Accepted for presentation in: 2025 IEEE Future Networks World Forum (FNWF)

Robots on the Edge: Latency-Critical and Sensitive Applications on 6G-Enabled Infrastructure

Manuel Patchou[1], Ching-Chi Lin[2], Niklas A. Wagner[1], Tim Gebauer[1], Kevin-Ismet Šabanović[1], Jannik Malcher[2], Stefan Böcker[1], Jian-Jia Chen[2], Christian Wietfeld[1]
[1]Communication Networks Institute (CNI), TU Dortmund University, 44227 Dortmund, Germany
[2]Design Automation for Embedded Systems Group, TU Dortmund University, 44227 Dortmund, Germany
Email: {firstname.lastname}@tu-dortmund.de

***Abstract*—Disaster relief missions are often time-sensitive, making network latency a critical factor for robotic operations. While robot applications designed for these missions must be robust in challenging conditions, the underlying networks must also support latency-critical operations. In this work, we formulate two rescue robotics scenarios with differentiated latency criticality. The first scenario, *Chemical Hazard Control*, focuses on immersive user experience, while the second, *Catching the Falling Rod*, is a latency-critical mission. We benchmark both scenarios with emulated network latency to assess their performance. The second scenario is further deployed on an Open RAN 6G research infrastructure, where ROS 2 node deployment strategies are investigated for meeting strict latency requirements. The results show that a 100ms latency increase can significantly impact mission outcomes in the first scenario, increasing completion time by 50% and quadrupling the failure rate. In the second scenario, proactive resource scheduling is essential for meeting real-time requirements. These findings validate 6G's hyper-reliable low-latency goals as essential for real time robotic control.**

***Video Abstract*—Demo video available online at: *https://tiny.cc/6GRealtimeRobotics*.**

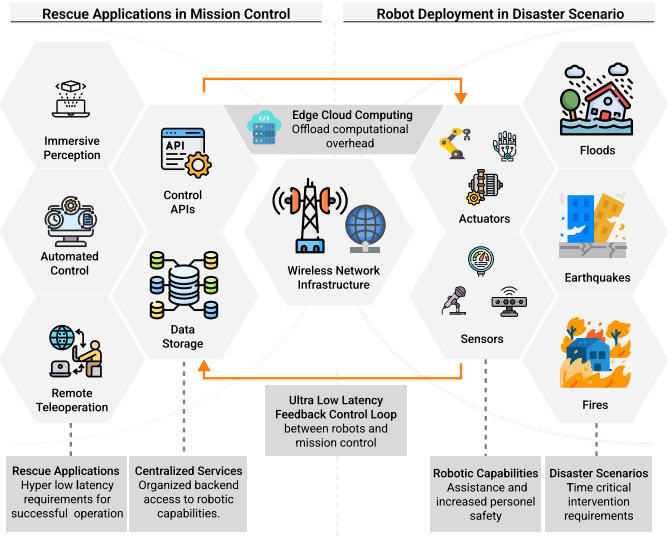

Fig. 1. Network infrastructure with edge-cloud extension as the enabler of latency-critical robotic applications architectures designed for disaster relief scenarios.

I. Introduction

Robotic platforms are reaching a level of operational readiness that warrants their integration into public safety and disaster relief situations, where timely data acquisition and rapid reaction are essential for efficient decision-making. While robot deployments can significantly reduce risks for first responders, a reliable and high-performance network infrastructure is a prerequisite.

Ongoing research on future networks is therefore of great importance to the field of rescue robotics, since it can fuel innovation in rescue robotic applications. For example, low network latencies enabled by Open Radio Access Network (Open RAN) research for the sixth-generation mobile radio networks (6G) can allow the relocation of energy-intensive and latency-sensitive applications, such as locomotion control, to a Mobile Edge Cloud (MEC) instance [1]. This approach extends robot battery life by offloading computationally demanding tasks. Furthermore, the 6G standard is envisioned as an enabler for augmented reality, which could unlock the potential for immersive robot control [2]. Figure 1 depicts the logical architecture of a typical rescue robot mission, where the network infrastructure connects the safe mission control area to the hazardous theater of operations.

This work emphasizes the central role of network infrastructure and the impact of 6G-envisioned solutions on robotic control applications designed for emergency and crisis management. Specifically, we formulate two rescue robotics scenarios: *Chemical Hazard Control* and *Catching the Falling Rod*. In the *Chemical Hazard Control* scenario, an operator remotely controls a robotic arm in an augmented reality environment to seal a jerrycan of hazardous chemicals. This scenario focuses on how network performance affects user experiences and control. On the other hand, the *Catching the Falling Rod* scenario is a latency-critical mission where a falling rod must be caught in time to prevent damage or injury. This mission is used to evaluate how 6G-envisioned solutions can meet strict latency requirements.

Our contributions are summarized as follows:

- Implementation of two distinct latency-sensitive rescue scenarios, with tailored control applications for each.
- Impact analysis of ROS 2 node deployment strategies on system latency.
- Benchmarking of the control applications under varying

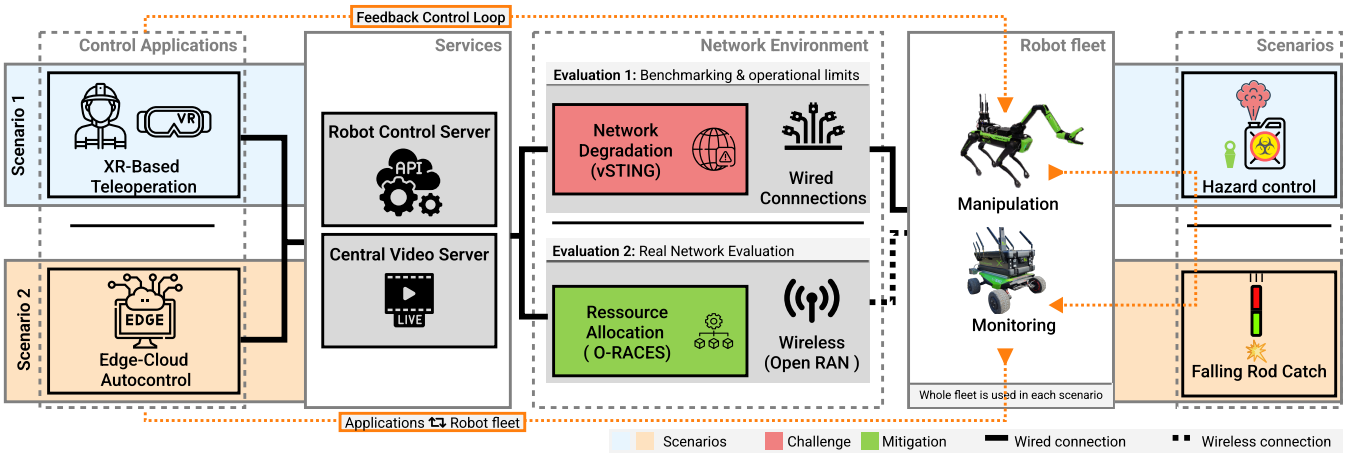

Fig. 2. System architecture of our experimental rescue robotic setup. This architecture allows for the validation of application latency in a controlled benchmarking environment and on a 6G-enabled research network.

latency conditions.

- Validation of benchmarking results on a real-world 6G research network.

The remainder of the paper is structured as follows: Section II provides a brief overview of related work. The general evaluation architecture is introduced in Section III, while the evaluation scenarios are presented in Section IV. Section V details the evaluation methodology, followed by the evaluation results and discussion. Lastly, our findings are summarized in Section VI.

II. Related Work

In emergency situations, the rapid deployment of communication infrastructure is essential for coordinating rescue efforts, gathering and disseminating information. With the progressive integration of robotic solutions in rescue scenarios, the communication infrastructure must also support massive machine-type communication and provide the guarantees needed by the applications powering these robotic systems.

The 5G standard was designed to meet Ultra Reliable Low Latency (URLLC) requirements. While novel applications are progressively leveraging this service category and the existing 5G infrastructure is maturing, the needs for even more real-time communications is slowly arising [3]. The International Telecommunication Union, Radiocommunication Sector (ITU-R) expects Hyper Reliable and Low-Latency Communication (HRLLC) will be one of six usage scenarios for future mobile communications by 2030 [4]. Furthermore, the 6G standard is expected to fulfill communications requirements for immersive applications [2].

The benefits of such immersive applications and their feasibility at the current point in time is already under investigation. Clifford et al. [5] establishes the beneficial effects of immersive perception on situational awareness in firefighting training. A network architecture leveraging multi-link communications to increase reliability and to sustain immersive situational awareness for first responders during missions is demonstrated in [6].

For a timely handling of emergency situations as they arise, the communication infrastructure to be deployed must be compact and flexible. The modularity of the Open RAN approach allows for the flexible integration of commercial-off-the-shelf hardware from different vendors. The deployed networks can be tailored for specific use cases like emergency alerts, as it is possible to reserve resources for public safety on a dedicated network slice [7]. Furthermore, customized resource allocation can be used to significantly reduce latency and enable novel latency sensitive applications, as shown in [1].

III. Experimental System Architecture Core

This section introduces the system architecture for our experimental rescue robotics setup. We detail the common components used across different scenarios, including the *robot fleet*, the *edge cloud server* instances, and the underlying *network environments*. Fig. 2 provides a visual representation of this architecture, while scenario-specific components are introduced in later sections.

A. Robot Fleet & Edge Cloud Servers

Our robot fleet consists of two main units: the *Rescuer*, a Boston Dynamics Spot quadruped robot with a mounted arm for manipulation tasks, and the *Xplorer*, a wheeled Scout-Mini robot from AgileX with a 360°-camera for immersive perception. Both robots are equipped with onboard computers serving as edge servers, providing additional computational power for local, real-time processing directly on the robots.

The edge cloud houses two server instances: a *robot control server* and a *central video server*. The central video server is responsible for receiving the video stream captured by the *Xplorer*. The robot control server runs mission-specific control services and unifies the controls of all robots behind a single Application Programming Interface (API), providing the necessary computing resources for complex decision-making processes.

B. Network Environments

We setup two network environments: a benchmarking network with controlled latency, and a real wireless network leveraging an Open RAN 6G research network that incorporates novel resource allocation schemes.

The **benchmarking network** is designed to systematically observe the behavior of the robot control applications under varying network conditions. To ensure a consistent baseline, we used wired connections between all components and applied artificial network degradation only on selected links. Since we are investigating latency-critical applications, the network degradation we apply is a constant delay. To achieve this, we use vSTING [8], a network emulation tool, to apply a controlled delay to the communications between mission control and the robots. The total experienced delay is the sum of the emulated delay and the negligible real delay from the wired connections.

The Open RAN 6G research network is tailored to address the challenges of latency-critical robotics applications in a **real-world wireless network** environment. Conventional reactive uplink resource scheduling, widely adopted in 5G, introduces significant latency because data transmission is deferred until the base station issues a reactive grant in response to a user's request. To solve this issue, we employ a fully softwarized, end-to-end Open RAN network with co-located edge computing capabilities. This platform enables tight integration of low-latency applications with a real-world communication network in a co-design process. Hence, novel algorithms and approaches for future 6G networks can be developed and experimentally validated under realistic conditions. The deployed network is based on srsRAN Project 25.04 with Open5GS as the core network, using a split 7.2 Benetel RAN550 n78 Open RAN radio unit. Detailed deployment parameters are listed in Table I.

To address the latency issue, we proposed the **proactive** uplink resource scheduling approach, *Open ran Real-time Ai-Coordinated Efficient Scheduling (O-RACES)* [1], which enables prediction-based targeted resource allocation in real-time. O-RACES utilizes the concept of Open RAN dApps, which enable real-time control loops of below 1 ms by running control policy microservices directly on the Distributed Unit (DU) of the Open RAN. Specifically, O-RACES anticipates the exact physical resource block resource demand in real-time on millisecond scales and provides proactive resource allocation towards the user, thereby reducing queuing times. This approach can reduce uplink latency by up to 75%.

TABLE I
PARAMETERS OF THE OPEN RAN 6G RESEARCH NETWORK

Parameter	Value
Frequency Band	n78 (3.7-3.8 GHz)
Bandwidth	100 MHz
MIMO factor	DL: 4x4 MIMO UL: 2x2 MIMO
Radio Unit TX Power	23 dBm
TDD Pattern	DDSUU, Special Slot 6:4:4
Subcarrier Spacing	30 kHz

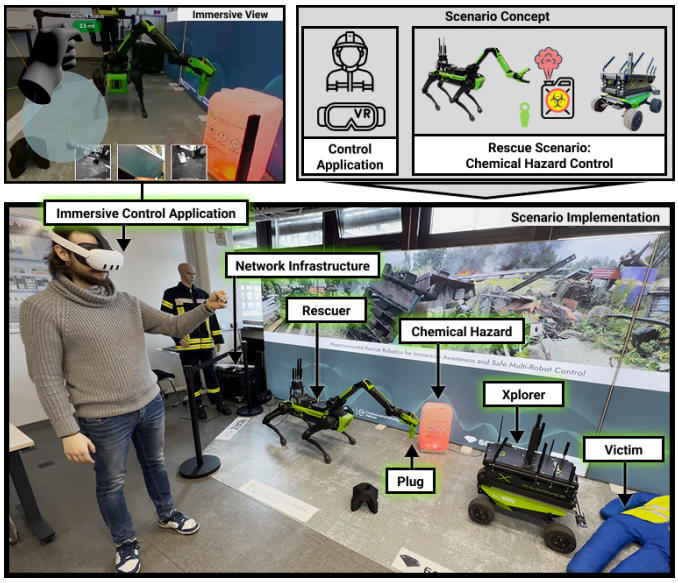

Fig. 3. Annotated experimental setup of the chemical hazard control scenario.

IV. 6G-ENABLED RESCUE ROBOTIC SCENARIOS

This section introduces the two rescue robotics scenarios we formulated: *Chemical Hazard Control* and *Catching the Falling Rod*. For each scenario, we detail the specific hardware setup, operational flow, and the control applications used.

A. Scenario 1: Chemical Hazard Control

The first scenario, *Chemical Hazard Control*, is illustrated in Fig. 3. The scenario includes an open jerrycan releasing toxic gas, a nearby plug to seal the jerrycan, an unconscious person, and our *Rescuer* and *Xplorer* robots. The objective is to contain the chemical hazard by having the *Rescuer* robot pick up the plug and seal the jerrycan through teleoperation. This mission must be completed in the shortest possible time to stop further diffusion of the toxic gas. While there is no fixed deadline, this scenario is still considered latency-sensitive, as network latency directly impacts the user's immersive teleoperation experience.

The user's immersive experience is achieved by a combination of the video stream from the *Xplorer* robot and the video feeds from the *Rescuer* robot. Since the immersive stream from the *Xplorer* robot originates from a single 360°-camera, depth information is supplemented by additional video feeds and physical interaction with the environment. This is especially critical for the delicate maneuvering required to seal the jerrycan. While the *Xplorer* provides the main perception, the *Rescuer* also contributes additional video feeds from its body and arm cameras.

To fully leverage the *Rescuer* robot's immersive insights, an **Mixed Reality (XR)-based teleoperation application** was developed. This application, built using web frameworks that adhere to the WebXR standard, runs directly in a web browser on any WebXR-compatible headset. This implementation en-

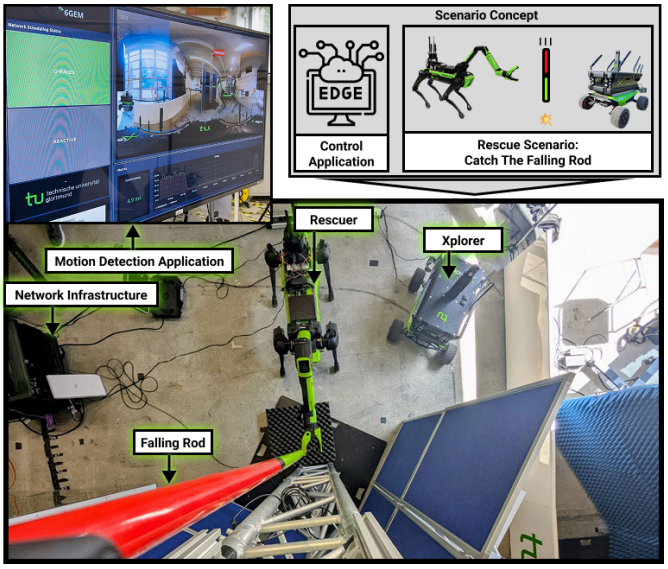

Fig. 4. Annotated experimental setup of the the falling rod scenario scenario.

sures easy deployment and broad compatibility. Hand gestures are used to switch between different camera perspectives and control the *Rescuer* robot's arm movements. Upon receiving control commands via these gestures, they are transmitted to the robot control server over a wired connection to avoid radio interference.

B. Scenario 2: Catching the Falling Rod

The second scenario we consider is a safety-critical application for disaster relief: catching a falling rod to prevent damage or injury. In such a high-stakes scenario, latency is paramount. A strict deadline must be met for the robot's action; failure can lead to significant physical harm or even death.

In addition to the common system components mentioned in Section III, our setup for this scenario includes a custom-built falling rod mechanism for releasing the rod. Figure 4 demonstrates the complete hardware setup for this scenario.

Our software stack for this scenario is built on the *ROS 2 framework* [9]. We employ two key ROS 2 nodes: the *frontend* node and the *Spot Driver* node. The *frontend* node receives command signals from external applications (in our case, a motion detection application). Upon receiving a signal, it triggers the appropriate action of the unofficial Boston Dynamics *Spot Driver* [10] on the *Spot Driver* node via a ROS 2 topic. The *Spot Driver* node, which uses the proprietary Spot SDK, then communicates with the *Rescuer* robot via gRPC [11], initiating the *gripper-close* action. gRPC is an open-source Remote Procedure Call (RPC) framework that allows different applications, often in microservices or mobile-to-backend architectures, to communicate with each other using Protocol Buffers for data serialization and Hypertext Transfer Protocol (HTTP)/2 for transport.

For this scenario, an automated solution was considered with respect to the short reaction required for success. A **motion detection application** was implemented to autonomously detect the fall of the rod through video processing and initiate the catching sequence. In order to freely allocate computing resources for the video processing without space and energy constraints, the motion detection application was deployed as an **edge cloud application**, on the robot control server.

The motion detection application is a multithreaded Python program featuring two primary threads to ensure responsive, real-time operation. One thread is dedicated to continuously fetching video frames from the *Xplorer* robot's stream, while the second thread runs the core detection algorithm simultaneously. This algorithm constantly compares the current frame with a predetermined "target frame" within a user-defined Area-of-Interest (AOI).

We evaluated several algorithms for the detection task. While methods like *pixel-wise Mean Squared Error* (MSE) and *Structural Similarity Index* (SSIM) were computationally fast, we found them to be highly sensitive to reflections and minor lighting changes. This led to false positives. To ensure robustness, we leveraged the *Farnebäck Optical Flow* algorithm [12] for our final implementation. *Optical Flow* effectively measures the apparent motion of objects between frames, making it ideal for reliably detecting the movement of the falling rod. Upon successful detection, the app triggers the catching sequence.

For enhanced usability, the application was built using PySide, a Python binding for the cross-platform Qt GUI toolkit. The interface displays the real-time video stream and allows the user to manually select the AOI with a mouse before the rod-catching sequence begins.

The operational flow of the scenario is as follows: The *Xplorer* robot continuously streams a video feed to the motion detection application running on the robot control server. An external switch is used to initiate the release of rod. The motion detection application, constantly analyzing the video stream, triggers our software stack the moment it detects the fall. This detection initiates the time-critical sequence of commands to the *Rescuer* robot to catch the rod.

V. Scenario Evaluation Results

This section details the evaluations conducted to investigate the impact of latency on our proposed rescue robotics system. In Section V-A, we analyze how different ROS 2 node deployment strategies affect system latency in the *Catching the Falling Rod* scenario. Sections V-B and V-C present the results of user studies conducted to benchmark both rescue robot scenarios. Finally, in Section V-D, we use a real network environment to validate our performance hypotheses regarding Open RAN schedulers, which were formulated from the benchmarking results of the falling rod scenario. Common setup parameters for all evaluations are listed in Table II.

A. Latency-optimized ROS 2 Node Deployment

We first investigate how ROS 2 node deployment impacts the system latency within the *Catching the Falling Rod* scenario. Recall that the scenario utilizes two key ROS 2 nodes:

TABLE II
GENERAL PARAMETERS OF THE EVALUATION SETUP

Parameter	Value
Immersive video stream bandwidth	$\sim$ 50 Mbps
Immersive video framerate	30 fps
Immersive camera latency	230 ms

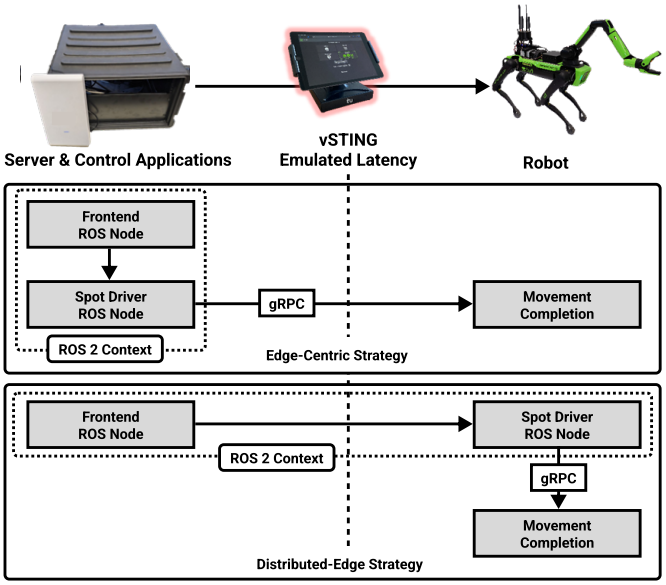

Fig. 5. Overview of the investigated ROS 2 deployment strategies for minimizing end-to-end latency.

the *frontend* node and the *Spot Driver* node. We examine two distinct deployment strategies, *Edge-Centric* and *Distributed-Edge* strategies, to assess their impact on system latency.

In the *Edge-Centric* strategy, both the *frontend* and *Spot Driver* nodes are deployed following the edge cloud approach on the robot control server. In this setup, ROS 2 message communication occurs locally on the server, while the gRPC commands from the *Spot Driver* to the robot are transmitted over the network. In the *Distributed-Edge* strategy, the *frontend* node remains in the robot control server, but the *Spot Driver* node is strategically placed closer to the robot on its mounted computing unit: the *Spot CORE*. This deployment leverages the edge computing capabilities of the *Spot CORE*, allowing the gRPC commands of the underlying SDK to be issued directly from the edge closer to the robot, rather than from the distant edge cloud server. An architectural overview of these strategies is depicted in Figure 5.

We conducted a series of evaluations to measure end-to-end system completion time for specific robot actions under varying levels of network degradation, specifically emulating network latencies of 0, 5, 20, and 50 ms. Our evaluation focused on three key latency metrics:

- *Frontend-to-Driver Latency*: The time from a request's initiation at the *frontend* node to when the *Spot Driver* node receives the corresponding ROS 2 message.

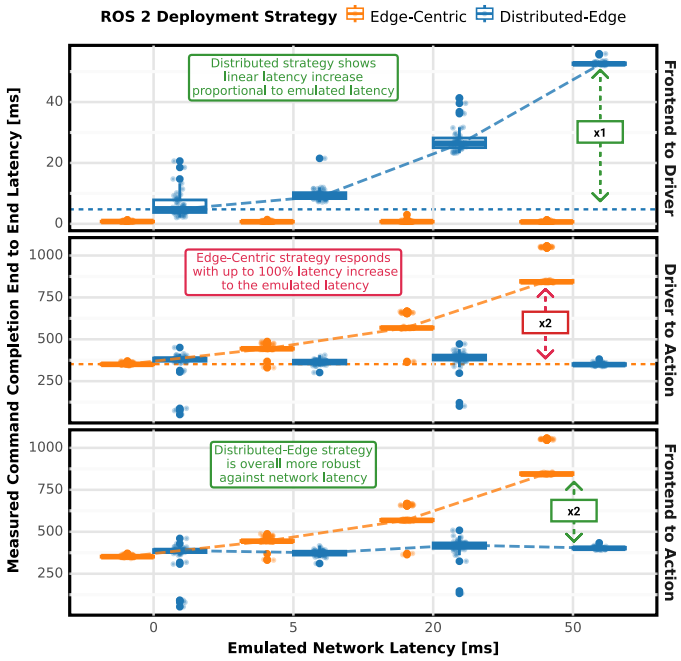

Fig. 6. Latency-based benchmarking results of the ROS 2 deployment strategies for end to end latency of command completion. Please note the different scales of the charts.

- *Driver-to-Action Latency*: The time from the *Spot Driver* node receiving the message to the robot completing the requested action. We treat the robot as a black box and measure the overall time for this action, due to the lack of access to the robot's internal workings. The specific action used is the opening and closing of the robot arm's gripper to a 35-degree opening angle, which corresponds to the angle used in the falling rod experiment.
- *Frontend-to-Action Latency*: The total time from the request's initiation at the *frontend* node to the robot completing the requested action.

TABLE III
ROS 2 LATENCY STATISTICS FOR 5MS DELAY (VALUES IN MS)

Strategy	Metric	Median	Mean	Std Dev
Edge-Centric	Frontend-to-Driver	0.691	0.727	0.16
	Driver-to-Action	443.285	446.174	22.781
	Frontend-to-Action	443.937	446.902	22.792
Distributed-Edge	Frontend-to-Driver	9.052	9.557	2.209
	Driver-to-Action	365.800	363.786	17.074
	Frontend-to-Action	374.493	373.343	17.832

Figure 6 demonstrates significant performance differences between the two deployment strategies. For *Frontend-to-Driver Latency*, the *Edge-Centric* strategy maintained a consistent latency regardless of network degradation, as the ROS 2 communication occurred on the same machine. Conversely, the *Distributed-Edge* strategy exhibited a linear increase in latency proportional to the simulated network delay, as the ROS 2 messages had to traverse the network.

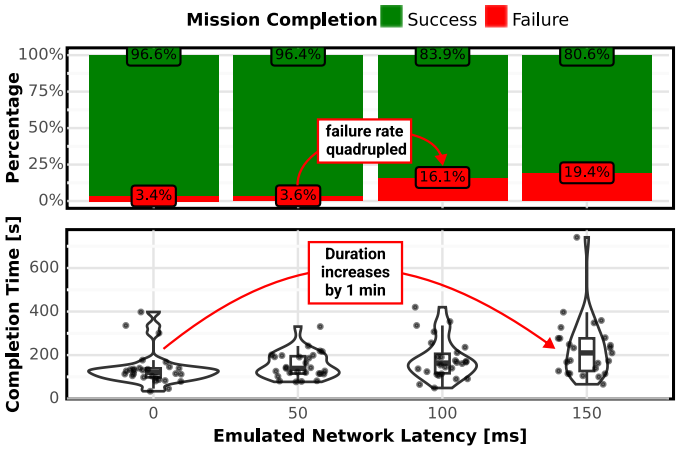

Fig. 7. Benchmarking results of the chemical hazard control scenario with immersive teleoperation under controlled emulated latency.

For *Driver-to-Action Latency*, the *Edge-Centric* strategy experienced a sharp increase in latency with higher network degradation. This is likely due to the nature of gRPC unary calls. A gRPC unary call over HTTP/2 involves multiple distinct communication frames, including the exchange of metadata before and after the actual data is sent [13]. Each of these frames may require a TCP acknowledgment, leading to a compounding increase in round-trip time in high-latency environments. Furthermore, for sporadically used actions, the underlying gRPC channel can grow stale, potentially requiring a costly TLS handshake for renewing authentication. Since the Boston Dynamics Spot's firmware is proprietary, this remains a subject for future investigation with an accessible gRPC pipeline.

In contrast, the *Distributed-Edge* setup's latency for this metric remained relatively low and less affected by the network degradation, as the gRPC communication occurred on a wired connection between the *Spot CORE* and the robot.

All of the described effects accumulate in the *Frontend-to-Action Latency*. As shown in the latency statistics for an emulated 5 ms latency in Table III, the *Distributed-Edge* strategy completes the command about 50 ms earlier.

These findings highlight the critical role of an optimized node deployment strategy for latency-sensitive applications that rely on network protocols, which might exhibit a particular sensitivity to network degradation. Based on our findings, we selected the *Distributed-Edge* approach for our final implementation due to its significantly lower overall latency for movement completion.

B. Benchmarking Results: Chemical Hazard Control

The *Chemical Hazard Control* scenario, described in Section IV-A, was evaluated through a user study with 25 participants. The study parameters are listed in Table IV

The user study was organized to minimize the impact of learning effects. Each participant received a tutorial on the immersive control, followed by a trial run to complete the mission. They then had to complete the mission under four different latency configurations within the benchmarking network environment. The order of these latency configurations was generated randomly to reduce the impact of accumulated experience over the runs on the results. Faulty manipulations which prevent mission continuation, such as dropping the plug out the robot's reach, are registered as failures. In the case of a failure, the participant was allowed one repeat attempt for that specific latency configuration.

TABLE IV
PARAMETERS OF THE CHEMICAL HAZARD CONTROL USER STUDY

Parameter	Value
Participant Count	25
Network Environment	Benchmarking Environment
Network Latencies	0 ms, 50 ms, 100 ms, 150 ms

The mission completion rate and completion times are shown in Figure 7. These results reveal a significant impact of latency on mission performance. A 100 ms increase in latency increases the average mission completion time by nearly a minute and quadruples the mission failure rate. In contrast, an increase of 50 ms shows less impact on mission performance, implying that this level of delay might be acceptable for the chemical hazard control scenario.

This benchmarking evaluation establishes the latency requirements and operational limits for immersive teleoperation of rescue robots. The results suggest that this scenario may already be feasible under ideal conditions within the 5G URLLC service category, though it highlights the importance of minimizing latency for mission success.

C. Benchmarking Results: Catching the Falling Rod

This section details the evaluation of the *Catching the Falling Rod* scenario within the benchmarking environment to investigate its latency requirements. The physical parameters of the setup are listed in Table V.

TABLE V
DIMENSIONS OF THE FALLING ROD SCENARIO

Parameter	Value
Rod Holder Height	270 cm
Robot Arm Height	80 cm
Rod length	75 cm

For a given drop distance z, the required falling time t is determined by $t = \sqrt{2z/g}$, where g is the standard gravitational acceleration (9.81 m/s^2). Based on the physical parameters in Table V, the rod's total fall time is 622 ms. After deducting the latency of the other components in the feedback control loop illustrated in Figure 8, a latency budget of ca. 80 ms remains for wireless communications.

For the benchmarking evaluation, we used latency configurations ranging from 0 to 50 ms in 10 ms steps, with ten runs per configuration. The key metric for performance was the

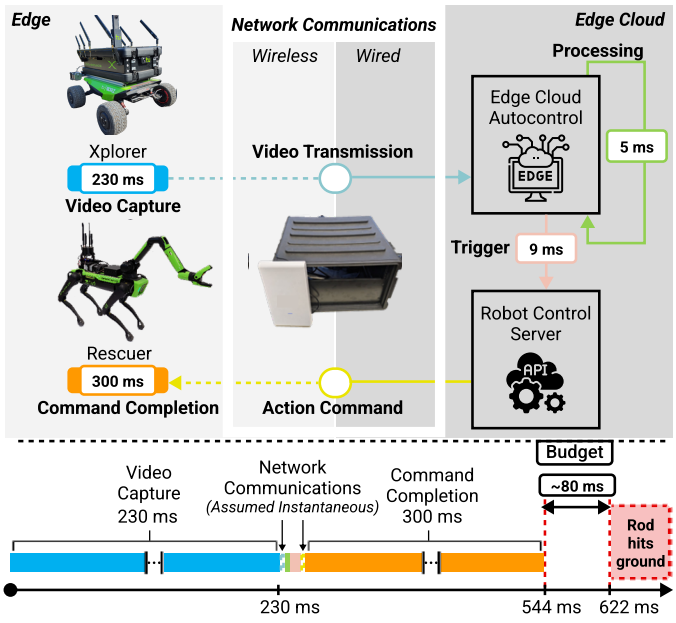

Fig. 8. Overview of the latency contributions of the feedback control loop's components for the *Catching the Falling Rod* scenario. The wireless communication latency is key for the experiment's success.

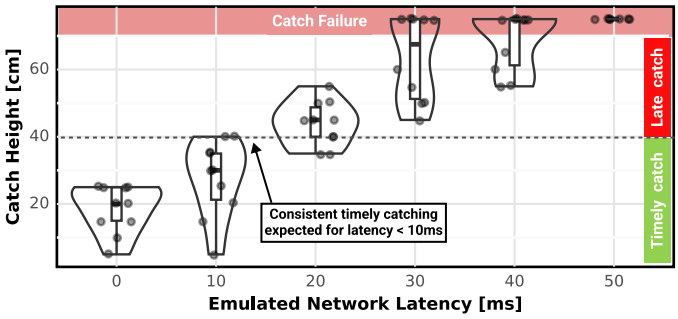

Fig. 9. Benchmarking results of the falling rod scenario with the motion detection edge cloud application under controlled emulated latency.

catch height, measured relative to the rod's bottom. A lower catch height indicates a faster reaction and better performance.

From the results shown in Figure 9, we observe that latencies under 10 ms should allow consistent and timely catches below 40 cm, while having latencies of 20 ms and above might already lead to late catches (above 40 cm) and increased failure rate. This indicates that the catching falling rod scenario might not be reliably feasible within the standard 5G URLLC service category.

D. Catching the Falling Rod over Open RAN

The *Catching the Falling Rod* scenario was further evaluated on the real-world wireless network environment using Open RAN schedulers. Based on the findings from the benchmarking network in Section V-C, we formulated a performance hypothesis regarding the Open RAN schedulers: the reactive scheduler, which typically produces uplink latencies

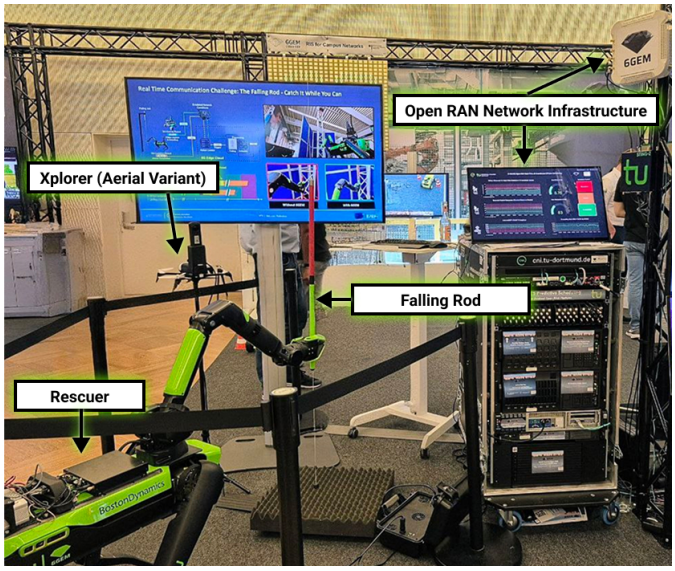

Fig. 10. Experimental falling rod scenario deployed on the Open RAN 6G research network.

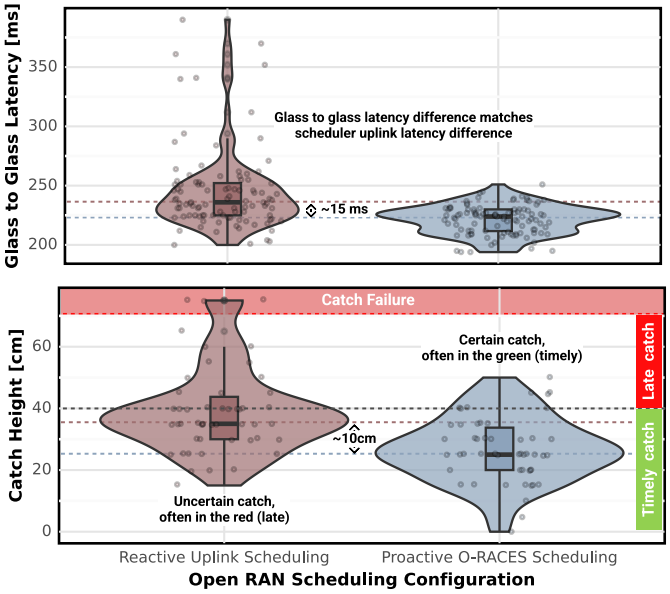

Fig. 11. Performance results of the falling rod scenario with the motion detection edge cloud application over the O-RAN 6G research network.

of approximately 20 ms, will result in later catches and more misses compared to the O-RACES scheduler, which averages a 5 ms latency. An impression of the scenario powered by the Open RAN 6G research network can be found in Figure 10.

To validate this hypothesis, the mission was completed 50 times for each scheduler. In addition to the catch height metric, we also measured the glass-to-glass latency of the video stream used for fall detection. The results are presented in Figure 11, and the catch height statistics with Confidence Interval (CI) are given in Table VI.

TABLE VI
CATCH HEIGHT STATISTICS OF THE FALLING ROD SCENARIO OVER OPEN RAN (VALUES IN CM)

Scheduler	Median	Mean	Std Dev.	95% CI
Reactive Uplink	35	39.10	14.02	[35.11 - 43.09]
Proactive O-RACES	25	26.50	10.51	[23.51 - 29.49]

Our hypothesis was confirmed by the results. The mean latency difference of 15 ms between the two Open RAN schedulers was reflected in the average glass-to-glass latencies of the video stream. The reactive scheduler also exhibited a higher variance in the glass-to-glass latency, which negatively impacted catch consistency. This is confirmed by the catch height results: the median catch height difference was 10 cm, which is slightly larger than the minimal analytical value of 8.2 cm based on the physical parameters and the 15 ms gain of O-RACES. While the reactive uplink scheduler might occasionally catch the rod, the proactive O-RACES scheduler consistently does so, demonstrating a significant increase in reliability.

It's worth noting that some late catches still occurred even with the proactive O-RACES scheduler. This can be explained by the variance of other latency sources in the feedback control loop, specifically the video camera and the robot arm. For instance, the glass-to-glass latency under the proactive O-RACES scheduler shows a variation of 50 ms, and the *Frontend-to-Action* duration, which is the completion time for an arm movement command evaluated in Table III, has a standard deviation of 17 ms for a 5 ms emulated latency. Therefore, even with the low uplink latency provided by the proactive O-RACES scheduler, the latency variation from these other components still strongly influences the final outcome.

VI. CONCLUSION

In this work, we investigate the feasibility of leveraging the edge cloud's computational efficiency for latency-sensitive, mission-critical robotic control applications in disaster relief scenarios. Our evaluation was conducted in two phases: first, we used a benchmarking network environment to determine the latency requirements of our scenarios, and then we evaluated the latency-critical applications on a real-world Open RAN 6G research network.

We first found that a *Distributed-Edge* ROS node deployment strategy consistently yielded a more stable performance compared to an *Edge-Centric* approach. This difference was due to the compounding latency effects of the gRPC protocol over a high-latency network. We then evaluated our *Chemical Hazard Control* scenario with a user study, which confirmed its latency sensitivity. The results showed that a 100 ms latency increase quadrupled the mission failure rate and extended completion time by a minute, demonstrating that while sensitive, the scenario is not strictly latency-critical and can be satisfied with 5G's URLLC. In contrast, our *Catching the Falling Rod* scenario was proven to be latency-critical in the benchmarking environment, requiring a latency of under 10 ms for consistent success. When deployed on the Open RAN 6G research network, only the proactive O-RACES scheduler was able to consistently achieve this, with an average uplink latency of 5 ms.

These results validate the role of 6G research as a key enabler for real-time robotic applications. For future work, a detailed analysis of the gRPC protocol's performance under adversarial network conditions and its behavior under 6G-envisioned HRLLC would be valuable.

ACKNOWLEDGMENT

This work has been funded by the German Federal Ministry of Education and Research (BMBF) via the 6GEM research hub (16KISK038) and was further supported by the project DRZ (Establishment of the German Rescue Robotics Center, 13N16476).

REFERENCES

[1] N. A. Wagner, J. Eßer, I. F. Priyanta, F. Kurtz, M. Roidl, and C. Wietfeld, "Real-time predictive scheduling for networked robot control using digital twins and openran," in *2024 IEEE Globecom Workshops (GC Wkshps)*, 2024, pp. 1–6.

[2] N. H. Mahmood, S. Böcker, I. Moerman, O. A. López, A. Munari, K. Mikhaylov, F. Clazzer, H. Bartz, O.-S. Park, E. Mercier, S. Saidi, D. M. Osorio, R. Jäntti, R. Pragada, E. Annanperä, Y. Ma, C. Wietfeld, M. Andraud, G. Liva, Y. Chen, E. Garro, F. Burkhardt, C.-F. Liu, H. Alves, Y. Sadi, M. Kelanti, J.-B. Doré, E. Kim, J. Shin, G.-Y. Park, S.-K. Kim, C. Yoon, K. Anwar, and P. Seppänen, "Machine type communications: key drivers and enablers towards the 6g era," *EURASIP J. Wirel. Commun. Netw.*, vol. 2021, no. 1, jun 2021.

[3] T. Tao, Y. Wang, D. Li, Y. Wan, P. Baracca, and A. Wang, "6g hyper reliable and low-latency communication – requirement analysis and proof of concept," in *2023 IEEE 98th Vehicular Technology Conference (VTC2023-Fall)*, 2023, pp. 1–5.

[4] ITU-R, *Framework and overall objectives of the future development of IMT for 2030 and beyond*, Jun. 2023.

[5] R. M. S. Clifford, T. McKenzie, S. Lukosch, R. W. Lindeman, and S. Hoermann, "The effects of multi-sensory aerial firefighting training in virtual reality on situational awareness, workload, and presence," in *2020 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, mar 2020, pp. 93–100.

[6] M. Patchou, T. Gebauer, F. Schmickmann, S. Böcker, and C. Wietfeld, "Immersive situational awareness for robotic assistance of first responders enabled by reliable 6G multi-x communications," in *2024 IEEE International Conference on 6G Networking (6GNet)*, Paris, France, Nov. 2024.

[7] IEEE Public Safety Technology Initiative, "Communications and networking technology for public safety: An ieee public safety technology initiative report," IEEE Public Safety Technology Initiative, Tech. Rep., Apr. 2025.

[8] M. Patchou, S. Böcker, and C. Wietfeld, "The radio degradation challenge: Fostering a joint system design of robotics and communication for better application robustness," in *2024 IEEE International Symposium on Safety Security Rescue Robotics (SSRR)*, New York, USA, 2024, pp. 249–254.

[9] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022.

[10] bdaiinstitute, "spot_ros2," Sep. 2025, [Online; accessed 8. Sep. 2025]. [Online]. Available: https://github.com/bdaiinstitute/spot_ros2

[11] "Networking — Spot 5.0.1 documentation," Sep. 2025, [Online; accessed 8. Sep. 2025]. [Online]. Available: https://dev.bostondynamics.com/docs/concepts/networking.html

[12] G. Farnebäck, "Two-frame motion estimation based on polynomial expansion," in *Image Analysis*, J. Bigun and T. Gustavsson, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 363–370.

[13] "Core concepts, architecture and lifecycle," Nov. 2024, [Online; accessed 8. Sep. 2025]. [Online]. Available: https://grpc.io/docs/what-is-grpc/core-concepts/#unary-rpc