, as illustrated in Figure 11c.

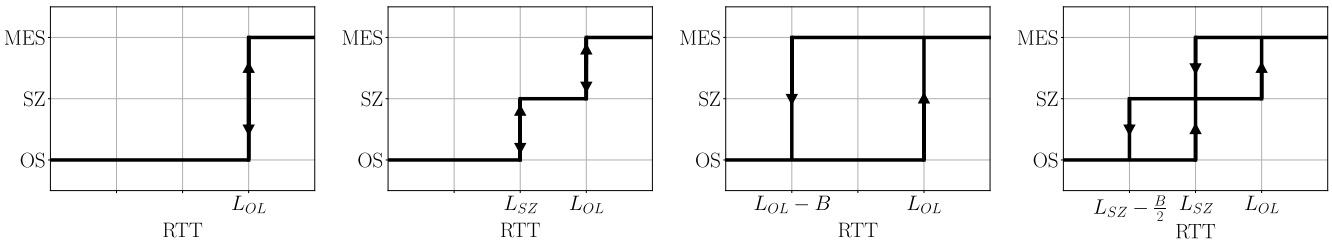
As described in Section IV-C, the SZ is intended to provide a smooth transition between the computation of the LM and LC modules on the mobile device and on the edge.

The hysteresis strategy prevents unnecessary state transitions during fluctuating latencies. Nevertheless, as described in Section IV-C2, situations can occur in which an outdated map is used on the mobile device. Therefore, we also introduce the strategy of a hysteresis with a SZ, as detailed in Algorithm 1. This leads to four different strategies as a function of the RTT, see Figure 11:

1) Without hysteresis and without Safe Zone (WO-H/WO-SZ)
2) Without hysteresis and with Safe Zone (WO-H/W-SZ)
3) With hysteresis and without Safe Zone (W-H/WO-SZ)
4) With hysteresis and with Safe Zone (W-H/W-SZ)

D. LATENCY MONITORING

Latency measurements can be performed in an active or passive manner. An active latency measurement, e.g., with ping, would mean that the edge sends packets to which the



(a) WO-H/WO-SZ (b) WO-H/W-SZ (c) W-H/WO-SZ (d) W-H/W-SZ

FIGURE 11. Illustration of the state dependency on the measured RTT for the four different strategies of DynNetSLAM.

Algorithm 1 Hysteresis Offloading Decision With Safe Zone (W-H/W-SZ)

```
procedure OffloadingDecision (prevState, L_OL, L_SZ, B)
    while NewKeyFramesReceived do
        RTT = receiveRTTFromSocket()
        if prevState == SZ then
            if RTT < L_SZ − B/2 then
                MoveToEdge()
                prevState = OS
            end if
            if RTT > L_OL then
                MoveToMobile()
                prevState = MES
            end if
        end if
        if prevState == MES then
            if RTT < L_SZ then
                MoveToSafeZone()
                prevState = SZ
            end if
        end if
        if prevState == OS then
            if RTT > L_SZ then
                MoveToSafeZone()
                prevState = SZ
            end if
        end if
    end while
end procedure
```

mobile device then responds to. The latency, specifically, the RTT, can then be determined from the difference between sending a packet and receiving the response. An active latency measurement method would create overhead in the implementation of the SLAM system. This could lead to unwanted delays in processing. The active method would also generate additional network traffic.

Since we work with TCP sockets in our implementation for the local map updates, we decided to use a passive monitoring approach. We consider the response time of the TCP acknowledgments for the latency measurement. We get this information from the sockets used for communication between the edge and mobile device. In particular, the RTT estimation `tcpi_rtt` [84] value is retrieved from the socket. The RTT estimation of TCP is updated continuously and hence delivers a timely estimation of the connection latency.

V. DynNetSLAM: PERFORMANCE EVALUATION

We now evaluate the novel DynNetSLAM approach of dynamically offloading the computationally intensive SLAM modules. First, we describe the used testbed and latency profiles. We then evaluate the influences of the dynamic offloading and the introduced SZ by comparing DynNetSLAM with static Edge SLAM.

A. TESTBED & BENCHMARK SEQUENCES

We use the data sets and sequences that are described in Section III-A3.a. To test the four strategies, introduced in Section IV-C3, we emulate two latency profiles. In the first latency profile, the artificially added latency increases linearly to 85 ms and decreases after 43 s again to 0 ms added latency. As already described in Section IV-C, we expect a lower reliability if strong latency fluctuations cause frequent state changes. To investigate these frequent transitions, we superimpose a fluctuating latency curve on a sine wave in the second latency profile. The base sine wave has a period of 43 s and oscillates between 10 ms and 70 ms. The superimposed wave has a period of 4 s and an amplitude of 20 ms. The artificial latencies are created by *NetEm* [80] to emulate the delays added by the wireless connection. For each test run, the sequence is played 100 times with the corresponding latency profiles.

B. DYNAMIC OFFLOADING & SAFE ZONE

We begin by investigating the dynamic offloading described in Section V-A for the linear latency curve. The decision threshold L_{OL} for the offloading is set to 40 ms. We choose this threshold because, as shown in Figure 9, the track loss ratio increases sharply thereafter. In practice, this means that when the decision threshold L_{OL} is exceeded, we switch to MES, i.e., all SLAM computations are again performed on the mobile device. If the latency drops below the L_{OL} threshold, then the LM and LC computations are moved back to the edge. We compare dynamic offloading with (WO-H/W-SZ) and without safe zone (WO-H/WO-SZ) compared to static Edge SLAM and to running ORB-SLAM2 on the

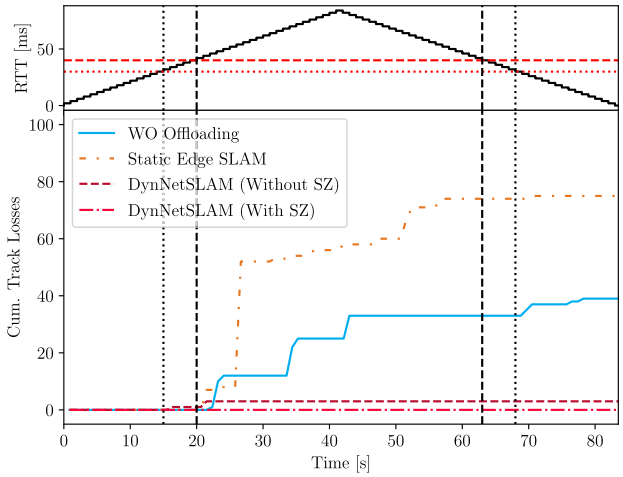


FIGURE 12. Cumulative number of track losses of 100 sequence runs over time of DynNetSLAM without (WO-H/WO-SZ) and with safe zone (WO-H/W-SZ) in comparison with static Edge SLAM and running ORB-SLAM2 on the mobile device (WO Offloading).

mobile device. Figure 12 shows the track loss events over time running the sequence `office` together with the $RTT = T_{NW} = T_{\mathrm{E}\to\mathrm{M}} + T_{\mathrm{M}\to\mathrm{E}}$ latency profile, whereby $L_{OL} = 40$ ms is marked in the latency profile (dashed) as well as the threshold for the safe zone $L_{\mathrm{SZ}} = L_{\mathrm{OL}} - B/2 = 35$ ms (dotted).

We observe from Figure 12 that the cumulative number of track loss occurrences of the static offloading with Edge SLAM increases strongly after 22 s runtime. Similarly, with ORB-SLAM2 on the mobile device, the track loss occurrences increase at 22 s runtime. The rotational movement described in Section III-A2.a and shown in Figure 6 leads to a strong increase of track loss occurrences for running ORB-SLAM2 on the mobile device (WO Offoading) and static Edge SLAM. A stronger increase can be observed with static Edge SLAM, with a track loss ratio of 75 %. This is due to the combination of high network latency and fast mobile device movement, resulting in large local map gaps. DynNetSLAM substantially reduces the occurrence of track losses. Strategy WO-H/WO-SZ results in three track losses for 100 sequence runs. We can see that the three WO-H/WO-SZ track losses occur at 22 s, which is the moment of rotational movement as well of switching the computation to the mobile device. With strategy WO-H/W-SZ, the computing starts earlier on the mobile device due to the safe zone. We can recognize that in the moment of the rotational movement, no track losses occur. The higher reliability of WO-H/W-SZ motivates us to use the SZ for the DynNetSLAM.

C. HYSTERESIS CONTROL

As explained in Section IV-C, latency fluctuations can lead to unnecessary switching between the states. To prevent this, we introduced hysteresis as a control function. To evaluate the performance of all strategies, we evaluate them with the superimposed sine wave, described in Section V-A, as the latency profile. The four variants described in Section IV-C

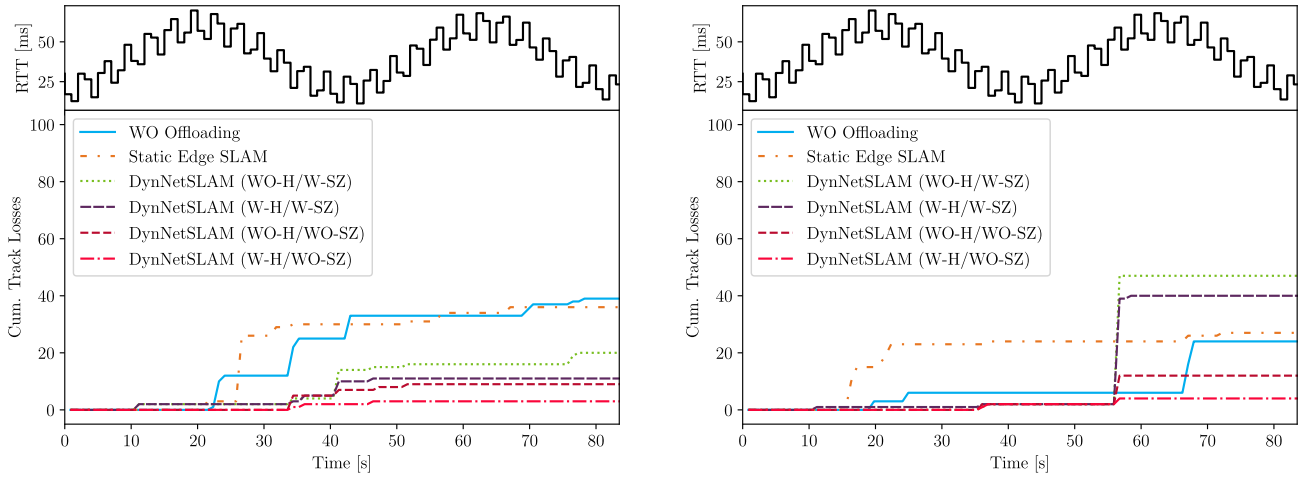
TABLE 3. Number of track losses for 100 replications of sequences `office` and `desk` for three different decision thresholds L_{OL} with sinusoidal latency profile.

Track Loss Ratio [%]						
Sequence	`office`			`desk`		
WO Offloading	39			24		
Static Edge SLAM	36			27		
	DynNetSLAM					
L_{OL} [ms]	30	40	50	30	40	50
WO-H/WO-SF	13	9	10	22	12	42
WO-H/W-SZ	11	20	45	5	47	33
W-H/WO-SZ	9	3	7	4	4	55
W-H/W-SZ	12	11	38	7	40	28

are evaluated in comparison with the static Edge SLAM and ORB-SLAM2 performed on the mobile device. The decision threshold was again set to $L_{\mathrm{OL}} = 40$ ms. For the width of the hysteresis $B = 10$ ms was chosen. Figure 13 shows the occurrence of track loss events.

All four dynamic offloading variants achieve a lower track loss ratio compared to static Edge SLAM and ORB-SLAM2 on the mobile device for both tested sequences. For the office sequence in Figure 13a, we observe that static Edge SLAM has a track loss ratio of 36 %, considerably higher than the dynamic variants with 9 % for WO-H/WO-SF, 20 % for WO-H/W-SZ, 3 % for W-H/WO-SZ and 11 % for W-H/W-SZ. This indicates that the hysteresis control can cope with the dynamic latencies in the network by lowering the track loss ratio from 36 % to 3 %. For the `desk` sequence in Figure 13b, similar robust results are achieved by introducing DynNetSLAM with hysteresis control (W-H/WO-SZ) with 4 % track loss ratio in comparison to static Edge SLAM with 27 %. Surprisingly, the SZ has a detrimental effect and increases the track loss ratio for the latency profile of the superimposed sine wave. The track loss ratio increases from 3 % (W-H/WO-SZ) to 11 % (W-H/W-SZ) for the `office` sequence. We furthermore observe a pronounced detrimental effect of the safe zone for the `desk` sequence, where the track loss ratio increases from 4 % (W-H/WO-SZ) to 40 % (W-H/W-SZ). We expect that this detrimental effect of the SZ is due to the additional computational load of running the LM and LC module on the mobile device. Therefore, T_{Mobile} is increased and as a result the occurrence of track losses due to local map gaps.

In order to investigate the influence of the L_{OL} threshold, we conducted evaluations for L_{OL} = 30, 40, and 50 ms. Table 3 compares the track loss ratio for all tested decision thresholds L_{OL} and sequences in comparison with static Edge SLAM and ORB-SLAM on the mobile device. We can see that the higher decision threshold leads to an increase of track loss occurrences caused by increased local map gaps in the offloading state. The lower decision threshold does not lead to consistently decreasing track losses as one could assume. This is due to the lower reliability of the computation on the weaker mobile device. From these results we can see that we are in a trade-off scenario between the reliability gain from



(a) `office` (b) `desk`

FIGURE 13. Cumulative number of track losses of 100 sequence runs over time within the two test sequences at sinusoidal superposition changing network latency. As decision threshold, $L_{OL} = 40$ ms was selected.

TABLE 4. ATE RMSE mean [cm] of successful sequence runs without track loss.

ATE RMSE Mean [cm]						
Sequence	`office`			`desk`		
ORB-SLAM2	2.2			2.1		
Static Edge SLAM	1.5			1.2		
	DynNetSLAM					
L_{OL} [ms]	30	40	50	30	40	50
WO-H/WO-SZ	1.9	1.7	1.7	1.4	1.3	1.2
WO-H/W-SZ	2.0	1.6	1.7	1.4	1.4	1.5
W-H/WO-SZ	1.7	1.6	1.4	1.5	1.4	1.2
W-H/W-SZ	1.5	1.7	1.5	1.5	1.5	1.3

TABLE 5. CPU utilization of different offloading strategies on the mobile device.

CPU Utilization [%]						
Sequence	`office`			`desk`		
WO Offloading	48.6			48.0		
Static Edge SLAM	34.3			33.5		
	DynNetSLAM					
L_{OL} [ms]	30	40	50	30	40	50
WO-H/WO-SF	37.1	36.2	35.7	39.4	38.6	36.4
WO-H/W-SZ	38.1	37.1	35.9	39.9	37.1	37.1
W-H/WO-SZ	38.5	37.1	36.8	39.9	38.6	37.1
W-H/W-SZ	38.6	36.3	36.1	40.6	38.8	36.1

higher computing power and an increased track loss ratio due to increased network latency.

Table 4 lists the accuracy as RMSE of the ATE for the successful runs, i.e., the runs without a track loss event. This means that only values are considered if the complete sequence run could be completed without track loss. We can see that for Edge SLAM, dynamic as well as static, the accuracy is higher than when running ORB-SLAM2 on the mobile device. The dynamic and static Edge SLAM variants have similar accuracy. This indicates that besides the higher reliability with less track loss events due to the dynamic offloading, the accuracy remains similar when dynamically offloading SLAM computational modules.

D. CPU UTILIZATION

As we bring computing back to the mobile device with our dynamic offloading, we naturally expect an increase in CPU utilization on the mobile device, which can increase the energy consumption [82]. We compare the average CPU utilization for the sine wave latency profile. Table 5 shows the average CPU utilization on the mobile device for the two different sequences. Our DynNetSLAM strategies are compared with running ORB-SLAM2 on the mobile device as well as with static Edge SLAM. We observe that the overall CPU utilization drops from 48.6 % to 34.3 % for static Edge SLAM offloading for the `office` sequence. The dynamic offloading leads to a relatively small CPU utilization increase. The strategy W-H/WO-SZ with $L_{OL} = 40$ ms exhibited the lowest track loss ratio and incurs a CPU utilization increase to 37.1 % for sequence `office` and 38.6 % for `desk`. In general, we can observe that a higher threshold L_{OL} leads to a lower CPU utilization for all strategies. This is due to the fact that the computation on the mobile device is started earlier with a lower L_{OL}.

VI. CONCLUSION AND FUTURE WORK

We designed, implemented, and evaluated DynNetSLAM, a system to offload vSLAM processing that adapts dynamically to changing network conditions in terms of latency. DynNetSLAM shifts compute-intensive functions from the mobile device to the edge or vice versa, depending on the latency measured in the network. We have proposed various dynamic offloading strategies, such as a hysteresis-based control system. We evaluated these strategies with publicly available sequences for SLAM performance benchmarking and applied two different latency profiles.

We found that higher computational power leads to higher reliability of SLAM, which motivates the offloading of processing to the edge for computationally weak and power-constrained devices. After the implementation of the

state-of-the-art approach, we first noticed that it could be further optimized to ensure higher reliability. We also found that Edge SLAM provides higher reliability than the use of native SLAM (ORB-SLAM2) on a typical mobile computing device. After that, the negative impact of latency on performance was evaluated. Even at a network latency of 40 ms, a typical latency for 4G networks [83], the track loss ratio increases to 10 %. At a latency of 60 ms, the track loss ratio is already 48 % and at 80 ms it increases to 73 %.

Therefore, offloading depending on the latency becomes imperative. To reliably detect high network latency, a monitoring function was implemented based on RTT estimation of TCP sockets. In Section IV, we presented DynNetSLAM with its four different strategies for dynamic offloading. We demonstrated that DynNetSLAM reduces the negative influence of network latency on Edge SLAM. DynNetSLAM allows to reduce the track loss ratio down to 3 % in comparison to 36 % without offloading while maintaining accuracy for a fluctuating latency profile. The hysteresis-based strategy (W-H/WO-SZ) reduces the influence of latency fluctuations and provides the lowest track loss ratio. Despite the dynamic relocation of the LM and LC modules back to the mobile device, a significantly lower CPU utilization on the mobile device could still be achieved compared to running the complete SLAM system on the mobile device.

DynNetSLAM can provide the basis for more sophisticated offloading strategies to be developed in future research. These can be integrated into more complex scenarios. One possibility is to scale up by using multiple robots in conjunction with multiple edge computing instances, as proposed in edge cloud computing. In particular, it will be interesting to integrate the dynamic offloading into collaborative multi-robot environments [85], [86], [87], [88] in future research.

ACKNOWLEDGMENT

A preliminary version of the enhancements of static edge SLAM in Sections III-B and III-C of this article appears in [1].

REFERENCES

[1] P. Sossalla, J. Hofer, J. Rischke, J. Busch, G. T. Nguyen, M. Reisslein, and H. P. F. Fitzek, "Optimizing Edge SLAM: Judicious parameter settings and parallelized map updates," in *Proc. IEEE Globecom*, Dec. 2022, pp. 1–6.
[2] D. Esparza and G. Flores, "The STDyn-SLAM: A stereo vision and semantic segmentation approach for VSLAM in dynamic outdoor environments," *IEEE Access*, vol. 10, pp. 18201–18209, 2022.
[3] K. Lv, Y. Zhang, Y. Yu, Z. Wang, and J. Min, "SIIS-SLAM: A vision SLAM based on sequential image instance segmentation," *IEEE Access*, early access, Jun. 30, 2022, doi: 10.1109/ACCESS.2022.3187541.
[4] J. Pak, J. Kim, Y. Park, and H. I. Son, "Field evaluation of path-planning algorithms for autonomous mobile robot in smart farms," *IEEE Access*, vol. 10, pp. 60253–60266, 2022.
[5] T. G. R. Reid, S. E. Houts, R. Cammarata, G. Mills, S. Agarwal, A. Vora, and G. Pandey, "Localization requirements for autonomous vehicles," *SAE Int. J. Connected Automated Vehicles*, vol. 2, no. 3, pp. 173–190, Sep. 2019.
[6] S. Yang, C. Zhao, Z. Wu, Y. Wang, G. Wang, and D. Li, "Visual SLAM based on semantic segmentation and geometric constraints for dynamic indoor environments," *IEEE Access*, vol. 10, pp. 69636–69649, 2022.
[7] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: Part I," *IEEE Robot. Autom. Mag.*, vol. 13, no. 2, pp. 99–110, Jun. 2006.
[8] M. R. Gkeka, A. Patras, N. Tavoularis, S. Piperakis, E. Hourdakis, P. Trahanias, C. D. Antonopoulos, S. Lalis, and N. Bellas, "FPGA accelerators for robust visual SLAM on humanoid robots," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2022, p. 51.
[9] A. M. Barros, M. Michel, Y. Moline, G. Corre, and F. Carrel, "A comprehensive survey of visual SLAM algorithms," *Robotics*, vol. 11, no. 1, p. 24, Feb. 2022.
[10] T. Taketomi, H. Uchiyama, and S. Ikeda, "Visual SLAM algorithms: A survey from 2010 to 2016," *IPSJ Trans. Comput. Vis. Appl.*, vol. 9, no. 1, pp. 1–11, Dec. 2017.
[11] M. Wasala, H. Szolc, and T. Kryjak, "An efficient real-time FPGA-based ORB feature extraction for an UHD video stream for embedded visual SLAM," *Electronics*, vol. 11, no. 14, Jul. 2022, Art. no. 2259.
[12] S. Song, H. Lim, S. Jung, and H. Myung, "G2P-SLAM: Generalized RGB-D SLAM framework for mobile robots in low-dynamic environments," *IEEE Access*, vol. 10, pp. 21370–21383, 2022.
[13] S. Zhang, L. Zheng, and W. Tao, "Survey and evaluation of RGB-D SLAM," *IEEE Access*, vol. 9, pp. 21367–21387, 2021.
[14] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," *IEEE Trans. Autom. Sci. Eng.*, vol. 12, no. 2, pp. 398–409, Apr. 2015.
[15] H. Zhang and L. Zhang, "Cloud robotics architecture: Trends and challenges," in *Proc. IEEE Int. Conf. Service-Oriented Syst. Eng. (SOSE)*, Apr. 2019, pp. 1–6.
[16] T. V. Doan, G. T. Nguyen, M. Reisslein, and F. H. P. Fitzek, "FAST: Flexible and low-latency state transfer in mobile edge computing," *IEEE Access*, vol. 9, pp. 115315–115334, 2021.
[17] T. V. Doan, G. T. Nguyen, M. Reisslein, and F. H. P. Fitzek, "SAP: Subchain-aware NFV service placement in mobile edge cloud," *IEEE Trans. Netw. Service Manage.*, early access, Aug. 24, 2022, doi: 10.1109/TNSM.2022.3201388.
[18] F. H. Fitzek, S.-C. Li, S. Speidel, T. Strufe, M. Simsek, and M. Reisslein, *Tactile Internet: With Human-in-the-Loop*. London, U.K.: Academic, 2021.
[19] H. Jin, M. A. Gregory, and S. Li, "A review of intelligent computation offloading in multiaccess edge computing," *IEEE Access*, vol. 10, pp. 71481–71495, 2022.
[20] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, 2016.
[21] R. F. Vieira, D. D. S. Souza, M. S. D. Silva, and D. L. Cardoso, "A heuristic for load distribution on data center hierarchy: A MEC approach," *IEEE Access*, vol. 10, pp. 69462–69471, 2022.
[22] J. Rischke, P. Sossalla, S. Itting, F. H. P. Fitzek, and M. Reisslein, "5G campus networks: A first measurement study," *IEEE Access*, vol. 9, pp. 121786–121803, 2021.
[23] P. Sossalla, J. Rischke, G. T. Nguyen, and F. H. P. Fitzek, "Offloading robot control with 5G," in *Proc. IEEE 19th Annu. Consum. Commun. Netw. Conf. (CCNC)*, Jan. 2022, pp. 461–464.
[24] P. Sossalla, J. Rischke, J. Hofer, and F. H. P. Fitzek, "Evaluating the advantages of remote SLAM on an edge cloud," in *Proc. 26th IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2021, pp. 1–4.
[25] A. J. Ben Ali, Z. S. Hashemifar, and K. Dantu, "Edge-SLAM: Edge-assisted visual simultaneous localization and mapping," in *Proc. 18th Int. Conf. Mobile Syst., Appl., Services*, Jun. 2020, pp. 325–337.
[26] J. Xu, H. Cao, D. Li, K. Huang, C. Qian, L. Shangguan, and Z. Yang, "Edge assisted mobile semantic visual SLAM," in *Proc. IEEE Conf. Comput. Commun.*, Jul. 2020, pp. 1828–1837.
[27] K.-L. Wright, A. Sivakumar, P. Steenkiste, B. Yu, and F. Bai, "CloudSLAM: Edge offloading of stateful vehicular applications," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, Nov. 2020, pp. 139–151.
[28] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of RGB-D SLAM systems," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Oct. 2012, pp. 573–580.
[29] T. Bailey and H. Durrant-Whyte, "Simultaneous localization and mapping (SLAM): Part II," *IEEE Robot. Autom. Mag.*, vol. 13, no. 3, pp. 108–117, Sep. 2006.
[30] R. Bajaj, S. L. Ranaweera, and D. P. Agrawal, "GPS: Location-tracking technology," *Computer*, vol. 35, no. 4, pp. 92–94, Apr. 2002.
[31] D. M. Cole and P. M. Newman, "Using laser range data for 3D SLAM in outdoor environments," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2006, pp. 1556–1563.
[32] C. Kerl, J. Sturm, and D. Cremers, "Dense visual SLAM for RGB-D cameras," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Nov. 2013, pp. 2100–2106.

[33] D. Nistér, O. Naroditsky, and J. Bergen, "Visual odometry," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 1, Jun./Jul. 2004, pp. 1–8.

[34] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Trans. Robot.*, vol. 32, no. 6, pp. 1309–1332, Dec. 2016.

[35] F. Lu and E. Milios, "Globally consistent range scan alignment for environment mapping," *Auto. Robots*, vol. 4, no. 4, pp. 333–349, Dec. 1997.

[36] G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based SLAM," *IEEE Intell. Transp. Syst. Mag.*, vol. 2, no. 4, pp. 31–43, Nov. 2010.

[37] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, "ORB-SLAM: A versatile and accurate monocular SLAM system," *IEEE Trans. Robot.*, vol. 31, no. 5, pp. 1147–1163, Oct. 2015.

[38] M. Labbé and F. Michaud, "RTAB-Map as an open-source LiDAR and visual simultaneous localization and mapping library for large-scale and long-term online operation," *J. Field Robot.*, vol. 36, no. 2, pp. 416–446, 2019.

[39] N. Ragot, R. Khemmar, A. Pokala, R. Rossi, and J.-Y. Ertaud, "Benchmark of visual SLAM algorithms: ORB-SLAM2 vs RTAB-Map," in *Proc. 8th Int. Conf. Emerg. Secur. Technol. (EST)*, Jul. 2019, pp. 1–6.

[40] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras," *IEEE Trans. Robot.*, vol. 33, no. 5, pp. 1255–1262, Oct. 2017.

[41] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle adjustment—A modern synthesis," in *Proc. Int. Workshop Vis. Algorithms*, 1999, pp. 298–372.

[42] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," in *Proc. Int. Conf. Comput. Vis.*, Nov. 2011, pp. 2564–2571.

[43] S. Semenova, S. Y. Ko, Y. D. Liu, L. Ziarek, and K. Dantu, "A quantitative analysis of system bottlenecks in visual SLAM," in *Proc. 23rd Annu. Int. Workshop Mobile Comput. Syst. Appl.*, Mar. 2022, pp. 74–80.

[44] S. Semenova, P. Meshram, T. Chase, S. Y. Ko, Y. D. Liu, L. Ziarek, and K. Dantu, "A modular, extensible framework for modern visual SLAM systems," in *Proc. 20th Annu. Int. Conf. Mobile Syst., Appl. Services*, Jun. 2022, pp. 579–580.

[45] A. Grunnet-Jepsen, P. Winer, A. Takagi, J. Sweetser, K. Zhao, T. Khuong, D. Nie, and J. Woodfill. (2018). *Using the Intel® RealSense™ Depth Cameras D4xx in Multi-Camera Configurations*. Accessed: Sep. 2, 2022. [Online]. Available: https://www.dev.intelrealsense.com/docs/multiple-depth-cameras-configuration

[46] C. Asavasirikulkij, C. Mathong, T. Sinthumongkolchai, R. Chancharoen, and W. Asdomwised, "Low latency peer to peer robot wireless communication with edge computing," in *Proc. IEEE 11th Int. Conf. Syst. Eng. Technol. (ICSET)*, Nov. 2021, pp. 100–105.

[47] A. W. Malik, A. U. Rahman, M. Ali, and M. M. Santos, "Symbiotic robotics network for efficient task offloading in smart industry," *IEEE Trans. Ind. Informat.*, vol. 17, no. 7, pp. 4594–4601, Jul. 2021.

[48] S. Chinchali, A. Sharma, J. Harrison, A. Elhafsi, D. Kang, E. Pergament, E. Cidon, S. Katti, and M. Pavone, "Network offloading policies for cloud robotics: A learning-based approach," *Auto. Robots*, vol. 45, no. 7, pp. 997–1012, Oct. 2021.

[49] M. Groshev, G. Baldoni, L. Cominardi, A. D. L. Oliva, and R. Gazda, "Edge robotics: Are we ready? An experimental evaluation of current vision and future directions," *Digit. Commun. Netw.*, vol. 2022, pp. 1–14, May 2022.

[50] S. Liu, L. Liu, J. Tang, B. Yu, and W. Shi, "Edge computing for autonomous driving: Opportunities and challenges," *Proc. IEEE*, vol. 107, no. 8, pp. 1697–1716, Jun. 2019.

[51] Y. Wang, W. Wang, D. Liu, X. Jin, J. Jiang, and K. Chen, "Enabling edge-cloud video analytics for robotics applications," in *Proc. IEEE Conf. Comput. Commun.*, May 2021, pp. 1–10.

[52] Z. Fan, W. Chen, G. Zhu, Y. You, F. Deng, Y. Hou, W. Liang, R. Fu, J. Xin, J. Chen, and H. Wang, "Collaborative robot transport system based on edge computing," in *Proc. IEEE 9th Annu. Int. Conf. CYBER Technol. Autom., Control, Intell. Syst. (CYBER)*, Jul. 2019, pp. 1320–1326.

[53] C.-H. Lu and K.-T. Lai, "Dynamic offloading on a hybrid edge–cloud architecture for multiobject tracking," *IEEE Syst. J.*, early access, Apr. 27, 2022, doi: 10.1109/JSYST.2022.3165571.

[54] D. Spatharakis, M. Avgeris, N. Athanasopoulos, D. Dechouniotis, and S. Papavassiliou, "Resource-aware estimation and control for edge robotics: A set-based approach," *IEEE Internet Things J.*, early access, Jan. 7, 2022, doi: 10.1109/JIOT.2022.3141266.

[55] Q. Liu, Y. Zhang, and H. Wang, "EdgeMap: CrowdSourcing high definition map in automotive edge computing," 2022, *arXiv:2201.07973*.

[56] L. Liu and M. Gruteser, "EdgeSharing: Edge assisted real-time localization and object sharing in urban streets," in *Proc. IEEE Conf. Comput. Commun.*, May 2021, pp. 1–10.

[57] S. Hayat, R. Jung, H. Hellwagner, C. Bettstetter, D. Emini, and D. Schnieders, "Edge computing in 5G for drone navigation: What to offload?" *IEEE Robot. Autom. Lett.*, vol. 6, no. 2, pp. 2571–2578, Apr. 2021.

[58] F. Ahmad, C. Shin, E. Chai, K. Sundaresan, and R. Govindan, "ARES: Accurate, autonomous, near real-time 3D reconstruction using drones," 2021, *arXiv:2104.08634*.

[59] P. Huang, L. Zeng, K. Luo, J. Guo, Z. Zhou, and X. Chen, "ColaSLAM: Real-time multi-robot collaborative laser SLAM via edge computing," in *Proc. IEEE/CIC Int. Conf. Commun. China (ICCC)*, Jul. 2021, pp. 242–247.

[60] P. Huang, L. Zeng, X. Chen, L. Huang, Z. Zhou, and S. Yu, "Edge robotics: Edge-computing-accelerated multirobot simultaneous localization and mapping," *IEEE Internet Things J.*, vol. 9, no. 15, pp. 14087–14102, Aug. 2022.

[61] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with Rao-Blackwellized particle filters," *IEEE Trans. Robot.*, vol. 23, no. 1, pp. 34–46, Feb. 2007.

[62] B. L. E. A. Balasuriya, B. A. H. Chathuranga, B. H. M. D. Jayasundara, N. R. A. C. Napagoda, S. P. Kumarawadu, D. P. Chandima, and A. G. B. P. Jayasekara, "Outdoor robot navigation using GMapping based SLAM algorithm," in *Proc. Moratuwa Eng. Res. Conf. (MERCon)*, Apr. 2016, pp. 403–408.

[63] S. Kamburugamuve, H. He, G. Fox, and D. Crandall, "Cloud-based parallel implementation of SLAM for mobile robots," in *Proc. Int. Conf. Internet Things Cloud Comput.*, Mar. 2016, pp. 1–7.

[64] M. Fukui, Y. Ishiwata, T. Ohkawa, and M. Sugaya, "IoT edge server ROS node allocation method for multi-SLAM on many-core," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops Affiliated Events (PerCom Workshops)*, Mar. 2022, pp. 421–426.

[65] V. K. Sarker, J. P. Queralta, T. N. Gia, H. Tenhunen, and T. Westerlund, "Offloading SLAM for indoor mobile robots with edge-fog-cloud computing," in *Proc. 1st Int. Conf. Adv. Sci., Eng. Robot. Technol. (ICASERT)*, May 2019, pp. 1–6.

[66] S. Dey and A. Mukherjee, "Robotic SLAM: A review from fog computing and mobile edge computing perspective," in *Proc. 13th Int. Conf. Mobile Ubiquitous Syst., Comput. Netw. Services*, Nov. 2016, pp. 153–158.

[67] B. D. Gouveia, D. Portugal, D. C. Silva, and L. Marques, "Computation sharing in distributed robotic systems: A case study on SLAM," *IEEE Trans. Autom. Sci. Eng.*, vol. 12, no. 2, pp. 410–422, Apr. 2015.

[68] W. Chen, Y. Yaguchi, K. Naruse, Y. Watanobe, and K. Nakamura, "QoS-aware robotic streaming workflow allocation in cloud robotics systems," *IEEE Trans. Services Comput.*, vol. 14, no. 2, pp. 544–558, Mar. 2021.

[69] Z. Hong, H. Huang, S. Guo, W. Chen, and Z. Zheng, "QoS-aware cooperative computation offloading for robot swarms in cloud robotics," *IEEE Trans. Veh. Technol.*, vol. 68, no. 4, pp. 4027–4041, Apr. 2019.

[70] Z. Zhou, G. Zhang, F. Zheng, T. Wang, L. Chen, and N. Duan, "A graph optimization-based acoustic SLAM edge computing system offering centimeter-level mapping services with reflector recognition capability," *Secur. Commun. Netw.*, vol. 2021, pp. 1–17, Dec. 2021.

[71] L. Riazuelo, J. Civera, and J. M. Montiel, "C^2TAM: A cloud framework for cooperative tracking and mapping," *Robot. Auto. Syst.*, vol. 62, no. 4, pp. 401–413, 2014.

[72] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," in *Proc. 6th IEEE ACM Int. Symp. Mixed Augmented Reality*, Nov. 2007, pp. 225–234.

[73] P. Benavidez, M. Muppidi, P. Rad, J. J. Prevost, M. Jamshidi, and L. Brown, "Cloud-based realtime robotic visual SLAM," in *Proc. Annu. IEEE Syst. Conf. (SysCon)*, Apr. 2015, pp. 773–777.

[74] J. Xu, H. Cao, Z. Yang, L. Shangguan, J. Zhang, X. He, and Y. Liu, "SwarmMap: Scaling up real-time collaborative visual SLAM at the edge," in *Proc. USENIX Symp. Netw. Syst. Design Impl. (NSDI)*, 2022, pp. 977–993.

[75] X. Cui, C. Lu, and J. Wang, "3D semantic map construction using improved ORB-SLAM2 for mobile robot in edge computing environment," *IEEE Access*, vol. 8, pp. 67179–67191, 2020.

[76] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," 2018, *arXiv:1804.02767*.

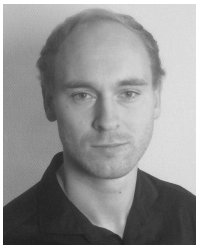
[77] J. Tang, S. Liu, L. Liu, B. Yu, and W. Shi, "LoPECS: A low-power edge computing system for real-time autonomous driving services," *IEEE Access*, vol. 8, pp. 30467–30479, 2020.

[78] D. Spatharakis, M. Avgeris, N. Athanasopoulos, D. Dechouniotis, and S. Papavassiliou, "A switching offloading mechanism for path planning and localization in robotic applications," in *Proc. Int. Conf. Internet Things (iThings) IEEE Green Comput. Commun. (GreenCom) IEEE Cyber, Phys. Social Comput. (CPSCom) IEEE Smart Data (SmartData) IEEE Congr. Cybermatics (Cybermatics)*, Nov. 2020, pp. 77–84.
[79] S. Schubert, S. Lange, P. Neubert, and P. Protzel, "Map enhancement with track-loss data in visual SLAM," in *Proc. Int. Conf. Intell. Robots Syst. (IROS)*, Daejeon, South Korea, 2016, pp. 9–14.
[80] S. Hemminger, "Network emulation with NetEm," in *Proc. Aust.'s Nat. Linux Conf.*, vol. 5, 2005, p. 2005.
[81] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: An open-source robot operating system," in *Proc. ICRA Workshop Open Source Softw.*, vol. 3, 2009, p. 5.
[82] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," *ACM SIGARCH Comput. Archit. News*, vol. 35, no. 2, pp. 13–23, 2007.
[83] E. Lyczkowski, H. A. Munz, W. Kiess, and P. Joshi, "Performance of private LTE on the factory floor," in *Proc. IEEE Int. Conf. Commun. Workshops (ICC Workshops)*, Jun. 2020, pp. 1–6.
[84] M. Sargent, J. Chu, D. V. Paxson, and M. Allman, *Computing TCP's Retransmission Timer*, document RFC 6298, Jun. 2011, pp. 1–11. [Online]. Available: https://www.rfc-editor.org/info/rfc6298
[85] H. Lee and S. Lee, "Extended spectra-based grid map merging with unilateral observations for multi-robot SLAM," *IEEE Access*, vol. 9, pp. 79651–79662, 2021.
[86] E. A. A. Memon, S. R. U. N. Jafri, and S. M. U. Ali, "A rover team based 3D map building using low cost 2D laser scanners," *IEEE Access*, vol. 10, pp. 1790–1801, 2022.
[87] M. Ouyang, X. Shi, Y. Wang, Y. Tian, Y. Shen, D. Wang, P. Wang, and Z. Cao, "A collaborative visual SLAM framework for service robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2021, pp. 8679–8685.
[88] Z. Zhu, W. Jiang, L. Yang, and Z. Luo, "Indoor multi-robot cooperative mapping based on geometric features," *IEEE Access*, vol. 9, pp. 74574–74588, 2021.

PETER SOSSALLA received the Diploma (Dipl.-Ing.) degree in electrical engineering, in 2019. He continued at the Deutsche Telekom Chair of Communication to pursue the Ph.D. degree. His current research interests include robotics, software-defined networking (SDN), time-sensitive networking (TSN), and edge computing.

JOHANNES HOFER received the Diploma (Dipl.-Ing.) degree in electrical engineering from Technical University Dresden, Germany, in 2022. His current research interests include the field of cloud robotics and in particular the offloading of computation in SLAM systems.

JUSTUS RISCHKE received the Diploma (Dipl.-Ing.) degree in electrical engineering from Technical University Dresden (TU Dresden), Dresden, Germany, in 2017. He is currently pursuing the Ph.D. degree with the Deutsche Telekom Chair of Communication Networks. His research interests include network coding and reinforcement learning in software-defined networks (SDN) for low latency communication.



CHRISTIAN VIELHAUS received the Diploma (Dipl.-Ing.) degree in electrical engineering from Technical University Dresden (TU Dresden), Germany, in 2019, where he is currently pursuing the Ph.D. degree in electrical engineering with the Deutsche Telecom Chair of Communication Networks. His research interests include machine learning, network simulation, and congestion control.

GIANG T. NGUYEN received the Ph.D. degree in computer science from TU Dresden, Germany, in 2016. He is currently an Assistant Professor, heading the Haptic Communication Systems Research Group, Faculty of Electrical and Computer Engineering, TU Dresden. His research interests include network softwarization and distributed systems.

MARTIN REISSLEIN (Fellow, IEEE) received the Ph.D. degree in systems engineering from the University of Pennsylvania, Philadelphia, PA, USA, in 1998. He is currently a Professor with the School of Electrical, Computer, and Energy Engineering, Arizona State University (ASU), Tempe, AZ, USA. He is currently an Associate Editor of IEEE Access, IEEE Transactions on Education, IEEE Transactions on Mobile Computing, and IEEE Transactions on Network and Service Management. He currently serves as an Area Editor of *Optical Communications* for the IEEE Communications Surveys and Tutorials and as a Co-Editor-in-Chief of *Optical Switching and Networking*. He served as an Associate Editor of the IEEE/ACM Transaction on Networking (2009–2013), served as an Associate Editor-in-Chief of the IEEE Communications Surveys and Tutorials (2007–2020), and chaired the Steering Committee of the IEEE Transactions on Multimedia, from 2017 to 2019. He received the IEEE Communications Society Best Tutorial Paper Award, in 2008, and a Friedrich Wilhelm Bessel Research Award from the Alexander von Humboldt Foundation, in 2015.

FRANK H. P. FITZEK (Senior Member, IEEE) received the Diploma (Dipl.-Ing.) degree in electrical engineering from the University of Technology—Rheinisch-Westfälische Technische Hochschule (RWTH), Aachen, Germany, in 1997, the Ph.D. (Dr.-Ing.) degree in electrical engineering from Technical University Berlin, Germany, in 2002, and the Honorary (Doctor Honoris Causa) degree from the Budapest University of Technology and Economy (BUTE), in 2015. He is currently a Professor and the Head of the Deutsche Telekom Chair of Communication Networks, Technical University Dresden, Germany, coordinating the 5G Laboratory, Germany. He is the Spokesman of the DFG Cluster of Excellence CeTI. He became an Adjunct Professor at the University of Ferrara, Italy, in 2002. In 2003, he joined Aalborg University as an Associate Professor and later became a Professor. He co-founded several start-up companies starting with acticom GmbH, Berlin, in 1999. He was selected to receive the NOKIA Champion Award several times in a row, from 2007 to 2011. In 2008, he was awarded the Nokia Achievement Award for his work on cooperative networks. In 2011, he received the SAPERE AUDE research grant from the Danish Government. In 2012, he received the Vodafone Innovation Prize. His current research interests include the areas of wireless and mobile 5G communication networks, mobile phone programming, network coding, cross layer, and energy efficient protocol design and cooperative networking.