COORP: Satisfying Low-Latency and High-Throughput Requirements of Wireless Network for Coordinated Robotic Learning

Shengliang Deng†, Xiuxian Guan, Zekai Sun, Shixiong Zhao†, Tianxiang Shen, Xusheng Chen, Tianyang Duan, Yuexuan Wang, Jia Pan, Yanjun Wu, Libo Zhang, and Heming Cui*, *Member, IEEE*

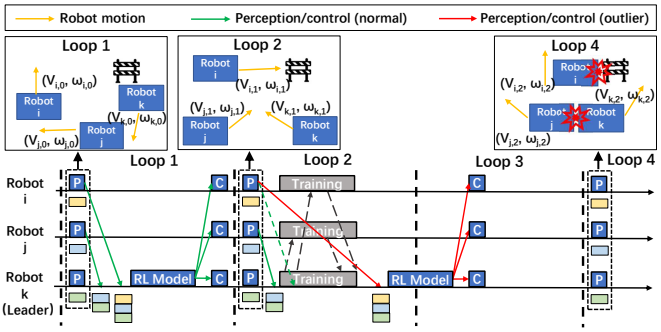
***Abstract*—In coordinated robotic learning, multiple robots share the same wireless channel for communication, and bring together latency-sensitive network flows for control and bandwidth-hungry flows for distributed learning. Unfortunately, existing wireless network supporting systems cannot coordinate these two network flows to meet their own requirements: prioritized contention systems (e.g., EDCA) prevent latency-sensitive messages from timely acquiring the wireless channel because multiple wireless network interface cards (WNICs) with bandwidth-hungry messages are contending for the channel; global planning systems (e.g., SchedWiFi) have to reserve a notable time window in the shared channel for each latency-sensitive flow, suffering from severe bandwidth degradation (up to 42%).**

We present the coordinated preemption method to meet both requirements for latency-sensitive flows and bandwidth-hungry flows. Globally (among multiple robots), coordinated preemption reduces the number of bandwidth-hungry flows to ensure that all latency-sensitive flows have the highest chance to win the contention against bandwidth-hungry flows, while such reduction does not affect the overall bandwidth through the lens of upper applications. Locally (within the same robot), coordinated preemption in real-time predicts the periodic transmission of latency-sensitive flows from the upper application and conservatively limits packets of bandwidth-hungry flows buffered in the WNIC only before latency-sensitive packets arriving, reducing the bandwidth degradation devoted to preemption. COORP, our implementation of coordinated preemption, reduced the violation of reaction time boundary from 53.9% (EDCA) to 8.8% (comparable to SchedWiFi). Regarding learning quality, COORP achieved a comparable (at times the same) learning reward with EDCA, which grew up to 76% faster than SchedWiFi.

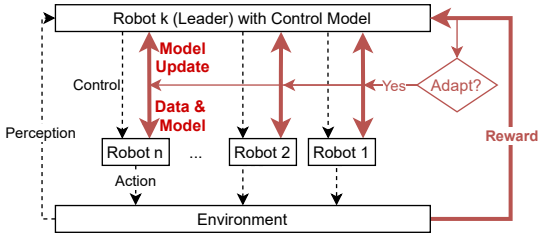
***Index Terms*—Cyber-Physical Systems, Efficient Communications and Networking, Machine-to-Machine Communications, Real-Time Systems**

I. INTRODUCTION

Coordinated robotic learning enables a group of robots to closely coordinate with each other and adapt to new environments, and is increasingly important to mission-critical multi-robot applications in field, including rescue [1], navigation [2], and surveillance [3], [4]. Despite each robot's local functionalities (e.g., collision avoidance) and traditional latency-sensitive communications (e.g., cooperative localization [5]), we summarize that a typical coordinated robotic learning workflow



(a) Workflow of MPC and why fast reaction is important. 'P' stands for 'perception'. 'C' stands for 'control'. DRLC is simplified to emphasize its network usage. Bandwidth-hungry flows for training block latency-sensitive flow (red line) and cause violation of the reaction time boundary (loop 2), leading to lag and collision (loop 4).



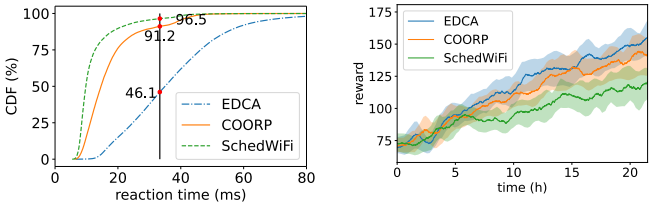
(b) Workflow of DRLC.

Fig. 1: Workflow of MPC and DRLC, and why fast reaction is important.

comprises two global coordinations: a *multi-view perception-control coordination* (MPC) that continuously exchanges information among robots and makes coordinated decisions, and a *distributed reinforcement learning coordination* (DRLC) that exploits the distributed computation power of all robots to adapt to new environments.

The MPC of a robot group loops over *perception exchange*, *inference*, and *control dissemination*, as shown in Figure 1a. During *perception exchange*, a leader robot collects and combines perceptions (e.g., images) captured by the sensors (e.g., cameras) of all the robots via wireless network for multi-view awareness of the environment. During *inference*, the leader robot processes and feeds the combined perception to a deep neural network control model which takes environmental information as input and produces control messages (e.g., velocities) as output. After that, the leader robot *disseminates* the control messages to each robot for the next move.

*Corresponding author.
†Student Member, IEEE.
Email: sldeng@cs.hku.hk, guanxiux@hku.hk, {zksun, sxzhao, txshen2, xschen}@cs.hku.hk, tianyang.duan@ucdconnect.ie, {amywang, jpan}@cs.hku.hk, {yanjun, libo}@iscas.ac.cn, heming@cs.hku.hk



(a) Reaction time. (b) Convergence over time.

Fig. 2: Comparison of different systems. COORP achieved a low percentage of violation of the reaction time boundary (vertical black line) while converging almost as fast as EDCA.

The DRLC is usually spawned when a robot group enters a new environment and would last for hours [6], [7]. As the hardware resources per robot are limited, data parallelism (DP, see §II-A) is usually adopted to exploit the distributed computation power of the robots [8]. A data parallel DRLC consists of *data dissemination*, *parallel training*, and *parameter synchronization*, as shown in Figure 1b. Note that, in each iteration of the MPC loop, the leader robot collects a composed data item of a combined perception, control messages that contain actions, and a corresponding reward that reflects the effect of the actions. When a batch of data is accumulated, the leader robot partitions the data batch and *disseminates* each robot a partition. Then each robot trains the control model *in parallel*. The leader robot gathers the parameter updates computed by each robot to *synchronize* the control model. DRLC runs until the robot group adapts to the new environment. In this paper, we focus on coordinated robotic learning with one MPC and one DRLC.

We identify two stringent requirements for mission-critical coordinated robotic learning applications. First, the MPC loop requires *fast reaction* (**R1**): the reaction time (i.e., the time spent in each iteration of MPC loop) should be within a tight boundary to maintain a high frequency (e.g., 33ms boundary for 30Hz [9]) and avoid severe flaws like robot collisions demonstrated in Figure 1a. Second, the DRLC requires *fast model convergence* (**R2**): the control model should adapt to a new environment (i.e., converge) as fast as possible.

However, the two coordinations (MPC and DRLC) generate network traffic flows with different characteristics, making it cumbersome for the underlying network supporting systems to meet these two stringent requirements simultaneously. In terms of traffic volume, the messages of MPC (i.e., perception and control messages) are often in small scale (e.g., several KB); while the messages of DRLC (i.e., data dissemination and parameter updates) are often in much larger scale (e.g., tens to hundreds of MB). Still, in terms of traffic pattern, the messages of MPC are generated periodically (e.g., 30Hz) and are latency-sensitive: a high latency would result in violation of reaction time boundary and cause flaws (i.e., violate **R1**); while the messages of DRLC are generated less frequently but are bandwidth-hungry: a limited available bandwidth will slow down the speed that a distributed learned control model adapts to a new environment (i.e., violate **R2**).

Unfortunately, no existing wireless network supporting system meets **R1** and **R2** simultaneously when serving these two aforementioned traffic flows in coordinated robotic learning. In general, existing systems [10]–[20] can be classified into two categories, we discuss each individual system in §II-B. The first category is *prioritized contention* systems: each wireless network interface card (WNIC) on each robot occupies the shared wireless channel on demand; WNICs that have high priority packets are given higher chances to occupy the channel than other WNICs. Prioritized contention enables robots to communicate without central control and achieves high bandwidth utilization. However, it fails to achieve **R1** in coordinated robotic learning because multiple WNICs with bandwidth-hungry messages to transmit contend for the channel simultaneously, lower the chance for WNICs with latency-sensitive messages to timely acquire the channel, and still cause latency increase.

The second category is *global planning* systems: a global scheduler divides the time into time windows and arranges the transmission of latency-sensitive messages and bandwidth-hungry messages into exclusive time windows. Global planning is suitable for cases that a limited available bandwidth does not affect the service quality of the application.

However, in dividing and assigning the time window, global planning has to consider uncertainties from two sources: first, the packet transmission time is not stable [21]; second, dividing a time axis across multiple devices requires a global clock but faces clock skewness [15]. In the presence of these uncertainties, global planning faces a paradox to meet both **R1** and **R2** in coordinated robotic learning: on the one hand, as the scheduler cannot collect the uncertainty on each robot in real-time, it has to reserve a large enough time window for each latency-sensitive message transmission, taking up too much of the bandwidth and violating **R2**; on the other hand, reducing the size of each time window to meet **R2** would make latency-sensitive transmissions more likely to miss their own windows, violating **R1**.

For instance, we ran a multi-robot navigation coordinated robotic learning application [2] on five robots with SchedWiFi [17], the most relevant global planning system that is designed for low latency of control messages in industrial automation. Despite a low percentage (3.5%) of violation of the reaction time boundary (Figure 2a), SchedWiFi suffered a 49% slowdown of the training process (Figure 2b) due to a 42% reduction of available bandwidth (§VI-A Table II).

Overall, a system meeting both **R1** and **R2** simultaneously for coordinated robotic learning is highly desired but missing. In this paper, we present COORP, the first network supporting system for cooperating the two types of coordinations (MPC and DRLC) and meeting both **R1** and **R2** for coordinated robotic learning. Lying in the core of COORP is our new method: *coordinated preemption*. Coordinated preemption should ensure that whenever a latency-sensitive message from any of the robots is to be transmitted: locally, the WNIC on the robot should immediately serve the latency-sensitive message for transmission (*local preemption*); globally, the WNIC which serves the latency-sensitive message should immediately acquire the wireless channel shared by all the WNICs of the robot group (*global preemption*). COORP meets

R1 by enforcing the coordinated preemption. Still, COORP maintains a reasonably high available bandwidth to meet **R2**.

Enforcing *global preemption* is challenging: as multiple WNICs are contending the same wireless channel, when a latency-sensitive message is to be transmitted by a WNIC, the wireless channel might already be contended by multiple WNICs that are serving a high volume traffic; this lowers the chance that the WNIC with a latency-sensitive message acquires the channel even with EDCA [14], a technique since WiFi 4 that makes a contention incline to transmissions assigned with higher priority; thus, the latency-sensitive message is blocked (violate **R1**). To ensure global preemption for coordinated robotic learning, we leverage a key observation that, in the *data dissemination* and *parameter synchronization* of the DRLC, the critical path is dominated by the transmission time of all the bandwidth-hungry messages via a single wireless channel. Leveraging this observation, COORP mitigates the contention by only allowing a constant and small number of bandwidth-hungry flows to be transmitted simultaneously. By doing so, at any time, there is only a constant number of WNICs with bandwidth-hungry messages contending for the channel regardless of the number of robots, and when any latency-sensitive message is to be transmitted, it can easily win the contention with a high chance (meet **R1**). Moreover, COORP lets bandwidth-hungry messages saturate the wireless channel in a round-robin manner for a high overall bandwidth utilization (meet **R2**).

Enforcing *local preemption* on commodity wireless network interface controllers (WNICs) is confronted with a unique challenge that, in modern WiFi protocol [22], when a large number of packets are being served, newly arrived packets have to wait until the WNIC finishes the transmission of a number of previous packets due to frame aggregation [23], a key technique in modern WiFi for high bandwidth. Our key observation for this challenge is that, in the MPC of coordinated robotic learning, the latency-sensitive messages are often generated at a constant frequency which is predictable by COORP. Leveraging this observation, we propose a novel virtual-preemption layer that predicts when a latency-sensitive message will arrive and limits the number of packets of bandwidth-hungry messages the OS passes to the WNIC, such that when a latency-sensitive message arrives, previous packets buffered by the WNIC have been finished. By doing so, a latency-sensitive message can be served immediately by the WNIC whenever it is generated (meet **R1**). COORP dynamically estimates the maximum number of packets that could be transmitted before the arrival of the next latency-sensitive message to meet **R2**. The virtual-preemption layer leverages common operations of the WNIC driver, thus it is portable to different commodity WNICs.

We implemented COORP with around 1000 lines of code in Linux 5.4.84 and ROS2. We compared COORP with two representative methods, EDCA [14] and SchedWiFi [17], on a typical multi-robot navigation application [2]. Evaluation shows that:

- COORP achieves fast model convergence. COORP achieved a comparable (at times the same) learning reward with EDCA, which grew up to 76% faster than SchedWiFi due to 1.45x more available bandwidth.
- COORP achieves fast reaction. COORP reduced the violation of reaction time boundary from 53.9% (EDCA) to 8.8% (comparable to SchedWiFi).
- COORP is easy to use. It took moderate effort to integrate COORP with the above-mentioned multi-robot navigation application.

Our major contribution is the new *coordinated preemption* method, the first work fulfilling both *fast reaction* (**R1**) and *fast model convergence* (**R2**) for mission-critical coordinated robotic learning applications consisting of both latency-sensitive flows and bandwidth-hungry flows. Our analytical analysis and experimental results show that, coordinated preemption greatly reduces violation of timing constraints with little bandwidth degradation, while having good scalability in terms of the number of robots. This makes COORP the first unique platform to support diverse coordinated robotic learning applications consisting of both latency-sensitive flows and bandwidth-hungry flows, including robot soccer [24], multi-view object tracking [25], and large-scale disaster response [26]. COORP's code is released on github.com/hku-systems/Coorp.

In the rest of this paper: §II discusses related work; §III presents the system model and overview of COORP; §IV describes the design of COORP; §V describes the implementation; §VI shows our evaluation; §VII concludes.

II. BACKGROUND AND MOTIVATION

A. Coordinated Robotic Learning

Multi-view Perception-Control Coordination. Traditional robot control algorithms mainly leverage local perceptions collected with sensors on each robot [27], [28]. While these algorithms perform well in single-robot tasks, when multiple robots work together, they are unable to leverage perceptions of other robots (e.g., image of an obstacle from different angles) and may lead to suboptimal decisions.

Increasing research efforts [2], [4], [29] in multi-robot applications have led to a multi-view paradigm: multiple robots in a group combine their perceptions of the environment and make coordinated decisions based on the combined perception. For example, in [2], the authors proposed a multi-robot navigation algorithm that combines perceptions of all the robots to get the state of all the obstacles and generates coordinated actions. Without the multi-view paradigm, each robot would only know the position of its nearby obstacles and take suboptimal actions. In real-world deployments, multi-view algorithms may coexist with traditional single-view algorithms for the flexibility of the entire robot group [27].

Distributed Reinforcement Learning Coordination. Deep reinforcement learning trains a deep neural network control model by interacting with the environment, accumulating experiences, and training the control model with the experiences, and is especially effective for robot control involving interaction with complex environments [30], [31]. Despite its flexibility and learning capability, even a well-trained control

model requires fine-tuning in new environments [32], [33], which is computation-intensive and memory-consuming.

Data parallelism is widely adopted in training deep neural networks to alleviate the computation and memory requirements on each single worker. In data parallelism, the training data is partitioned and scattered to multiple workers. These workers train the same control model using their own partition of the data in parallel and synchronize the model parameters with others. There are different paradigms for synchronizing the parameters across workers [34], namely Bulk Synchronous Parallel (BSP), Asynchronous Parallel (ASP), and Stale Synchronous Parallel (SSP). These paradigms differ in when to synchronize the parameters, but they all transmit training data and parameter updates over the network and incur large volume traffic.

MPC and DRLC often colocate in real-world deployments. It is widely reported [35], [36] that models trained in simulated or known environments often have unsatisfactory performance in real world or unknown environments. Therefore, it is a trending practice [37], [38] to fine-tune pre-trained models (DRLC) with real-world newly collected data while the robots are conducting real-time actions (MPC), so that they can adapt to new data samples never encountered before. For example, in [37], the authors proposed an improved training strategy to fine-tune a deep reinforcement learning based navigation model during deployment in real environments.

B. Related Work

The widely deployed WiFi usually uses distributed contention to decide which WNIC transmits its packets over the shared wireless channel: each time a WNIC wants to transmit, it first sets a counter to a randomly chosen number between 0 and a pre-defined parameter CW; then it decrements the counter when the channel is not used by others until zero before it actually transmits. This requires little coordination among WNICs and is easy to setup, thus it is widely adopted. However, the distributed contention is unable to provide low transmission latency for latency-sensitive traffic. Existing work can be classified into two categories: *prioritized contention* and *global planning*.

Prioritized Contention. This category improves the contention process such that a WNIC with latency-sensitive packets to transmit has a higher chance to win in the contention. QAir [10] dynamically limits the queue length on each host such that latency-sensitive packets can bypass most of the packets to be transmitted, but the latency-sensitive packets still have to wait for packets that are already passed to the WNIC and also contend with transmission from other WNICs. Idle Sense [12] and SynchMAC [11] improve the fairness of contention, but does not optimize for the coexistence of multiple latency-sensitive flows and bandwidth-hungry flows, and requires modification of WiFi protocol. EDCA [14], a part of modern WiFi protocol, sets up four traffic categories (`AC_VI`, `AC_VO`, `AC_BE`, and `AC_BK`), and assigns different initial CWs for these traffic categories such that higher priority traffic can gain earlier access to the shared wireless channel. While these systems are effective in daily use cases, they are not suitable for coordinated robotic learning, since multiple robots have bandwidth-hungry flows to transmit simultaneously, each trying to saturate the limited bandwidth, causing intensive contention and significantly lowering the chance for latency-sensitive packets to be transmitted timely.

Global Planning. This category leverages a global scheduler to divide the time into time windows and arrange the transmission of latency-sensitive messages and bandwidth-hungry messages into exclusive time windows. A majority of work [13], [16]–[20] proactively assigns time windows to different wireless stations. While the contention is fully avoided, they lack flexibility to accommodate dynamically generated traffic like those sent by TCP. Also, they require all the wireless stations to have a globally synchronized clock, incurring additional synchronization overhead and are vulnerable to clock skewness [15]. OpenTDMF [15] alleviates the need for a global clock by polling each wireless station to transmit. It achieves high efficiency and fairness across wireless stations. However, the global scheduler needs to know when a latency-sensitive message is generated to timely poll the corresponding wireless stations.

C. Motivation

In this work, we propose a new method named *coordinated preemption* consisting of *local preemption* and *global preemption*.

Global Preemption. Global preemption should ensure that, whenever a WNIC has latency-sensitive messages to transmit, it immediately acquires the wireless channel shared by all the WNICs of the robot group. We observe that, in the data dissemination and parameter synchronization of the DRLC, the critical path is dominated by the transmission time of all the bandwidth-hungry messages for synchronization via a single wireless channel. Meanwhile, even a single bandwidth-hungry flow is able to saturate the available bandwidth due to its large traffic volume and the limited network capacity. Thus, it is viable to reduce the contention by only allowing a constant and small number of bandwidth-hungry flows to be transmitted at any time, and leverage EDCA to prioritize the transmission of latency-sensitive flows. In this way, bandwidth-hungry flows keep saturating the wireless channel in a round-robin fashion to achieve a high bandwidth utilization (benefitting **R2**), and latency-sensitive flows only need to contend with a small number of bandwidth-hungry flows at any time (benefitting **R1**).

Local Preemption. Local preemption should ensure that, on each robot, whenever a latency-sensitive message is to be transmitted, the WNIC on the robot immediately serves the latency-sensitive message for transmission. Although this was well-studied as a problem of the OS network stack [39], a pre-condition for existing work to be effective under large traffic volume no longer holds on modern commodity WNICs: *the OS network stack must be the last component where the latency-sensitive message can be blocked*. Modern commodity WNICs work asynchronously with the OS to achieve high throughput (i.e., bytes transmitted per second) with frame aggregation [23]. Packets delivered to the WNIC accumulate in

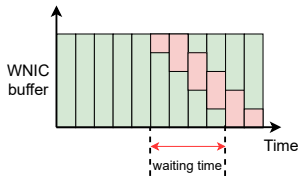


Fig. 3: Conceptual depiction of congestion in the WNIC. Packets of bandwidth-hungry flows (green) delivered by the OS accumulate in the WNIC buffer and keep the WNIC busy transmitting. Packets of latency-sensitive flows (red) have to wait until transmission of previous packets is finished.

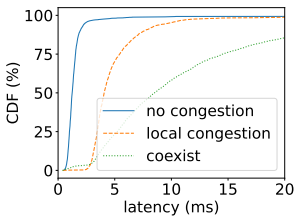


Fig. 4: Local congestion and its exacerbation. In 'coexist', local congestion is exacerbated by the coexistence of a bandwidth-hungry flow from another robot. Both local congestion and its exacerbation cause a significant latency increase.

the WNIC's internal buffer and keep the WNIC busy transmitting. Newly delivered packets have to wait until the ongoing transmission of previous packets is finished, as conceptually depicted in Figure 3. We term this as *local congestion*. Local congestion exists even if the WNIC has multiple hardware queues, because the hardware components for transmission are saturated by bandwidth-hungry flows. Further, when there are other robots transmitting bandwidth-hungry flows, it takes more time to finish previously buffered packets, making local congestion exacerbated.

Figure 4 demonstrates local congestion and its exacerbation. The experiment was carried out with the same testbed as described in §VI, and FQ-CoDel [40], a queue management algorithm that optimizes for latency-sensitive sparse traffic, was adopted by default such that the latency-sensitive messages were not blocked in the OS network stack. The latency-sensitive flow was configured to transmit with the highest priority (`AC_VO`), and the bandwidth-hungry flow was configured to transmit with the default priority (`AC_BE`).

In COORP, local preemption is achieved with a virtual-preemption layer which enables latency-sensitive flows to virtually preempt local bandwidth-hungry flows to avoid the increase of latency. This layer leverages common driver operations (see §V) and requires software-only modifications to the OS, thus it supports different commodity WNICs.

COORP has a similar purpose with Frame Preemption (IEEE 802.1Qbu [41]) which was originally introduced in Ethernet and interrupts both local and remote ongoing transmissions to handle latency-sensitive packets. However, Frame Preemption for WiFi is still in early discussion [42], facing open research obstacles including needing an additional channel with special access control [43] and backward incompatibility with existing WiFi devices [43]. Different from Frame Preemption's medium access control (MAC) layer solution which requires an extensive and expensive hardware upgrade, COORP leverages the feature of existing commodity WNICs, providing low latency and high bandwidth utilization for coordinated robotic learning applications at the software level. We believe COORP is complimentary to Frame Preemption, and COORP's software-only and easy-to-deploy techniques can greatly promote the deployment of coordinated robotic learning applications.

III. SYSTEM OVERVIEW

A. System Model

In the deployment model of COORP, two coordinations, i.e., MPC and DRLC as described in §I, are deployed in a robot group: the MPC makes coordinated decisions for the robot group, and the DRLC adapts the robot group to new environments. Each robot is equipped with sensors (e.g., cameras) that generate perceptions. A leader robot gathers the perceptions of all the robots and holds a deep neural network as the control model to make coordinated decisions. The control model is pre-trained and needs fine-tuning in a new environment for better performance. The other robots (worker robots) are within one-hop distance from the leader robot so that the communication between the worker robots and the leader robot does not need any relay. All robots communicate via a wireless network in a shared channel. RTS/CTS [44] is enabled in the wireless network to avoid interference caused by hidden terminal problems [45]: multiple worker robots transmit packets simultaneously and interfere with each other. Also, we assume all robots have the same number of antennas and SU-MIMO is enabled by default which utilizes multiple antennas to improve the bandwidth between two robots.

In the MPC, for the multi-view coordination, the perceptions are generated from the sensors on each robot periodically. Time is divided according to the generation period (e.g., 33ms) of perceptions. Each period is expected to contain exactly one MPC loop. A reaction time boundary no longer than the generation period is configured, and MPC loops are expected to finish within the reaction time boundary. An MPC loop initiated in a period can lag and finish in the next period or later as shown in loop 3 in Figure 1a, and put off subsequent MPC loops.

In the DRLC, the leader robot collects a training data item (i.e., perceptions, control messages, and rewards) per MPC loop. The rewards are computed according to subsequent perceptions. When a certain number of training data items are collected, the leader robot disseminates partitions of training data among robots and continues the concurrent MPC loop without collecting training data, because these data items are generated under the old control model. When parameter synchronization finishes and the control model is updated, the leader robot resumes collecting new training data under the updated control model.

B. Overview of COORP

Figure 5 shows the architecture of COORP. COORP fulfills the two requirements of coordinated robotic learning, i.e., *fast reaction* (**R1**) and *fast model convergence* (**R2**), by enforcing coordinated preemption while maintaining a high available bandwidth. The BH controller and global controller enforce global preemption, while the preemption controller and the virtual-preemption layer enforce local preemption. Overall,

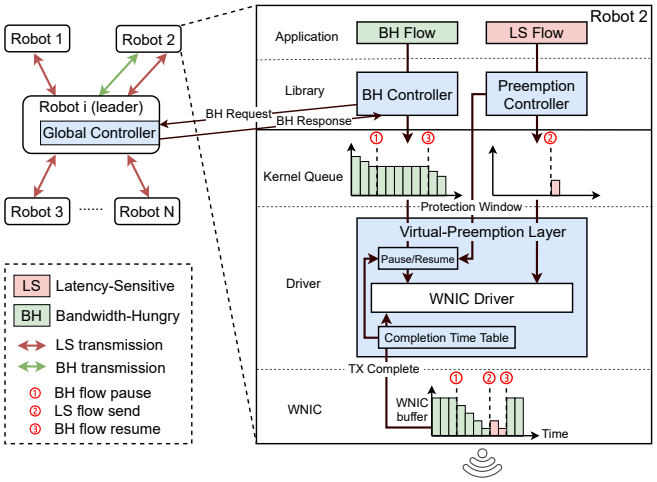


Fig. 5: The architecture of COORP. COORP's components are shaded in blue. Latency-Sensitive (LS) flows are for perceptions and control messages. Bandwidth-Hungry (BH) flows are for parameter synchronization. ① The virtual-preemption layer pauses the kernel queue of the BH flow to avoid buffering too many packets in the WNIC; the WNIC still transmits the packets it has buffered. ② The application sends an LS message; the WNIC serves the LS message immediately because it has finished previously buffered packets. ③ The virtual-preemption layer resumes the kernel queue of the BH flow because the transmission of the LS message is finished. Meanwhile, the global controller and BH controllers ensure there are only a small number (one in the figure) of BH flows being transmitted in the robot group at any time.

COORP adopts a cross-layer design leveraging information from both application and WNIC buffer to effectively schedule latency-sensitive flows and bandwidth-hungry flows. Such scheduling cannot be achieved in any of the individual layers due to the unawareness of wireless channel usage in the application layer and the unawareness of traffic patterns in the WNIC driver layer.

a) ***Enforcing Global Preemption****:* As discussed in §II, the concurrently generated bandwidth-hungry flows from different robots in the DRLC contend for the channel intensively and increase the latency of latency-sensitive flows, violating **R1**. We observe that reducing contention intensity is a promising way to enforce **R1** and **R2** simultaneously among flows from different robots. A strawman approach is to leverage a time division multiple access (TDMA) method: dividing time into time slices and exclusively assigning time slices to each bandwidth-hungry flow. However, the TDMA method is unsuitable for coordinated robotic learning because different bandwidth-hungry flows in the learning process can start and finish transmission at different times due to the variation of training speed and network condition. Once a certain bandwidth-hungry flow finishes transmitting, the future time slice assigned to it will be wasted (violating **R2**).

Instead, COORP adopts a request-based semi-TDMA method. When a robot attempts to transmit bandwidth-hungry flows, its BH controller will intercept the transmission and request the global controller for permission to transmit. If the number of transmitting bandwidth-hungry flow is below a preconfigured limit (1 by default for lowest contention), the global controller will grant the transmission with a response immediately. Otherwise, the global controller will postpone the response until some transmitting bandwidth-hungry flow finishes. This process only needs a request message and a response message (several bytes), which consumes little of the bandwidth compared to the large size of bandwidth-hungry messages (tens to hundreds of MB), guaranteeing **R2**. Moreover, limiting the number of simultaneously transmitting bandwidth-hungry flows would not lower bandwidth utilization because even a single bandwidth-hungry flow is able to saturate the bandwidth due to its large traffic volume.

With EDCA (configuring latency-sensitive flows with the highest priority) and the limited number (e.g., 1 or 2) of simultaneously transmitting bandwidth-hungry flows (achieved with global controller and BH controller), latency-sensitive flows get high probability (§IV-D) of winning the contention with bandwidth-hungry flows, achieving global preemption.

b) ***Enforcing Local Preemption****:* Due to the closed-source design of commodity WNICs, the transmission on the WNIC cannot be manipulated directly. To enforce local preemption, we design and implement a software-only virtual-preemption layer surrounding the WNIC driver between the WNIC and upper kernel layers to virtually enforce preemption for latency-sensitive flows. The virtual-preemption layer relies on a preemption controller to anticipate the arrival time of latency-sensitive packets (i.e., the time when a latency-sensitive flow would send a message). The virtual-preemption layer records the time that bandwidth-hungry packets take to be transmitted by the WNIC (transmission completion time) and further estimates the future transmission completion time of new bandwidth-hungry packets to be fed to the WNIC. If the transmission completion time of a bandwidth-hungry packet would overlap with the arrival time of latency-sensitive packets, the virtual-preemption layer will stop feeding the bandwidth-hungry packet to the WNIC by pausing the kernel queue until the latency-sensitive packets are sent out. In this way, preemption on the WNIC is virtually achieved and latency-sensitive packets would not be blocked by the bandwidth-hungry packets on the WNIC, minimizing the transmission latency (**R1**). Moreover, the transmission completion time is collected in real-time from the driver locally, which is close to the WNIC. Thus the time to pause kernel queue of bandwidth-hungry flows is accurate (benefitting **R1**) and minimal (benefitting **R2**).

IV. DETAILED DESIGN

A. Global Controller and BH Controller

In COORP, a request-based semi-TDMA method as shown in Algorithm 1 is integrated with EDCA enabled, so that contention intensity among bandwidth-hungry flows is reduced and latency-sensitive flows can acquire the channel with the highest probability when contending with bandwidth-hungry flows. The global controller on the leader robot maintains a queue Q of robots that requested to transmit bandwidth-hungry

Algorithm 1: Request-Based Semi-TDMA

```
1  Function schedule(): // On global controller
       Data: Q: queue of robots requesting for
             transmission; S: set of robots transmitting
2      upon get request from robot r do
3          Q.put(r);
4      upon robot r in S for more than TIMESLICE
5            || get release from robot r do
6          S.remove_if_exist(r);
7      upon r ← Q.get() do
8          wait until |S| < LIMIT;
9          send permit to r;
10         S.add(r);
11 Function send(data): // On BH controller
12     while data not all sent do
13         request_wait();
14         transmit data for no more than TIMESLICE;
15         release();
16     end
```

flows, and a set S of robots that are transmitting bandwidth-hungry flows simultaneously.

Function `send()` is called each time a worker robot transmits a bandwidth-hungry flow. The BH controller on this robot sends a request to the global controller on the leader robot and waits for a response (line 13) before transmission. The global controller allows the requested transmission by sending a response when the number of simultaneously transmitting bandwidth-hungry flows in S has not reached a limit (1 in this paper), so that EDCA can effectively prioritize the transmission of flows according to their latency requirements (**R1**). If the limit is reached, the global controller adds the request to Q and postpones the response until the number of simultaneously transmitting bandwidth-hungry flows decreases (line 8). On receiving the response, the robot keeps transmitting bandwidth-hungry flows for a preconfigured time slice (e.g., 5s), or if the robot finishes transmitting bandwidth-hungry flows within the preconfigured time, it will notify the global controller to recycle the unused time for other bandwidth-hungry flows.

Function `schedule()` runs continuously on the leader robot. Each time a robot is permitted to transmit bandwidth-hungry flows, the robot is added to set S (line 10). If a robot has been transmitting for the preconfigured time, or a robot notifies that it finishes transmitting bandwidth-hungry flows (release), the robot is removed from S (line 6).

B. Preemption Controller

Each time a latency-sensitive flow sends a message, the preemption controller records the time point as a sample. Suppose a latency-sensitive flow started from time q and its period is p, the ideal arrival time (i.e., when it is generated by the application) of the kth ($k = 0, 1, 2, ...$) message can be modeled as $\hat{T}_k = pk+q$. However, due to random fluctuations, there is a difference between the actual time T_k and the ideal time $\hat{T}_k$. As shown in Figure 6, the fluctuation roughly follows

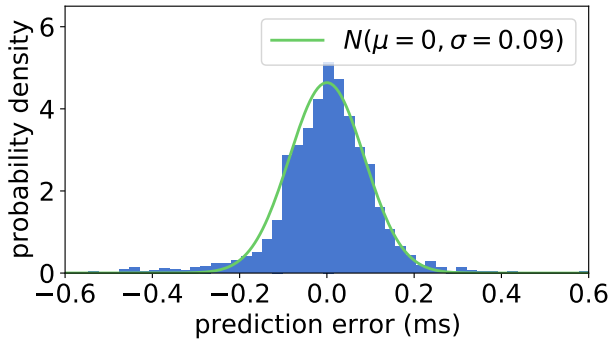


Fig. 6: The prediction error roughly follows normal distribution.

a normal distribution $N(0, \sigma^2)$: $T_k \sim N(pk + q, \sigma^2), k = 0, 1, 2...$ The parameters p, q and σ constitute a time model $\{p, q, \sigma\}$ of a latency-sensitive flow.

With enough samples of a latency-sensitive flow, the preemption controller runs linear regression on these samples to estimate the corresponding time model of a latency-sensitive flow. Since the inferred parameters may become inaccurate over time, the preemption controller recomputes the model in two conditions: 1) the difference between the predicted time and the actual time of a message exceeds 2σ; 2) a statically configured time interval has passed since last recomputing. With the time model $\{p, q, \sigma\}$, the future arrival time T_k of a latency-sensitive flow can be easily inferred as $[\hat{T}_k - \Delta, \hat{T}_k + \Delta]$ according to normal distribution under a given accuracy requirement denoted as C. We set C to 95% for which the Δ can be approximated with 2σ.

The design of the preemption controller has to cope with two challenges: first, the future arrival time of each latency-sensitive flow may overlap with each other; second, the future arrival time should be reported to the virtual-preemption layer timely, otherwise there will not be enough time for the WNIC to finish its ongoing transmission. To tackle these challenges, we use a protection window (PW) that covers the future arrival time of multiple latency-sensitive flows. Suppose the current future arrival time of all latency-sensitive flows constitutes a set T and the earliest one is $t = [t_1, t_2]$, we set the PW to t, remove t from T and look up the new earliest one t' in T. If $t' = [t'_1, t'_2]$ overlaps with PW (i.e., $t_1 \leq t'_1 \leq t_2$), t' is combined into PW (i.e., PW$= [t_1, max(t_2, t'_2)]$) and removed from T. The process repeats until a new t' does not overlap with PW, so that PW starting from t covers all future arrival time that needs to be protected as a whole. Upon the ending of a PW, the preemption controller will update the future arrival time of each time model and calculate a new PW, then it notifies the virtual-preemption layer to resume the paused bandwidth-hungry flows and update the new PW simultaneously. In this way, the virtual-preemption layer always controls the feeding of bandwidth-hungry packets according to the latest PW.

C. Virtual-Preemption Layer

A WNIC driver mainly contains three major operations: dequeuing packets from upper kernel queues; enqueuing packets to the WNIC; receiving the transmission completion callback of a packet. The virtual-preemption layer wraps these three operations as shown in Algorithm 2.

The virtual-preemption layer maintains a completion time table (CTT) for predicting the required time to transmit specific numbers of packets buffered in the WNIC. $CTT[n]$ records times it took to complete the transmission of $n+1$ bandwidth-hungry packets on the WNIC, and keeps a number of the most recent samples for estimation later. To maintain the CTT, the virtual-preemption layer intercepts into both the enqueue and the transmission completion callback of the WNIC driver (line 11-15 and line 16-20). Each time the driver is enqueueing a bandwidth-hungry packet into the WNIC's bandwidth-hungry queue, the enqueue time t_{enq} of the packet is recorded, along with the current number of packets n in the WNIC's bandwidth-hungry queue. Suppose the packet transmission is completed at time t_{comp}, we learn that the bandwidth-hungry queue needs $(t_{comp}-t_{enq})$ time to complete the transmission of a packet together with n packets previously queued in the WNIC queue (line 20).

Algorithm 2: Virtual-Preemption Layer

Data: CTT: completion time table; $[t_1, t_2]$: current PW; N_{samp}: number of samples to keep in the CTT; pkt: bandwidth-hungry packet

1 **upon** dequeue pkt of kerenl queue **do**
2 $n \leftarrow$ current number of packets in hardware queue;
3 $t_n \leftarrow percentile(CTT[n], p_{prot})$;
4 $T \leftarrow now()$;
5 **if** $(T < t_1$ && $t_1 - T \leq t_n)$ || $t_1 \leq T \leq t_2$ **then**
6 cancel dequeue;
7 pause kernel bandwidth-hungry queue;
8 **else**
9 let the driver dequeue pkt;
10 **end**
11 **upon** enqueue pkt of bandwidth-hungry flow **do**
12 $n \leftarrow$ current number of packets in hardware queue;
13 $t_{enq} \leftarrow now()$;
14 $pkt.t \leftarrow t_{enq}$;
15 $pkt.n \leftarrow n$;
16 **upon** complete pkt of bandwidth-hungry flow **do**
17 $t_{comp} \leftarrow now()$;
18 **if** $|CTT[pkt.n]| > N_{samp}$ **then**
19 remove oldest sample from $CTT[pkt.n]$;
20 **end**
21 $CTT[pkt.n] \leftarrow CTT[pkt.n] \bigcup \{t_{comp} - pkt.t\}$;
22 **upon** receive PW $[t_1', t_2']$ **do**
23 $t_1 \leftarrow t_1'$;
24 $t_2 \leftarrow t_2'$;
25 resume kernel bandwidth-hungry queue;

Given a PW denoted as $[t_1, t_2]$ and current time T, if $t_1 \geq T$, the virtual-preemption layer protects the latency-sensitive flows that are supposed to arrive within the PW by intercepting the dequeue operation of the driver (line 1-10). When the driver attempts to dequeue a bandwidth-hungry packet from the upper kernel queues, the virtual-preemption layer queries the number of packets currently queued in the WNIC n, and queries the corresponding transmission completion time t_n by finding p_{prot} percentile of the recorded times in $CTT[n]$ (line 2-3). If $t_n > t_1 - T$ or $t_2 > T > t_1$, the transmission of this bandwidth-hungry packet is not likely to be finished before the arrival of the next latency-sensitive message and would block the transmission of the latency-sensitive message (line 5). Thus the virtual-preemption layer aborts the dequeuing operation of this packet and further pauses the kernel queue of bandwidth-hungry flows (line 6-7).

However, if the PW arrives late (e.g., $t_{n-1} > t_1 - T$ when PW is updated), there could be too many bandwidth-hungry packets already queued on the WNIC, blocking the incoming latency-sensitive packets. To avoid this problem, we design the synergy between the preemption controller and the virtual-preemption layer as follows (line 22-25): the controller updates the PW whenever a PW ends (i.e., $t_2 \leq T$) and notifies the virtual-preemption layer upon the update; the virtual-preemption layer first updates its PW to the latest and then resumes the dequeue operation of the bandwidth-hungry flows on the driver. In this way, when the driver attempts to dequeue a bandwidth-hungry packet, the virtual-preemption layer will always estimate whether the bandwidth-hungry packet will block the latency-sensitive flows on the WNIC according to the latest PW. Further, each computed PW was extended with a constant (empirically set to 2ms) to mitigate the impact of the late arrival of latency-sensitive messages.

The CTT maintained in the virtual-preemption layer guarantees an accurate estimation of transmission completion time. Based on the accurate estimation, the virtual-preemption layer minimizes the time for which the driver stops transmitting bandwidth-hungry packets (**R2**) while guaranteeing that bandwidth-hungry packets queued on the WNIC will be finished transmitting before latency-sensitive packets arrive (**R1**).

D. Scalability Analysis

Here we build an analytical model for the performance of CRL applications running with COORP. We focus on two aspects, the reaction time, and the achievable throughput (i.e., bytes transmitted per second) of bandwidth-hungry flows, given $N+1$ robots with one of them chosen as the leader robot.

The reaction time mainly consists of three parts: perception exchange, inference, and control dissemination. For a given number of robots, the perception exchange and control dissemination require transmitting a fixed amount of data. For simplicity, we assume the same bandwidth across different robots, denoted as B, and omit the variation of transmission time caused by contention. Then the time required for transmission is $T_{tr} = \frac{N*L_{perc}+N*L_{act}}{B}$, where L_{perc} is the size of perception messages, and L_{act} is the size of action messages.

Meanwhile, the transmitting bandwidth-hungry flows keep trying to inject into this whole process and increase the

reaction time, denoted as T_{BH}. To simplify the analysis, we set $LIMIT$ in Algorithm 1 to 1 in the remaining of this section. To breakdown T_{BH}, first note that, given the bulk data to transmit, it is viable to assume each transmission to be the maximum size of an AMPDU L_{ampdu}, thus the major variation comes from how many times such transmission could happen, denoted as a random variable N_{BH}. Still, it is difficult to find an accurate distribution of N_{BH} given the highly dynamic nature of the CSMA/CA contention process. We choose to build a rough model only to estimate the supported number of robots. We first look at a single contention between a latency-sensitive transmission and a bandwidth-hungry transmission. Given initial CW parameters of EDCA (3 for `AC_VO`, 15 for `AC_BE` [14]), the possibility for bandwidth-hungry to transmit before latency-sensitive can be derived as $p_0 = \frac{1+2+3}{4*16} + \frac{4}{4*16} * \frac{1+2+...+7}{8*32} \approx 0.1$ by counting the cases for bandwidth-hungry to get lower backoff value and considering at most one collision. Then we treat the N times transmission of perception messages and N times transmission of action messages as $2N$ times independent contention with bandwidth-hungry to simplify the analysis, which tends to over-estimate N_{BH} and lead to conservative estimation of the supported number of robots. Then, the possibility for N_{BH} to take each value ranging from 0 to $2N$ follows

$$P(N_{BH} = k) = \binom{2N}{k} p_0^k (1-p_0)^{2N-k}, (k = 0, 1, ..., 2N). \tag{1}$$

The time for inference comes from executing specific computations on the leader robot, and could vary for different DNN architectures and different hardware (CPU, GPU, etc.), but is generally constant once the DNN architecture and the hardware are fixed, denoted as T_{inf}. For each possible number of bandwidth-hungry transmissions k, the reaction time is

$$\begin{aligned} T_k &= T_{inf} + T_{tr} + T_{BH} \\ &= T_{inf} + \frac{N * L_{perc} + N * L_{act}}{B} + \frac{k * L_{ampdu}}{B}. \end{aligned} \tag{2}$$

With Equation 1 and Equation 2, we can estimate the distribution of the reaction time given $N + 1$ robots.

For the achievable throughput of bandwidth-hungry flows, the coordination overhead is negligible, thus the major drop comes from the more and more latency-sensitive messages for transmission. Besides, COORP's local preemption sacrifices some bandwidth, but it is independent of the number of robots, thus it can be omitted for the scalability analysis. For $N + 1$ robots including the leader robot, the achievable throughput for bandwidth-hungry flows can be estimated with

$$B_{BH} = B - N * F * (L_{perc} + L_{act}) \tag{3}$$

where F is the required frequency of the MPC loop. The above model will be used in §VI-C along with simulation using NS3 for the evaluation of the scalability of COORP.

V. IMPLEMENTATION

The virtual-preemption layer was implemented as a kernel device module with about 600 lines of code based on Linux 5.4.84 with PREEMP_RT patch [46], and interfaced with the preemption controller through `ioctl()`. Three operations of the WNIC driver were hooked. The operation to enqueue a packet to the WNIC queue was hooked to record the current number of enqueued packets in the WNIC queue and the timestamp. The operation to handle packet completion from the WNIC queue was hooked to compute the transmission completion time, and update the CTT as was described in Algorithm2. Further, the virtual-preemption layer was hooked into the driver operation of dequeueing packets from mac80211 queues to prevent it from enqueueing too many packets into the WNIC queue. Although the exact code of WNIC drivers differs, these three operations are general and necessary. For evaluation, the hooks were added into the MT76 driver in the kernel source tree, and the functions corresponding to the above three operations were `mt76_queue_ops.tx_queue_skb()`, `mt76_queue_ops.tx_complete_skb()` and `mt76_txq_dequeue()`. The preemption controller was decoupled as two modules, a proxy in the ROS2 rcl (ROS client library) [47] to compute the time model for latency-sensitive flow, and a controller as a stand-alone ROS2 application. These involved around 400 lines of code in the ROS2 rcl.

Ideally, the BH controller should also be implemented in ROS2. However, ROS2 does not currently provide direct TCP support, and the default UDP support limits the size of the message (64KB) and suffers bandwidth problems in unreliable networks like WiFi [48]. Thus we chose to temporarily implement a message streaming interface with TCP socket, integrated with the BH controller.

VI. EVALUATION

Testbed. The evaluation was performed using 5 hosts running Ubuntu 20.04 with Linux 5.4.84 kernel patched with Preempt-RT [49], shown in Figure 7. Two of the hosts were desktops with Intel Core i7-8700 CPU@3.20GHz, and the other three were laptops with different models of CPUs, namely Intel Core i7-10510 CPU@1.80GHz, i5-7200U CPU@2.50GHz, and i7-8565U CPU@1.80GHz respectively. Robots were simulated with these five hosts, and we refer to these hosts as robots for convenience in this section. One of them (the leader) was equipped with MediaTek MT7612E WNIC and configured an access point on channel 44 (5GHz) which was found clear in our environment. The other four were equipped with MediaTek MT7612U USB WNICs and connected to the leader's access point. To measure transmission latency, the hosts were connected to a TP-LINK TL-SG108 desktop Ethernet switch and their system clocks were synchronized with PTP [50]. The synchronization accuracy was reported to be below 10μs by `ptp4l`, sufficient for measuring wireless transmission latency. Note that the accurate time synchronization is only for measuring the latency and reaction time, and is not required by COORP to work in real deployments. In the evaluation, the $LIMIT$ in Algorithm 1 was set to 1 for the lowest contention.

Baselines. We compared COORP with two baselines chosen from existing work: EDCA [14] from the prioritized con-

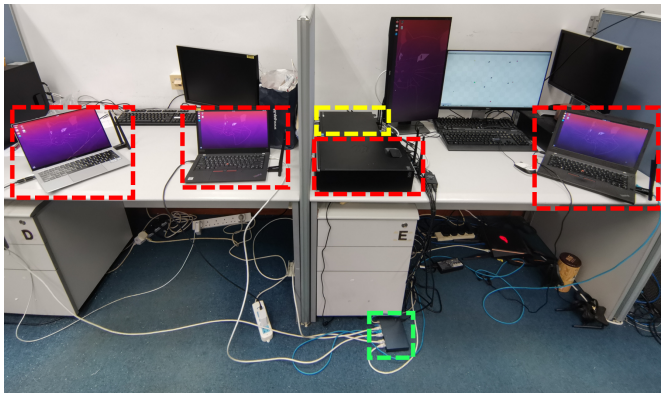


Fig. 7: Hosts (leader in yellow box, others in red box) and Ethernet switch (green box) used in the evaluation.

tention category and SchedWiFi [17] from the global planning category. Specifically, EDCA is a part of WiFi standard that prioritizes latency-sensitive flows over others during wireless channel contention. SchedWiFi is the latest work optimized for both latency-sensitive flows and bandwidth-hungry flows in the global planning category. Subsequent studies [19], [20] in the global planning category aim at fully resolving wireless channel contention, but not for the coexistence of latency-sensitive flows and bandwidth-hungry flows, oblivious of our scenario. Further, our evaluation in §VI-A shows that, both SchedWiFi and CoorP are close to the ideal latency. Therefore, we believe SchedWiFi is the tightest baseline for our goal of optimizing both latency-sensitive flows and bandwidth-hungry flows for coordinated robotic learning applications.

For EDCA, the latency-sensitive messages were configured to go through `AC_VO`, the highest priority, and the training data and parameter updates were configured to go through `AC_BE`, lower priority than the latency-sensitive messages.

For SchedWiFi, since the original paper only evaluated with simulation, to compare its actual performance with CoorP, all its features were implemented on our testbed. ST-Window (STW) is the time window reserved for each latency-sensitive flow; it was calculated with Equation 4 following the original paper, where L is the size of each message in a latency-sensitive flow, $SIFS$ is $16\mu s$ according to the standards of IEEE 802.11 [51], and the maximum retransmission time R was set to 7 [52]. TX_{ack} was empirically set to $30\mu s$. The Time-Aware Shaper of SchedWiFi required modification to the physical layer of the WNICs, which is impractical on commodity WNICs. To achieve the same effect, CoorP's virtual-preemption layer was adapted as SchedWiFi's Time-Aware Shaper which proactively stops the kernel queue before required time point.

$$STW = 2\times 25\mu s + (\frac{L}{bandwidth} + 2\times SIFS + TX_{ack})\times(1+R) \quad (4)$$

Workload. The multi-robot navigation task in [2] was adopted as the workload, since this task requires multiple robots to gather their real-time sensor data for cooperative control, and is an important building block of mission-critical multi-robot applications such as surveillance and rescue [2], [53], [54]. The multi-robot navigation task was set up in the Stage simulator [55] with 5 moving obstacles and 5 robots. The deep neural network model and the reward function were the same as that in [2]. Since the parameter size of real-world reinforcement learning models for robotics has exceeded 100 million [56] and existing methods can reduce the number of parameters by 10x-50x [57], the number of parameters was extended to 10 million with dummy data during parameter synchronization to approximate the size of reinforcement learning models in real-world tasks.

To bridge the simulated robots with real-world wireless network, the perceptions of the robots were retrieved from the simulator and sent to the corresponding host through the wired network. Each robot reported the perception it received to the leader robot at a frequency of 30Hz via the wireless network as if the perception was collected from local sensors. The perception messages were padded to 12KB (64x64 RGB images) according to RoboNet dataset [32] to approximate the size of real-world perceptions. The leader robot sent control messages back to each robot via the wireless network. The control messages were padded to 1KB as were typically considered in existing work [5], [17]. Overall, we believe our evaluation setup represents diverse latency-sensitive flows (e.g., camera image, laser scan, position, and control command) and bandwidth-hungry flows (e.g., distributed learning parameter synchronization and high-resolution video streaming) appeared in real-world robotic applications.

The evaluation focused on these questions:

- RQ1: How is CoorP compared to baseline systems in terms of *fast reaction* and *fast model convergence*?
- RQ2: What is the effectiveness of different components of CoorP?
- RQ3: How does CoorP scale with the number of robots?
- RQ4: What are the limitations of CoorP?

A. End-to-end Performance

For end-to-end comparison between CoorP and the baseline systems, the reaction time (i.e., the time to finish each MPC loop), the reward of the control model, and the power consumption were recorded during fine-tuning a pre-trained model. Figure 2a shows the CDF of reaction time of different systems. The vertical line marks the typical reaction time boundary (33ms [9], [58], [59]).

Among all the three systems, EDCA had the highest violation rate (53.9%) of the reaction time boundary, since it did not resolve local congestion in the WNIC, and the simultaneous transmission of multiple (1 to 5) bandwidth-hungry flows resulted in heavy congestion. SchedWiFi best guaranteed fast reaction with a 3.5% violation rate, because latency-sensitive messages were assigned exclusive and long transmission time slots according to Equation 4, so as to cope with various uncertainty and avoid interference of any bandwidth-hungry messages. CoorP also achieved a low violation rate (8.8%) of the reaction time boundary, comparable to SchedWiFi and much lower than EDCA. These results correspond to the recorded CDF of latency of latency-sensitive

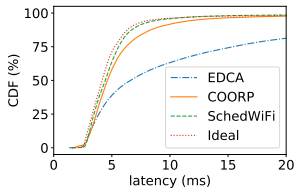


Fig. 8: Latency CDF of latency-sensitive messages.

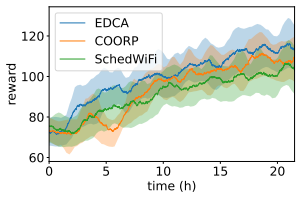


Fig. 9: Convergence over time, SSP.

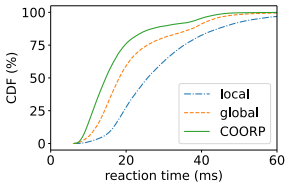


Fig. 10: Contribution of COORP's components to the reaction time. In 'local' ('global'), only local (global) preemption was enforced.

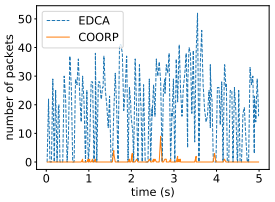


Fig. 11: Effectiveness of the virtual-preemption layer. Y-axis is the number of packets in the WNIC queue when latency-sensitive packets enqueue, which is zero most of the time in COORP.

TABLE I: Time per epoch. Normalized to EDCA's.

	EDCA	SchedWiFi	COORP
BSP	1.00	1.49	1.15
SSP	1.00	1.56	1.13
ASP	1.00	1.56	1.16

messages achieved by different systems in Figure 8. It is worth noting that, both SchedWiFi and COORP achieved comparable latency to the ideal baseline (within a 12% difference), while SchedWiFi outperformed COORP by eliminating nearly all contention in the wireless channel with the cost of much slower convergence as follows.

The convergence of the control model is shown in Figure 2b (BSP) and Figure 9 (SSP, the limit of version difference of the model between the fastest worker and the slowest worker was set to 1). ASP was not converging in this task using all three systems, thus its figure is omitted. The non-convergence of ASP would be due to outdated updates from slow workers [34]. All three systems started from the same pre-trained model. Despite the jitters caused by the inherent uncertainty of reinforcement learning, overall, COORP was slightly slower than EDCA but much faster than SchedWiFi. Such a difference was a result of time required per epoch shown in Table I: SchedWiFi required 49%~56% more time per epoch, while COORP required 13%~16% more.

Table II shows the bandwidth utilization of different systems which further explains the difference of time per epoch. The throughput of the ideal baseline (i.e., bytes transmitted per second when MPC was disabled) was 198.3Mbps. EDCA was the clothest to the ideal baseline (98%). SchedWiFi was only 58% since all the robots had to pause bandwidth-hungry flows for all the time slots for latency-sensitive messages, and the sizes of the time slots were often oversized to handle clock skewness, limiting the overall time for bandwidth-hungry flows to transmit. In COORP, although only one robot was allowed to transmit bandwidth-hungry flow at any time, the only transmitting bandwidth-hungry flow was able to saturate the bandwidth, and each robot only needed to pause bandwidth-hungry flows for its own latency-sensitive messages. Besides, the coordination between global controller and BH controller incurred little overhead. Thus there was a minor drop (13%) of the bandwidth utilization, leading to shorter time per epoch

TABLE II: Comparison of bandwidth utilization.

	EDCA	SchedWiFi	COORP
utilization	0.98 (194.7Mbps)	0.58 (115.1Mbps)	0.85 (168.2Mbps)

TABLE III: Impact of COORP's components on the bandwidth utilization. In 'local' ('global'), only local (global) preemption was enabled. Normalized to EDCA's (194.7Mbps).

	local	global	COORP
utilization	0.87 (169.4Mbps)	0.98 (192.1Mbps)	0.86 (168.2Mbps)

and faster model convergence compared with SchedWiFi.

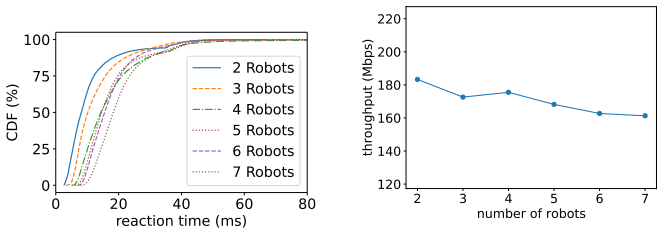
The power consumption of the laptops was measured with Gnome Power Statistics and exhibited no significant difference (16.4±0.3W) for different systems, because the computation overhead of scheduling the network traffic was negligible compared to that of the applications.

B. *Effectiveness of Components*

To understand the effectiveness of COORP's components, the following two cases were studied: the BH controller was disabled such that only local preemption was enforced; the preemption controller was disabled such that only global preemption was enforced. The corresponding reaction time is shown in Figure 10. When only global preemption was enabled, there was at most one bandwidth-hungry flow contending with latency-sensitive flows at any time, and local congestion was no longer exacerbated by bandwidth-hungry flows from other WNICs. When only local preemption was enabled, local congestion was eliminated, but latency-sensitive flows still contend with multiple bandwidth-hungry flows. Further, since the timing of control messages sent from the leader robot depends on the latency of perception messages and has a large variance (several milliseconds), control messages were not handled by local preemption. Thus global preemption was more effective when enabled alone, but the combination of them led to an even lower reaction time.

The corresponding bandwidth utilization is shown in Table III. Global preemption sacrificed little bandwidth because each single bandwidth-hungry flow was able to saturate the bandwidth. Local preemption accounted for most of the sacrifice of the bandwidth because it had to pause ongoing bandwidth-hungry flow in each protection window.

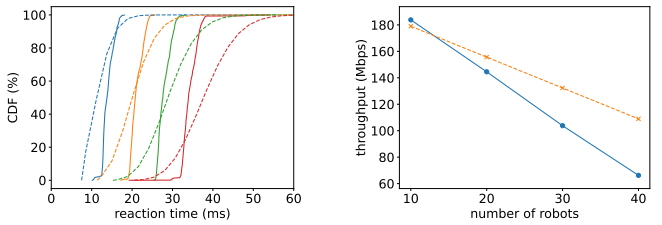
To further validate the effectiveness of the virtual-preemption layer, the number of bandwidth-hungry packets queued in the WNIC when latency-sensitive packets were



(a) Reaction time.

(b) Throughput of bandwidth-hungry flows.

Fig. 12: Scalability of COORP.



(a) Reaction time. The number of robots are 10, 20, 30, and 40 from left to right respectively.

(b) Throughput of bandwidth-hungry flows.

Fig. 13: Scalability of COORP, model (dotted lines) and simulation (solid lines).

passed to the WNIC was inspected, shown in Figure 11. In EDCA, bandwidth-hungry packets accumulated in the WNIC queue and would block the transmission of latency-sensitive packets. In COORP, the virtual-preemption layer avoided passing too many bandwidth-hungry packets to the WNIC before latency-sensitive packets. Thus, the WNIC was able to finish transmitting bandwidth-hungry packets before latency-sensitive packets were passed and transmit latency-sensitive packets as soon as possible. However, the CTT of the virtual-preemption layer was not always accurate, thus it was not able to ensure the WNIC queue was always clean.

C. Scalability to the Number of Robots

The scalability of COORP to the number of robots was studied with both real experiments and simulations. In the real experiments, the number of robots was scaled from 2 to 7. In the simulations, an application that generates the same network traffic pattern as the CRL application in the real experiments running with COORP was implemented in NS3. The WiFi standard and the number of antennas were set to the same as our testbed (IEEE 802.11ac, two antennas for each wireless node). In each run of the simulations, the leader robot was placed at the origin, and the other robots were randomly placed in a disc centered at the origin with 10m radius.

Figure 12a shows the scalability of COORP in terms of reaction time in real experiments. With more robots, the contention caused by the transmission of latency-sensitive messages from different robots increases, and it takes more time to gather all the perceptions from all the robots, thus there was a slight increase in the overall reaction time with the number of robots. Meanwhile, the CDF at the reaction time boundary (33ms) exhibited little change, because the tail was mainly influenced by contention from bandwidth-hungry flows which had been mitigated and was independent of the number of robots when using COORP. Figure 12b shows the scalability of the throughput of bandwidth-hungry flows in the real experiments. In COORP, the sacrifice of throughput comes from two aspects. First, enforcing global preemption requires requests and responses between BH controller and global controller, incurring a small traffic volume compared to that of actually transmitting bandwidth-hungry flows. Second, enforcing local preemption requires each robot only to pause transmission of bandwidth-hungry messages for its own latency-sensitive messages, which is independent of the number of robots.

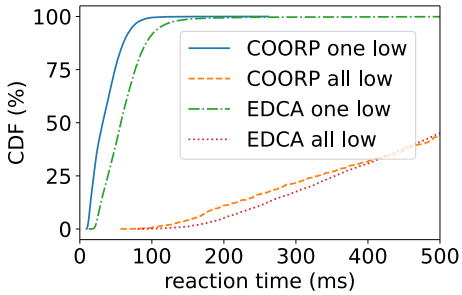


Fig. 14: Reaction time under low available bandwidth. In 'one(all) low', one(all) of the robots used the lowest transmission rate.

Thus, the decrease of the achieved throughput of bandwidth-hungry flows was mainly due to the increasing traffic volume of latency-sensitive flows in the whole robot group.

The results of simulations are shown in Figure 13a and Figure 13b along with the results derived from the analytical model in §IV-D. Taking 33ms as the reaction time boundary, both simulations and the analytical model imply a limit of around 40 robots. Meanwhile, the throughput of bandwidth-hungry flows dropped linearly with respect to the number of robots. Further, the simulations exhibited generally lower reaction time than real experiments. This would be due to the difference between simulated environments and real environments. Indeed, the analytical model has a gap with the simulations due to coarse approximations, but it suffices for the purpose to estimate the supported number of robots.

D. Performance under Low Available Bandwidth

In coordinated robotic learning, robots move around and may result in significant drop of available bandwidth. To understand the performance of COORP under such scenarios, two individual cases were set up, and the results were shown in Figure 14 and Table IV.

In the first case, denoted as 'one low', one of the robots was configured to use the lowest transmission rate (or bitrate) `vht-mcs-5 1:0` using `iw` tool. Compared to COORP, EDCA achieved around 30% lower throughput. This would be related to the performance anomaly of 802.11 previously discussed by [39], [60] that the stations in a wireless network

TABLE IV: Throughput under low available bandwidth. In 'one(all) low', one(all) of the robots used the lowest transmission rate.

	EDCA	COORP
one low	99.8Mbps	148.1Mbps
all low	19.6Mbps	16.3Mbps

would achieve similar throughput even when their network conditions differ. By only allowing one robot to transmit simultaneously, COORP enabled each robot to achieve the highest throughput allowed by its network condition, resulting in higher aggregate throughput (averaged over time). This indicates that COORP provides faster training than EDCA when the available bandwidth of a small number of robots is low.

In the second case, denoted as 'all low', all the robots were configured to use the lowest transmission rate. Both EDCA and COORP suffered a significant increase of reaction time and a low throughput due to the restricted bandwidth, which would render the DRL application unavailable. In real deployments, robots may pause the task and restore the connectivity proactively [61], or continue the task suboptimally with little coordination until the connectivity recovers.

E. Future Applications

Diverse applications can benefit from COORP. For example, in robot soccer [24], a team of robots exchange teammates' poses, ball's position, and opponents' poses in real-time (latency-sensitive flows) while learning to adapt to new opponents (bandwidth-hungry flows). Multi-view object tracking [62] collects and matches images from different cameras in real-time (latency-sensitive flows) while learning to adapt to the change of varying factors like weather condition (bandwidth-hungry flows). In large-scale disaster response [26], multiple robots can coordinate their angles and sensors in the disaster environment in real-time (latency-sensitive flows) while learning to optimize their actions in unseen environments (bandwidth-hungry flows). We leave the realization of these applications as future work.

VII. CONCLUSION

In this paper, we identify two technical requirements of the network supporting system for mission-critical coordinated robotic learning, namely fast reaction and fast model convergence, and present COORP, the first network supporting system that achieves both of the requirements through a software-only cross-layer design. COORP facilitates diverse coordinated robotic learning applications, including multi-robot navigation, robot soccer, multi-view object tracking, and large-scale disaster response. COORP's code is released on github.com/hku-systems/Coorp.

REFERENCES

[1] J. P. Queralta, J. Taipalmaa, B. Can Pullinen, V. K. Sarker, T. Nguyen Gia, H. Tenhunen, M. Gabbouj, J. Raitoharju, and T. Westerlund, "Collaborative multi-robot search and rescue: Planning, coordination, perception, and active vision," vol. 8, pp. 191 617–191 643, publisher: IEEE.

[2] R. Han, S. Chen, and Q. Hao, "Cooperative Multi-Robot Navigation in Dynamic Environment with Deep Reinforcement Learning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, May 2020, pp. 448–454, iSSN: 2577-087X.

[3] E. Vidal, J. D. Hernández, N. Palomeras, and M. Carreras, "Online robotic exploration for autonomous underwater vehicles in unstructured environments," in *2018 OCEANS - MTS/IEEE Kobe Techno-Oceans (OTO)*, pp. 1–4.

[4] Y. Zhang, D. Shi, Y. Wu, Y. Zhang, L. Wang, and F. She, "Networked Multi-robot Collaboration in Cooperative–Competitive Scenarios Under Communication Interference," in *Collaborative Computing: Networking, Applications and Worksharing*, H. Gao, X. Wang, M. Iqbal, Y. Yin, J. Yin, and N. Gu, Eds. Cham: Springer International Publishing, 2021, vol. 349, pp. 601–619.

[5] D. Tardioli, R. Parasuraman, and P. Ögren, "Pound: A multi-master ROS node for reducing delay and jitter in wireless multi-robot networks," *Robotics and Autonomous Systems*, vol. 111, pp. 73–87, Jan. 2019.

[6] R. Jeong, Y. Aytar, D. Khosid, Y. Zhou, J. Kay, T. Lampe, K. Bousmalis, and F. Nori, "Self-Supervised Sim-to-Real Adaptation for Visual Robotic Manipulation," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, May 2020, pp. 2718–2724, iSSN: 2577-087X.

[7] R. Nishihara, P. Moritz, S. Wang, A. Tumanov, W. Paul, J. Schleier-Smith, R. Liaw, M. Niknami, M. I. Jordan, and I. Stoica, "Real-Time Machine Learning: The Missing Pieces," *arXiv:1703.03924 [cs]*, May 2017, arXiv: 1703.03924.

[8] J. Song and M. Kountouris, "Wireless Distributed Edge Learning: How Many Edge Devices Do We Need?" *arXiv:2011.10894 [cs]*, Nov. 2020, arXiv: 2011.10894.

[9] H. Nascimento, M. Mujica, and M. Benoussaad, "Collision avoidance interaction between human and a hidden robot based on kinect and robot data fusion," *IEEE Robotics and Automation Letters*, vol. 6, no. 1, pp. 88–94, 2021, publisher: IEEE.

[10] Changhua Pei, Youjian Zhao, Yunxin Liu, Kun Tan, Jiansong Zhang, Yuan Meng, and Dan Pei, "Latency-based WiFi congestion control in the air for dense WiFi networks," in *2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*, Jun. 2017, pp. 1–10.

[11] D. Kim, I. Yeom, and T.-J. Lee, "Mitigating tail latency in IEEE 802.11–based networks," *International Journal of Communication Systems*, vol. 31, no. 1, p. e3404, 2018.

[12] M. Heusse, F. Rousseau, R. Guillier, and A. Duda, "Idle Sense: An Optimal Access Method for High Throughput and Fairness in Rate Diverse Wireless LANs," p. 12.

[13] A. Saeed, M. Ammar, E. Zegura, and K. Harras, "If you can't Beat Them, Augment Them: Improving Local WiFi with Only Above-Driver Changes," in *2018 IEEE 26th International Conference on Network Protocols (ICNP)*, Sep. 2018, pp. 378–388, iSSN: 1092-1648.

[14] "IEEE 802.11e-2005," Feb. 2021, page Version ID: 1006381497. [Online]. Available: https://en.wikipedia.org/w/index.php?title=IEEE_802.11e-2005&oldid=1006381497

[15] Z. Yang, J. Zhang, K. Tan, Q. Zhang, and Y. Zhang, "Enabling TDMA for today's wireless LANs," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, Apr. 2015, pp. 1436–1444, iSSN: 0743-166X.

[16] Y. Wei, Q. Leng, S. Han, A. K. Mok, W. Zhang, and M. Tomizuka, "RT-WiFi: Real-Time High-Speed Communication Protocol for Wireless Cyber-Physical Control Applications," in *2013 IEEE 34th Real-Time Systems Symposium*, Dec. 2013, pp. 140–149, iSSN: 1052-8725.

[17] G. Patti, G. Alderisi, and L. L. Bello, "SchedWiFi: An innovative approach to support scheduled traffic in ad-hoc industrial IEEE 802.11 networks," in *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, Sep. 2015, pp. 1–9, iSSN: 1946-0759.

[18] F. Santos, L. Almeida, and L. S. Lopes, "Self-configuration of an adaptive TDMA wireless communication protocol for teams of mobile robots," in *2008 IEEE International Conference on Emerging Technologies and Factory Automation*, Sep. 2008, pp. 1197–1204, iSSN: 1946-0759.

[19] S. Cao and V. C. S. Lee, "A Novel Adaptive TDMA-Based MAC Protocol for VANETs," *IEEE Communications Letters*, vol. 22, no. 3, pp. 614–617, Mar. 2018, conference Name: IEEE Communications Letters.

[20] X. Guo, S. Wang, H. Zhou, J. Xu, Y. Ling, and J. Cui, "Performance Evaluation of the Networks with Wi-Fi based TDMA Coexisting with CSMA/CA," *Wireless Personal Communications*, vol. 114, no. 2, pp. 1763–1783, Sep. 2020.

[21] P. Bosch, S. Latré, and C. Blondia, "Latency Modelling in IEEE 802.11 Systems with non-IEEE 802.11 Interfering Source," in *2018 14th International Conference on Network and Service Management (CNSM)*, Nov. 2018, pp. 275–280, iSSN: 2165-963X.

[22] "IEEE Standard for Information Technology–Telecommunications and Information Exchange between Systems - Local and Metropolitan Area Networks–Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," *IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016)*, pp. 1–4379, Feb. 2021, conference Name: IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016).

[23] "Frame aggregation," Sep. 2020, page Version ID: 978836459. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Frame_aggregation&oldid=978836459

[24] J. M. C. Ocana, F. Riccio, R. Capobianco, and D. Nardi, "Cooperative Multi-Agent Deep Reinforcement Learning in Soccer Domains," p. 3, 2019.

[25] M. Fernández-Sanjurjo, M. Mucientes, and V. M. Brea, "Real-Time Multiple Object Visual Tracking for Embedded GPU Systems," *IEEE Internet of Things Journal*, vol. 8, no. 11, pp. 9177–9188, Jun. 2021, conference Name: IEEE Internet of Things Journal.

[26] K.-M. Yang, J.-B. Han, and K.-H. Seo, "A Multi-robot Control System based on ROS for Exploring Disaster Environment," in *2019 7th International Conference on Control, Mechatronics and Automation (ICCMA)*, Nov. 2019, pp. 168–173.

[27] J. Scherer, S. Yahyanejad, S. Hayat, E. Yanmaz, T. Andre, A. Khan, V. Vukadinovic, C. Bettstetter, H. Hellwagner, and B. Rinner, "An Autonomous Multi-UAV System for Search and Rescue," in *Proceedings of the First Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use*. Florence Italy: ACM, May 2015, pp. 33–38.

[28] R. N. Darmanin and M. K. Bugeja, "A review on multi-robot systems categorised by application domain," in *2017 25th Mediterranean Conference on Control and Automation (MED)*, Jul. 2017, pp. 701–706, iSSN: 2473-3504.

[29] K. Zhang, Z. Yang, and T. Basar, "Networked multi-agent reinforcement learning in continuous spaces," in *2018 IEEE Conference on Decision and Control (CDC)*, pp. 2771–2776.

[30] M. J. Matarić, "Reinforcement Learning in the Multi-Robot Domain," in *Robot Colonies*, R. C. Arkin and G. A. Bekey, Eds. Boston, MA: Springer US, 1997, pp. 73–83.

[31] J. Ibarz, J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, and S. Levine, "How to train your robot with deep reinforcement learning: lessons we have learned," *The International Journal of Robotics Research*, vol. 40, no. 4-5, pp. 698–721, Apr. 2021, publisher: SAGE Publications Ltd STM.

[32] S. Dasari, F. Ebert, S. Tian, S. Nair, B. Bucher, K. Schmeckpeper, S. Singh, S. Levine, and C. Finn, "RoboNet: Large-Scale Multi-Robot Learning," *arXiv:1910.11215 [cs]*, Jan. 2020, arXiv: 1910.11215.

[33] A. Nagabandi, I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn, "Learning to Adapt in Dynamic, Real-World Environments Through Meta-Reinforcement Learning," *arXiv:1803.11347 [cs, stat]*, Feb. 2019, arXiv: 1803.11347.

[34] X. Zhao, A. An, J. Liu, and B. X. Chen, "Dynamic Stale Synchronous Parallel Distributed Training for Deep Learning," *arXiv:1908.11848 [cs, stat]*, Aug. 2019, arXiv: 1908.11848.

[35] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: a survey," in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 737–744.

[36] Y. Cho, S. Manzoor, and Y. Choi, "Adaptation to environmental change using reinforcement learning for robotic salamander," vol. 12, no. 3, pp. 209–218.

[37] T. Chaffre, J. Moras, A. Chan-Hon-Tong, and J. Marzat, "Sim-to-real transfer with incremental environment complexity for reinforcement learning of depth-based robot navigation."

[38] J. van Baar, A. Sullivan, R. Cordorel, D. Jha, D. Romeres, and D. Nikovski, "Sim-to-real transfer learning using robustified controllers in robotic tasks involving complex dynamics."

[39] T. Høiland-Jørgensen, M. Kazior, D. Täht, A. Brunstrom, and P. Hurtig, "Ending the Anomaly: Achieving Low Latency and Airtime Fairness in WiFi," p. 15, 2017.

[40] D. Taht, J. Gettys, E. Dumazet, T. Hoeiland-Joergensen, E. Dumazet, P. McKenney, J. Gettys, T. Hoeiland-Joergensen, and P. McKenney, "The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm." [Online]. Available: https://tools.ietf.org/html/rfc8290

[41] IEEE 802.1: 802.1qbu - frame preemption. [Online]. Available: https://www.ieee802.org/1/pages/802.1bu.html

[42] T. Adame, M. Carrascosa, and B. Bellalta, "Time-sensitive networking in IEEE 802.11be: On the way to low-latency WiFi 7."

[43] E. Khorov, I. Levitsky, and I. F. Akyildiz, "Current status and directions of IEEE 802.11be, the future wi-fi 7," vol. 8, pp. 88 664–88 688.

[44] "IEEE 802.11 RTS/CTS," Nov. 2020, page Version ID: 991386765. [Online]. Available: https://en.wikipedia.org/w/index.php?title=IEEE_802.11_RTS/CTS&oldid=991386765

[45] "Hidden node problem," Oct. 2020, page Version ID: 984728511. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Hidden_node_problem&oldid=984728511

[46] RTwiki. [Online]. Available: https://rt.wiki.kernel.org/index.php/Main_Page

[47] "ros2/rcl," Apr. 2021, original-date: 2015-02-21T00:06:40Z. [Online]. Available: https://github.com/ros2/rcl

[48] "DDS tuning information." [Online]. Available: https://index.ros.org/doc/ros2/Troubleshooting/DDS-tuning/

[49] "realtime:start [Wiki]." [Online]. Available: https://wiki.linuxfoundation.org/realtime/start

[50] "Precision Time Protocol," Jan. 2021, page Version ID: 1001872854. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Precision_Time_Protocol&oldid=1001872854

[51] "Short interframe space," page Version ID: 988080746. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Short_Interframe_Space&oldid=988080746

[52] "802.11/MAC/Lower/Retransmissions – WARP Project." [Online]. Available: https://warpproject.org/trac/wiki/802.11/MAC/Lower/Retransmissions

[53] P. Long, T. Fan, X. Liao, W. Liu, H. Zhang, and J. Pan, "Towards Optimally Decentralized Multi-Robot Collision Avoidance via Deep Reinforcement Learning," *arXiv:1709.10082 [cs]*, May 2018, arXiv: 1709.10082.

[54] A. Benevento, M. Santos, G. Notarstefano, K. Paynabar, M. Bloch, and M. Egerstedt, "Multi-Robot Coordination for Estimation and Coverage of Unknown Spatial Fields," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, May 2020, pp. 7740–7746, iSSN: 2577-087X.

[55] R. Vaughan, "rtv/Stage," May 2021, original-date: 2010-05-19T13:40:02Z. [Online]. Available: https://github.com/rtv/Stage

[56] "RoboNet: A Dataset for Large-Scale Multi-Robot Learning – The Berkeley Artificial Intelligence Research Blog." [Online]. Available: https://bair.berkeley.edu/blog/2019/11/26/robo-net/

[57] N. Liu, X. Ma, Z. Xu, Y. Wang, J. Tang, and J. Ye, "AutoCompress: An automatic DNN structured pruning framework for ultra-high compression rates," vol. 34, no. 4, pp. 4876–4883, number: 04.

[58] X. Sun, X. Weng, and K. Kitani, "When We First Met: Visual-Inertial Person Localization for Co-Robot Rendezvous," *arXiv:2006.09959 [cs]*, Nov. 2020, arXiv: 2006.09959.

[59] L. S. Scimmi, M. Melchiorre, S. Mauro, and S. Pastorelli, "Experimental Real-Time Setup for Vision Driven Hand-Over with a Collaborative Robot," in *2019 International Conference on Control, Automation and Diagnosis (ICCAD)*, Jul. 2019, pp. 1–5.

[60] M. Heusse, F. Rousseau, G. Berger-Sabbatel, and A. Duda, "Performance anomaly of 802.11b," in *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No.03CH37428)*, vol. 2, Mar. 2003, pp. 836–843 vol.2, iSSN: 0743-166X.

[61] Y. Marchukov and L. Montano, "Multi-robot coordination for connectivity recovery after unpredictable environment changes," *IFAC-PapersOnLine*, vol. 52, no. 8, pp. 446–451, 2019.

[62] W. Liu, O. Camps, and M. Sznaier, "Multi-camera Multi-Object Tracking," *arXiv:1709.07065 [cs]*, Sep. 2017, arXiv: 1709.07065.