# Lahmol: Achieving Low Latency and High Throughput Simultaneously in Wireless Multi-Robot Online Learning

Anonymous Author

## ABSTRACT

In multi-perception distributed online learning applications on a wireless multi-robot system, bandwidth-hungry traffic flows for synchronizing learned parameters among robots and latency-sensitive flows for coordinated perceptions and actions coexist. Unfortunately, in such scenarios, WNIC queues are often occupied by bandwidth-hungry packets and latency-sensitive packets need to wait until the clearance of those packets ahead before they can be transmitted, causing the head-of-line (HoL) blocking problem. Worse, multiple bandwidth-hungry flows from different robots in the same channel will contend with each other and the clearance of WNIC queues takes even longer, deteriorating the HoL blocking.

To mitigate HoL blocking in such scenarios, we propose Lahmol, a software level solution designed for commodity WNICs to transparently and automatically enforce latency-throughput service level agreements (LT-SLAs): each LT-SLA describes a stringent latency requirement while still requiring a high throughput that is close to the bandwidth capacity of a WNIC.

Lahmol lies between the WNIC and the user space applications and controls the enqueueing of both bandwidth-hungry and latency-sensitive traffic flows. Globally, to avoid contention among bandwidth-hungry flows, Lahmol divides a channel's time into slices and assign each slice exclusively to a bandwidth-hungry flows. Locally, on each robot, Lahmol predicts when a latency-sensitive packet will arrive; leveraging this prediction, Lahmol dynamically controls the max number of enqueued bandwidth-hungry packets in the WNIC queues and ensures that whenever a latency-sensitive packet arrives, the WNIC queue is clear; therefore, latency-sensitive flows get rid of HoL blocking with minimum interruption on the bandwidth-hungry flows.

Lahmol's design doesn't intrude any logic of the WNIC and its driver and thus can be easily integrated with other commodity WNICs. We evaluated Lahmol on ROS 2 with the RoboNet dataset and compared Lahmol with four baselines: default WiFi 5, WiFi EDCA, Pound, and SchedWiFi. Evaluations show that Lahmol effectively enforces the LT-SLAs, i.e. securing 10 ms end-to-end latency for latency-sensitive flows with 3.2% violation rate while maintaining 77.7% of the bandwidth capacity of a WNIC.

## CCS CONCEPTS

• **Computer systems organization → Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

## KEYWORDS

TODO

**ACM Reference Format:**
Anonymous Author. 2018. Lahmol: Achieving Low Latency and High Throughput Simultaneously in Wireless Multi-Robot Online Learning. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/1122445.1122456

## 1 INTRODUCTION

The latest wireless technologies of 5G and Wi-Fi 6 are accommodating real-time wireless applications where both latency-sensitive traffic flows and bandwidth-hungry flows coexist, which unfortunately brings special challenges to the wireless networks. For example, online reinforcement learning deployed on multi-robots requires model parameters or training data exchange to train or fine-tune the model and brings bandwidth-hungry traffic flows. They would congest the network and block the latency-sensitive messages that are essential for robot applications, like real-time sensor data and locations. Such congestion leads to violation of the stringent latency requirement and failed robotic operations and can cause severe consequences like collisions, deadlocks or out-of-order.

To address the catastrophe brought by the network congestion, the Real Time Application Technical Interest Group [] is promoting to transfer the wired time-sensitive networking (TSN) features like time-aware shaping and frame preemption, to wireless networks, which is also a major research aspect of Wi-Fi 7 []. In a wired TSN network, switch and devices connected to it separate traffic flows based on a coordinated scheduling and enforce prioritized transmission, so that latency-sensitive flows can virtually enjoy a free channel and achieve optimal latency.

Unfortunately, we identify that directly transferring the solution in wired TSN to wireless networks is confronted with a fundamental problem: the coordinated scheduling is not suitable for wireless networks due to their dynamicity. In wired TSN, because the network is switch based and each device has an exclusive cable, the transmission speed and latency are both stable. Thus, transmission time in a wired TSN network is also stable and can correctly fit in a pre-configured scheduling. In contrast, in a Wi-Fi network scenario, devices typically share a same and interfered channel, the devices are mobilized, and many techniques to increase bandwidth are used, causing the fluctuation in delay, transmission speed and transmission time. If a coordinated scheduling is reached, due to the dynamicity of the network, either the scheduled time slot for a transmission is too short to accommodate the varying transmission time, or the scheduled time slot is long enough, but wastes too much of the whole bandwidth because too much redundance is introduced. In a word, in dynamic wireless networks, a coordinated scheduling cannot precisely and efficiently stagger transmission of different priority, leading to failure of TSN features.

To tackle this problem, we identify that from the perspective of a device, the coordinated scheduling in wired TSN can be divided into non-local scheduling that indicates scheduling for traffic flows from other devices and local scheduling for traffic flows from itself; non-local scheduling that needs coordination can be bypassed and only local scheduling needs to be enforced thanks to the opportunity provided by wireless networks.

Non-local scheduling in wired TSN can be replaced by an priority-aware channel access mechanism in wireless TSN. In switch-based wired TSN, from the perspective of a certain device, both local flows and non-local ones can be buffered in the switch and cause congestion problem, so wired TSN relies on coordinated non-local scheduling to avoid congestion caused by non-local flows. Nevertheless, in wireless networks, since no switch is involved, such buffering does not exist and non-local flows cause congestion by contending for the channel with local latency-sensitive flows. Such contention can typically be mitigated by an priority-aware channel access mechanism like enhanced distributed channel access (EDCA) and thus coordinated non-local scheduling is not necessary.

Although this provides an opportunity for wireless TSN, the design of an efficient and robust priority-aware channel access mechanism is yet another challenge. For example, we observe that contention intensity is the key element for EDCA to guarantee a prioritized channel access as required by wireless TSN. When contention gets intensive by introducing multiple bandwidth-hungry flows to contend for the channel simultaneously, EDCA fails and the latency of latency-sensitive flows increases, because bandwidth-hungry flows get higher possibility to acquire the channel. So mechanisms to control contention intensity is important for EDCA, but its design is non-trivial. A strawman approach is to allow only one node at a time to transmit its bandwidth-hungry flow until finishing, but it brings unfairness when transmission speeds of different nodes vary. The slowest node will occupy transmission opportunity for too long, degrading the overall throughput. A simple fix of this approach is to make each node transmit a constraint amount of data in turn at a time. However, in extreme situations a node may move too far and its transmission speed declines significantly, leading to long transmission time even for a small amount of data.

Inspired by the concept of time division multiple access (TDMA), we propose a semi-TDMA enhancement for EDCA to control contention intensity and enforce prioritized channel access while guaranteeing fairness. Latency-sensitive flows are typically small and last for a short time, so we neglect their influence on contention intensity. For bandwidth-hungry flows, a chosen leader will assign an exclusive time slice of equal length at a time to a node in turn to transmit bandwidth-hungry flows, so that each node has equal transmission opportunity. The redundant time can be recycled and assigned to other nodes, so as to better utilize transmission opportunity. Since such scheduling is only aimed for bandwidth-hungry flows that are not sensitive to latency, no precise coordination is needed. From Figure [TODO] we observe no evident throughput degradation through our proposed method and the latency of latency-sensitive flows maintains optimal.

Different from non-local scheduling, local scheduling in TSN mainly aims at solving congestion problem caused by bandwidth-hungry flows from the same node and dynamicity is a major challenge. Retransmission due to channel interference, which is common for bandwidth-hungry flows, can increase the transmission time and block the latency-sensitive flows. Increased distance when devices are moving further from each other will cause the bandwidth to decline and increase transmission time for a packet. Wi-Fi techniques like frame aggregation increase throughput significantly, but also make the transmission time highly variable. These together make the local scheduling difficult. A strawman approach is to allow latency-sensitive packets preempt bandwidth-hungry ones. However, this approach requires redesign of existing protocols of TSN and Wi-Fi 7, which is still under research, and the commodity wireless network interface controllers (WNICs) are not ready to support such features. Besides, in the classic design of computer operating system and WNICs, operating system and the WNIC are working asynchronously and operating system cannot manipulate the transmission process of WNICs directly, making it difficult to ensure prioritized transmission.

To tackle this challenge, we observe that the transmission of packets on WNICs can be manipulated indirectly by controlling the order in which packets are fed to the WNIC. It provides opportunity to manipulate future transmission of different traffic flows happening on the WNIC and support prioritized transmission for wireless TSN. Based on this opportunity, we further leverage the most recent records of transmission time to empirically predict the future transmission time and form a temporary local scheduling that adapts to the dynamicity of the network.

Here, we propose LAHMOL, an OS level wireless TSN system that enables hybrid scheduling traffic shaping[TODO]. LAHMOL lies in the kernel level and controls the enqueueing of both bandwidth-hungry and latency-sensitive flows into the WNIC queues. Globally, to avoid contention among bandwidth-hungry flows, LAHMOL divides a channel's time into slices and makes each time slice exclusive to a bandwidth-hungry traffic. Locally, on each robot, LAHMOL acts as a time-aware traffic shaper. It models the traffic pattern of latency-sensitive flows and predicts when a latency-sensitive packet will arrive; leveraging this prediction, LAHMOL performs a WNIC queue control. LAHMOL dynamically controls the number of enqueued bandwidth-hungry packets in the WNIC queues and ensures that whenever a latency-sensitive packet arrives, the WNIC queue is clear.

We implemented LAHMOL on Robot Operating System 2 (ROS2)[6] over Linux-RT [7] with MediaTek Mt76 WNICs that support WiFi 5 and have open source drivers. We chose four major baselines: default WiFi, WiFi with EDCA, Sched-WiFi, and Pound and we used RoboNet [5], a dataset for large-scale multi-robot learning as our main dataset for a multi-perception distributed learning application. Evaluation shows that LAHMOL can guarantee prioritized transmission [TODO] automatically and efficiently, and outperforms all the baselines in terms of the balance between overall throughput and end-to-end latency:

- LAHMOL could ensure with a high possibility (99.6%) that when a latency-sensitive packet arrived, bandwidth-hungry packets queued in WNIC queues were already clear.
- LAHMOL could maintain a lowest latency SLAs violation rate (2.3%) with a minor degradation (22.3%) in overall throughput, while the baselines either fail to secure latency SLAs or sacrifice too much bandwidth capability.

Summarizing, our main contribution is LAHMOL, the first software level solution to automatically resolve the conflict between bandwidth-hungry flows and latency-sensitive flows in WNIC queues and enforce LT-SLAs for latency-sensitive flows and bandwidth-hungry ones in multi-robot distributed online learning applications. Our major novelty are the driver level queue control function of LAHMOL that can be easily integrated with commodity WNICs and effectively solve the conflict among flows in WNIC queues. With the LT-SLAs secured, besides traditional multi-robot applications, more distributed online learning applications can be applied to multi-robot systems like adaptation to new environments in the long term robot control through reinforcement learning [16].

The rest of this paper is described as follows. Section 2 describes the background, related works and the motivation for LAHMOL; Section 3 gives an overview of LAHMOL; Section 4 presents the detailed design for LAHMOL's key components to enforce LT-SLAs; Section 5 and Section 6 describe LAHMOL's implementation details and evaluation results. Section 7 describes the limitations of LAHMOL, future works and then concludes.

## 2 BACKGROUND AND MOTIVATION

### 2.1 Time-Sensitive Networking

Time-Sensitive Networking (TSN) is a set of standards under development by the Time-Sensitive Networking task group of the IEEE 802.1 working group**??**. The standards define mechanisms for the time-sensitive transmission of data over deterministic Ethernet networks and can be divided into three basic key components: Time synchronization, Scheduling and traffic shaping, Selection of communication paths. No matter whether in wired or wireless environment, time synchronization has a long history, and there are a lot of mature techniques, like [TODO: REF]. And selection of communication paths is mainly about path reservations and fault-tolerance, ensuring the reliability of the transmission. Because our work is not focused on time synchronization and reliability of the transmission, these two components will not be introduced further.

Scheduling and traffic shaping allows for the coexistence of different traffic with different priorities on the same network. IEEE 802.1Q provides different requirements to available bandwidth and end-to-end latency for traffic with different priorities. As an enhancement to Traffic Scheduling, IEEE 802.1Qbv uses Time-Aware scheduler to achieve transmission times with guaranteed end-to-end latency. The IEEE 802.1Qbv time-aware scheduler is designed to separate the communication on the Ethernet network into fixed length, repeating time cycles. Within these cycles, different time slices are assigned to different priorities. By doing this, for those traffic classes that need transmission guarantees, it is possible to grant exclusive use of transmission medium for a limited time. The basic concept is a time-division multiple access (TDMA) scheme. By establishing virtual communication channels for specific time periods, time-critical communication can be separated from non-critical background traffic. In TSN, time-critical traffic is given a high priority and non-critical background traffic are given low priorities. One example for an IEEE 802.1Qbv scheduler configuration is visible in figure 1. In this example, each cycle consists of two time slices. Time slice 1 only allows the transmission of traffic tagged with a high priority, and time slice 2 in each cycle allows for low priorities to be sent.

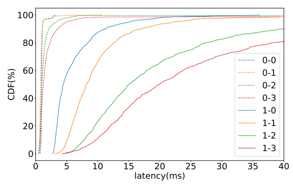

**Figure 1: Example of IEEE 802.1Qbv schedule and frame preemption**

When an Ethernet interface has started the transmission of a frame to the transmission medium, this transmission has to be completely finished before another transmission can take place. Just before the end of time slice 2 in cycle n, a new frame transmission is started. Unfortunately, this frame is too large to fit into its time slice. Since the transmission of this frame cannot be interrupted, the frame infringes the following time slice 1 of the next cycle n+1. So such frame that may block the next time slice of high priority should be avoided. The time-aware scheduler achieves this by putting a guard band in front of every time slice that carries time-critical traffic. During this guard band time, no new Ethernet frame transmission may be started, only already ongoing transmissions may be finished. However, they also have some significant drawbacks:the time that is consumed by a guard band is lost and a single time slice can never be configured smaller than the size of the guard band.

To further mitigate the negative effects from the guard bands, the IEEE working groups 802.1Qbu have specified the frame preemption technology. Frame preemption defines two MAC services for an egress port, preemptable MAC (pMAC) and express MAC (eMAC). Express frames can interrupt transmission of preemptable frames. On resume, MAC merge sublayer re-assembles frame fragments in the next bridge. Figure gives a basic example how frame preemption works. During the process of sending a low priority frame, the MAC interrupts the frame transmission just before the start of the guard band. The partial frame is completed with a CRC and will be stored in the next switch to wait for the second part of the frame to arrive. After the high priority traffic in time slice 1 has passed and the cycle switches back to time slice 2, the interrupted frame transmission is resumed. Frame preemption allows for a significant reduction of the guard band and minimizes the bandwidth that is lost.

## 2.2 Wireless TSN

Wireless TSN aspects are currently under development within the cellular (i.e., 5G) as well as the non-cellular (i.e., IEEE 802.11be) wireless communication community. The authors in **??** have already shown which functionalities a wireless TSN system must provide and which aspects of wireless TSN are already part of standardization, especially within IEEE 802.11be. However, there are challenges in moving TSN directly from an wired network to a wireless network.

Time synchronization in a wireless network is the first problem. Time synchronization in a wired network is usually achieved using the IEEE 802.1AS standard, which can already be operated over IEEE 802.11 by means of the timing measurement (TM) procedure defined in IEEE 802.11v, which takes wireless link asymmetric delay into consideration. In a future version of the standard, the Fine Timing Measurement (FTM) method described in IEEE 802.11mc is intended to be used so to achieve more accurate time synchronization. It can provide 0.1 ns of timestamp resolution, far more accurate than TM, whose timestamp resolution is 10 ns. Besides, there are many related work for the precise synchronization of wireless networks in order to extend TSN to wireless networks. for example, [TODO: REF] As these efforts continue, time synchronization no longer seems to be a challenge blocking the development of wireless TSN.

However, this is only the first step towards wireless TSN, and there are other challenges that need to be addressed. Unreliability of wireless transmission brings unpredictability of wireless transmission time. TSN requires to eliminate factors of uncertainty on the transmission paths, such as channel competition or collisions. This is done via allocating dedicated slots in time to senders, where only this sender is allowed to access the transmission medium. But, because of unreliability of wireless transmission, it is impossible to predict whether a retransmission will occur. In addition, wireless transmission requires preemption of the channel before transmission. When multiple wireless devices transmit simultaneously, it is difficult to predict when the device get the permission of transmission medium. All these make the transmission time unpredictable.

What is more, the working principle of TSN and wireless transmission mechanism are not well coupled. The working principle of wired transimision is Carrier Sense Multiple Access / Collision Detection (CSMA/CD),operating by detecting the occurrence of a collision. It senses of the shared channel is busy for broadcasting and interrupts the broadcast until the channel is free. In CSMA/CD collision is detected by broadcast sensing from the other stations. Upon collision detection in CSMA/CD, the transmission is stopped and a jam signal is sent by the stations and then the station waits for a random time context before retransmission. And the working principle of wireless transmission is Carrier Sense Multiple Access / Collision Avoidance (CSMA/CA), hencing reduces possibility of collision be avoiding collision from happening. Unlike CSMA/CD which is effective after a collision, CSMA/CA is effective before a collision. The essence of TSN is to make other background traffic give way to time-critical traffic. Therefore, in a wired network scenario, in order to avoid conflicts, it is necessary to know the transmission time of high-priority streams in advance. And since wireless transmissions can already know the occurrence of conflicts in advance, there is no need to conduct scheduling in advance and the only thing that needs to be done is how to handle the conflict. [TODO:polish]

Actually, there is also some other work on wireless TSN,like SchedWiFi [18]. It achieves low latency by separating latency-sensitive flows from each other and all the bandwidth-hungry flows. The former is done by offsetting the transmission of latency-sensitive flows with an off-line scheduler. To achieve the latter, it assigns time window named ST-Window for each latency-sensitive transmission, and disables transmission of bandwidth-hungry flows from all the nodes for each ST-Window with a MAC layer component named time-aware shaper. However, such design of disabling transmission of bandwidth-hungry flows from all the nodes dramatically sacrifices throughput.

## 2.3 Other Related Work for Low Latency

Except TSN, there are also some other work to maintain the balance between latency-sensitive flows and bandwidth-Hungry flows. Depending on implementation method, We divide them into three categories. The first category is modifying MAC protocols.

The second one is modifying ADHOC.

The last one is modifying WiFi operation system.

[TODO:]

By further introducing a case of each category, we show the differences between them and LAHMOL

*ROS 2 and Linux-RT.* ROS 2 [6] is implemented on Linux-RT [7] and they are two major efforts on the software level towards a real-time operating system. They allow tasks and applications to be executed almost exactly at the scheduled time by preemption. However, they mainly focus on real-time execution and end-to-end communication latency is not their main concern. From the perspective of flows, these real-time operating systems can only guarantee the flows are delivered to the network layers timely. Although ROS2 adopted fast Real Time Publish Subscribe protocol that reduces the end-to-end latency, the contention and the HoL blocking are not taken into account. To conclude, these two systems enforce real-time execution on the application layer and have no major benefit on the end-to-end communication latency. They are orthogonal to LAHMOL and thus are not the comparison targets of LAHMOL.

*EDCA.* Enhanced Distributed Channel Access (EDCA) [?] is part of the IEEE802.11 standard. It differentiates the traffic into four Access Categories (ACs),namely ACVO, ACVI, ACBE and ACBK, with priority from high to low.Time sensitive applications like VoIP and video streaming can specify the prior-ity of their traffic so as to get higher chance of being sent first when contendingfor the wireless channel. Unfortunately, as will be illustrated in Figure 1, highpriority traffic, which usually consumes relatively small bandwidth, still suffersfrom HoL blocking when coexisting with bandwidth-hungry traffic. As described above, EDCA [19] is a classic way to enhance end-to-end latency of a latency-sensitive flow by assigning a higher probability for high priority flows to acquire the channel in contention. When there is no bandwidth-hungry flows on the same machine contending for the WNIC, robots with only EDCA achieved comparative end-to-end latency as if there is no bandwidth-hungry flows from other robots.

*Pound.* Pound [20] is a ROS node for reducing delay and jitter in wireless multi-robot systems. It provides a priority scheme for registered traffic flows from other ROS nodes by maintaining a queue at the application level. The registered flows from other ROS nodes are intercepted and gathered in the Pound queue before they are enqueued into the network queues of the operating system. Pound then sorts the packets according to their priority, arrival time and expected transmission time. Specifically, if the transmission time of two packets overlaps, the one with lower priority will be placed behind the one with higher priority, so that the high priority packet will not be blocked by this low priority packet when it enters the network queues. To avoid congesting the network, it waits for a interval based on a pre-configured transmission rate and delay each time it delivers a packet to the OS.

While Pound is able to ensure low latency for high priority traffic, the design of waiting for each single packet transmission fails to leverage the frame aggregation technique, thus sacrifices much throughput. It can not be fixed by shrinking the time to wait, since this again causes packets to accumulate in the OS transmission queue, which in turn invalidates the design of Pound, brings back HoL blocking and increases latency.

### 2.4 Key observation

In wired TSN, when both local flows and non-local ones are transmitted to a certain device through the switch, the packets are buffered before they can be forwarded, congesting the switch. However, such buffering does not exist in wireless networks since no switch is involved. In wireless networks, non-local flows cause congestion by Occupying wireless shared medium for transmission.

[TODO: wireless vs wired mechism]

We further observe that for the most widely used priority-aware channel access mechanism in wireless networks, typically enhanced distributed channel access (EDCA), contention intensity is the key element for EDCA to guarantee a prioritized channel access as required by wireless TSN. When contention gets intensive by introducing multiple bandwidth-hungry flows to contend for the channel simultaneously, the latency of latency-sensitive flows increases because bandwidth-hungry flows get higher possibility to acquire the channel.

Although many WNICs provide standalone hardware queue for each AC in EDCA, the WNIC has to finish ongoing transmission before serving new coming packets due to the complexity of wireless transmission.

[TODO: HOL]

Thus EDCA still fails to meet the latency requirements although it can mitigate HoL blocking to some extent. Besides, some scheduling has to be determined among the hardware queues.

Now suppose there is a bandwidth-hungry flow sending large amount of packets. The WNIC will be busy transmitting packets most of the time. When some packets from latency-sensitive flows come, they have to wait at least until the WNIC finish the previous transmission, which counts into the latency. What's worse, the WNIC may schedule the low-priority queue again after transmitting some of the enqueued high-priority packets, making the remaining wait longer, and the latency of the whole message suffers. What's even worse, when there are multiple wireless devices sending bulk data simultaneously, the contention among them increases the time required for each transmission, which further hurts latency-sensitive flows. And this is exactly what happens in multi-robot online learning: all the robots may transmit bulk data like parameter updates while sending latency-sensitive messages frequently. [TODO: ROBOT?]

We conduct an experiment to illustrate the presence of HoL blocking. The topology of the machines follows Figure **??** with one leader and four workers. We monitor the latency of 512B sized messages sent from a worker to the leader under different settings: settings where there are no bandwidth-hungry flow from local machine and 0 to 3 bandwidth-hungry flows from different machines are denoted as 0-0, 0-1, 0-2 and 0-3; settings where there are one bandwidth-hungry flow from local machine and 0 to 3 bandwidth-hungry flows from different machines are denoted as 1-0, 1-1, 1-2 and 1-3. The 512B messages go through AC_VO, the highest priority in EDCA. The bandwidth-hungry flows go through AC_BE, the default priority in EDCA.

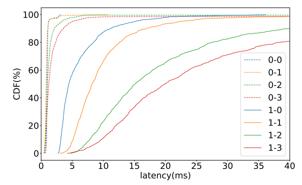

**Figure 2: Latency CDF under different settings.**

As can be seen in Figure 2, the latency is much lower and almost the same when there is no bandwidth-hungry traffic from local machine (0-0, 0-1, 0-2 and 0-3). This confirms that EDCA is quite effective in granting priority for traffic from different machines. Unfortunately, when there is bandwidth-hungry traffic on local machine (1-0, 1-1, 1-2 and 1-3), the latency dramatically increases, which means the latency-sensitive traffic is blocked by the bandwidth-hungry traffic although they should be served first.

In a word, EDCA effectively ensures low latency when light-weight, typically one, bandwidth-hungry flow is running. Based on this observation, if we control the contention intensity by allowing light-weight, typically one, bandwidth-hungry flow to contend with latency-sensitive flows, EDCA can well guarantee that latency-sensitive flows have higher channel access priority and they enjoy latency almost as low as a free channel and the overall throughput does not degrade. So, A priority-aware channel access mechanism to control contention intensity is necessary for the enhanced channel access mechanism.

## 3 SYSTEM OVERVIEW

### 3.1 System Model & LT-SLA

Lahmol targets the scenarios where multiple mobile robots work cooperatively for some specific task. These robots form a wireless device-to-device network to communicate with each other. Nearby robots communicate directly without the relay of other robots. For robots far away from each other, the underlying protocol (e.g. IEEE 802.11s [9]) finds appropriate route to relay the communication. Further, we assume that any two robots that share latency-sensitive data communicate directly without relay $mostofthetime$, since the increase of hops inevitably increases latency and is out of the scope of this paper.

To cooperate with each other, various kinds of messages are transmitted over the network. Each kind of message between two robots, e.g. a series of video frames from robot A to robot B to share the visual perception, will be referred to as a flow. Different flows have different requirements. For example, real-time sensor data and locations are shared and used for cooperative navigation [11, 13, 17], and have stringent latency requirements. In collaborative Simultaneous Localization And Mapping (SLAM), a typical application in robotics, robots share and merge their submaps to get a global map [10, 12, 15]. Reinforcement learning is being widely adopted [11, 14, 17], and multiple robots may train or fine-tune the model cooperatively for better performance when they enter a new environment. These map data and network model parameters are generally large, thus the throughput, i.e. number of bytes transmitted per second, is the major focus.

In the real deployment of a multi-robot group, various flows coexist to support different functionalities like performing cooperative navigation and fine-tuning pre-trained reinforcement learning models in new environment. While TSN promises low transmission latency in a general sense, to enable users to *quantitatively* and *intuitively* describe the requirements of different flows in a multi-robot group, we propose Latency-Throughput Service Level Agreements (LT-SLAs) as follows:

(1) Latency SLA: A customizable requirement for the latency of messages in a flow. For the $i$th flow, it is formalized as
$$P(D_i > T_i) < C_i \tag{1}$$
where $D_i$ denotes the latency, i.e. the time it takes for a message to be received by the receiver. $T_i$ denotes a latency bound, and $C_i$ denotes a tolerance bound. Such a requirement means that the probability for the latency to exceed the latency bound $T_i$ should be lower than the tolerance bound $C_i$.
(2) Throughput SLA: The requirements for the throughput of a flow. It is not customizable and is implicitly assumed to be maximizing the throughput, namely number of bytes transmitted per second, for any flow that does not have a latency SLA.

With LT-SLAs, the administrator of a multi-robot group only need to give intuitive parameters for flows that require low latency. For example, setting $T = 10ms$ and $C = 0.05$ in Equation 1 specifies that the probability for the latency of some flow to exceed 10ms should be lower than 5%. This is both *intuitive* to the user for ease of configuration, and *quantitative* to the system for better scheduling, which will be shown in the design of LAHMOL.

For ease of discussion, we refer to flows with latency SLAs as LL (Low Latency) flows, and flows with throughput SLAs as HT (High Throughput) flows in the remaining of this paper. Messages in a flow will eventually be transmitted as one or more packets on the network. The messages and packets originating from these flows are referred to as LL/HT messages/packets correspondingly.

## 3.2 Overview of LAHMOL

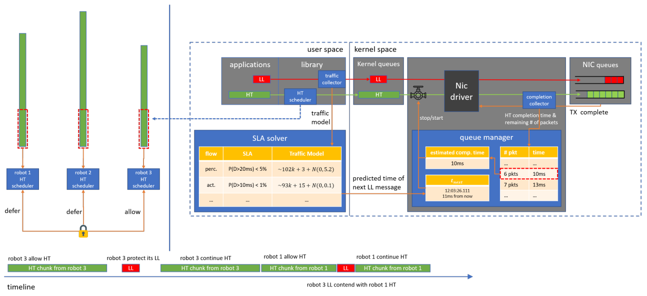

**Figure 3: The architecture of LAHMOL.[TODO: find a better way to show the global part.]**

The goal of LAHMOL is to bring the low latency feature of TSN into WiFi with commodity WNICs, and is further formalized as enforcing the LT-SLAs. LAHMOL enforces the latency SLAs by protecting the transmission of LL flows from HT flows such that when an LL flow sends some packets, 1. there is no local contention so that the WNIC can start transmitting LL packets immediately, 2. there is limited global contention so that the transmission can be effectively prioritized by EDCA with minimal coordination among robots. LAHMOL enforces the throughput SLAs by sending as many HT packets as possible without violating latency SLAs.

LAHMOL consists of four key components to realize the above strategy: *queue manager*, *traffic collector* and *SLA solver* for resolving local contention, and *HT scheduler* for resolving global contention.

***Queue Manager.*** The queue manager provides a software-only and WNIC-independent implementation of prioritized transmission to mitigate the local contention caused by HT traffic. Ideally, the WNIC hardware should support preemptive transmission, namely when any prioritized packets are enqueued to the hardware queue, the WNIC should interrupt the ongoing transmission of non-prioritized packets to guarantee low latency. However, existing WiFi protocol, namely IEEE 802.11b/g/a/n/ac/ax, never specified such preemptive behavior, thus it is normal that existing commodity WNICs do not support. What's more, to the best of our knowledge, wireless transmission involves complex pipelines of hardware components with FIFO queues as intermediate components [2, 3]. Preempting the transmission requires insertion to the head of the FIFO and cleaning up the transmission pipeline, involving redesign and remanufacture of the WNIC hardware. Consequently, preemptive transmission cannot be supported by existing commodity WNICs.

Indeed, one can not control the deep hardware queue directly without modifying the hardware logic. However, the enqueuing to the hardware queue is controlled by the operating system and the driver. If we foresee that there will be prioritized transmission at some time point $t$, we can limit the number of HT packets enqueued in the hardware queue so that they are highly likely to have been finished before $t$. When the prioritized transmission actually happens, the WNIC will not be busy dealing with HT packets. This not only avoids latency increase caused by local contention, but also retains high throughput since the transmission of HT packets is paused for a minimal amount of time. There are two main problems to be addressed to realize this idea.

First, it is nontrivial to determine the number of packets that can be finished before a given time point. We estimate it using recent completion events of packet transmission reported by the WNIC. The queue manager monitors enqueue and completion of the hardware queue by intercepting related driver code, and maintains a table which we call *Completion Time Table* (CTT) to estimate the time required for completing a certain number of packets in HT queue.

Second, the time point of future LL transmission has to be known before it actually happens. In LAHMOL, the functionality of predicting future transmissions and prioritizing the transmission are decoupled to increase modularity and allow for developing more advanced prediction methods. The queue manager leaves the complexity of prediction to the traffic collector and SLA solver, and provides two abstract operation *Inform* and *Relax* for them. The *Inform* operation takes a future time point $t_{next}$ as parameter, and tells the queue manager to make sure that all the packets in HT queue will be finished before $t_{next}$. The queue manager supports this operation by dynamically limiting the number of enqueued HT packets according to the aforementioned CTT, and prevents enqueueing any packet into hardware HT queue after $t_{next}$. The *Relax* operation puts an end to the previous *Inform* operation and recovers the normal enqueue operation of HT packets.

To effectively leverage the operations provided by the queue manager, the application has to call the *Inform* operation early enough before its actual transmission so that the queue manager gets enough time to react, requiring precise prediction of future transmissions. This is generally impossible for arbitrary applications. Fortunately, latency sensitive traffic in robotic applications are often periodic [20], making it possible to predict future transmissions. However, letting applications call such operations by themselves is not desired. Though the traffic is logically periodic, there are random fluctuations in the actual transmission time point caused by various factors like imperfect clock source and system scheduling. Thus, the interval between calling *Inform* and *Relax* should be large enough so that the random fluctuations are handled. Meanwhile, the interval between them should also be small enough to minimize the influence to the HT transmissions. To leverage the operations properly, the application has to learn about the random factors so as to determine when to call *Inform* and when to call *Relax*. This will bring much burden to the developers and increase the complexity of the application.

LAHMOL addresses this problem with by proving traffic collector and SLA solver, introduced as follows.

***Traffic Collector.*** The traffic collector generates traffic models for predicting when an LL flow will send messages. Since LL flows are typically periodic, the time point of each transmission can be characterized as the time point of the first transmission, or offset for short, plus an integer multiple of the period. However, due to random fluctuations caused by imperfect clock source, scheduling, etc., there are differences between the ideal time points and the actual ones. To account for the difference, the traffic collector keeps a certain number of recent transmissions, and runs linear regression to estimate three parameters, namely the transmission period, offset, and the variance of a normal distribution residual with mean 0. With such models, the probability for the actual transmission time point to be within a range around the expected one can be estimated, serving as the basis for controlling the queue manager accordingly.

***SLA Solver.*** The SLA solver enables the user to configure the LT-SLAs described in Section 3.1, and calls the operations provided by the queue manager according to the LT-SLAs and traffic models reported by traffic collectors.

To resolve local contention, one just need to call *Inform* with the estimated time point of future transmission, and call *Relax* after the transmission. The interval between *Inform* and *Relax* should be long enough to cover the error of prediction, but is yet to be determined. With a model of the probability distribution of the error between the predicted time point and the actual one, we can specify the desired portion $p_{prot}$, e.g. 95%, to care about, and map it into some time interval $(t_a, t_b)$ according to the probability distribution and the predicted time point. By calling *Inform* at $t_a$ and *Relax* at $t_b$, the LL packets that are actually sent between $t_a$ and $t_b$ are protected from the contention caused by HT flows.

The exact value of $p_{prot}$ is dependent to the latency SLA and the characteristics of the wireless network condition. With LAHMOL, each LL transmission has probability $p_{prot}$ to encounter no local contention, and $(1 - p_{prot})$ to encounter local contention. The overall latency distribution is a probabilistic combination of the distributions under these two cases. By acquiring latency distribution models in these two cases with offline tests, the SLA solver solves the value of $p_{prot}$ that satisfies the latency SLA. Section 4.5 gives the detailed mathematical rationale.

Further, when the required latency distribution models are unavailable, the SLA solver works in a fallback mode that uses a pre-configured $p_{prot}$.

***HT Scheduler.*** As is shown in ???, EDCA is effective only when the contention level is low. To limit the global contention level, LAHMOL limits the number of robots that send HT traffic simultaneously with minimal coordination cost.

For this purpose, a straightforward approach is to maintain a conceptual mutex across the network, and a robot is allowed to send HT traffic only if it has acquired the mutex. The drawback is that it severely hurts the fairness, since a robot with more data to send will hold the mutex longer. A simple fix would be to send a fixed size chunk, like 20MB, during each acquisition of the mutex. However, due to the high mobility in our targeted scenario, the available bandwidth may vary and drop dramatically, again making it possible to take a long time for some robot to finish its transmission of the chunk.

To fully address this problem, LAHMOL applies time division such that each acquisition of the mutex is only valid for a fixed time slice like 10 seconds. The mutex will be automatically revoked after the time slice. Also, the robot proactively releases the mutex if it finishes the transmission before the end of the time slice to enable others to utilize the unused opportunity. The implementation faces technical challenges due to the asynchronous design of POSIX socket APIs, and will be explained in ???.

Note that LAHMOL is independent to the implementation of the mutex. For ease of development, in this paper we implemented it in a centralized manner. Further, the mutex can be generalized to allow more than one simultaneous transmission, but in this paper we limit it to one for lowest contention level and lowest latency.

## 3.3 Example Workflow

To help understanding, we explain the example workflow of LAHMOL shown in Figure 3.

In Figure 3, the SLA solver is configured with two latency SLAs: no more than 5% of the latency of perception messages can exceed 10ms, and no more than 1% of the latency of control messages can exceed 5ms. With such configuration, each time the SLA solver receives a traffic model from a traffic collector, it determines an interval around each predicted transmission time point.

[TODO: this part is not finished.] There is only 11ms until next transmission of some LL flow and there are already 6 packets in the HT queue. By looking up the CTT, it can be found that enqueueing more packets will take at least 13ms to complete, thus corresponding kernel queues are stopped to fulfill the semantics of the latest *Inform* or *Renew* operation.

## 4 DETAILED DESIGN

In this section, we describe the design details of the four components of LAHMOL: queue manager, traffic collector, SLA solver and HT scheduler. Moreover, we present the mathematical rationale of the SLA solver in Section 4.5.

## 4.1 Queue Manager

The queue manager controls the process of WNIC driver dequeueing HT packets from upper kernel queues and enqueueing them to the WNIC queues, according to the *Inform* operations from the SLA solver, the current depth of the HT queue, and the completion time of each HT packet (i.e. the time between when it is enqueued into the WNIC queue and its completion callback is called), such that whenever the WNIC driver is handling the packet of LL messages, the HT queue is empty from the driver's perspective and the LL messages can be transmitted without being blocked by other packets.

As was mentioned earlier, the queue manager maintains a CTT that is used to predict the required time of transmitting specific numbers of packets in the HT queue. The CTT contains rows with form $(n, t)$ meaning that it takes time $t$ to complete the transmission of $n$ packets in the HT queue. To maintain CTT, the queue manager intercepts into both the enqueue and the packet completion callback of the WNIC driver. Each time the driver enqueues an HT packet into the WNIC's HT queue, the enqueue time $t_{enq}$ of the packet is recorded, along with the current number of packets $n_{pkt}$ in the WNIC's HT queue. Each time the packet transmission is completed, at time $t_{comp}$, we learn that the HT queue needs $(t_{comp} - t_{enq})$ time to complete the transmission of a packet with $n_{pkt}$ packets already queued in the WNIC queue. Given that the completion time varies due to instability and contention in the wireless channel, we keep a number of the most recent records for each $n_{pkt}$ ranging from 0 to the maximum queue depth of the WNIC, and take the max as the estimated transmission time.

After each call to *Inform* with parameter $t$ by the SLA solver, when the driver attempts to dequeue an HT packet from upper kernel HT queues, the queue manager looks up the CTT for the expected transmission time of this HT packet with the number of queued HT packets in WNIC. Once the CTT reports a time longer than the difference from now to the informed $t$, the queue manager prevent the driver from dequeuing HT packets by stopping the upper HT queues. When *Relax* is called, the queue manager wakes the upper HT queues to resume the HT flows.

[TODO: pseudo code to show how things are handled exactly when 1. maintain the CTT 2. dequeue.]

```
Algorithm 1: Maintain the CTT
  Input: pkt: the packet completed by the WNIC
```

```
Algorithm 2: Dequeue
  Input:
```

## 4.2 Traffic Collector

Suppose an LL flow started from time $q$ and its period is $p$, then the ideal time point of sending the $k$th ($k = 0, 1, 2, ...$) message should be $\hat{t}_k = pk + q$. However, due to random fluctuations, there will be difference between the actual time $t_k$ and $\hat{t}_k$. We assume it to be a normal distribution $N(0, \sigma^2)$, leading to:

$$t_k \sim N(pk + q, \sigma^2), k = 0, 1, 2...$$

To infer the parameters $p$, $q$ and $\sigma$ automatically, each time an LL flow sends a message, the traffic collector records the time point as a sample. With enough samples, the traffic collector runs linear regression on these samples to estimate corresponding $p$, $q$ and $\sigma$. Then it reports the parameters, i.e. $(p, q, \sigma)$ to the SLA solver.

Since the inferred parameters may become inaccurate over time, the traffic collector recomputes and reports the model in two conditions: 1) the difference between the predicted time and the actual time of a message exceeds $2\sigma$; 2) a statically configured interval has passed since last report.

## 4.3 SLA Solver

The SLA solver is configured with a set of latency SLAs $P(D_i > T_i) < C_i$. For the $i$th latency SLA with parameters $T_i$ and $C_i$, let $D_{free,i}$ be the latency distribution for the messages of the $i$th flow when there is no local contention, and $D_{busy,i}$ be the latency distribution when there is local contention. With the parameters $(p_i, q_i, \sigma_i)$ reported by the traffic collector, at any time $t$, it is straightforward to compute the nearest future time point of transmission as [TODO: finish the eqn.]

$$\hat{t} = \tag{2}$$

The parameter of the *Inform* operation $t_{next}$ should be

$$t_{next} = \tag{3}$$

And the time point to call *Relax* should be

$$t_{relax} = \tag{4}$$

In case the latency distributions $D_{busy,i}$ and $D_{free,i}$ are not provided and the $p_{prot}$ is configured with a fixed value, these two becomes

$$t'_{next} = \tag{5}$$

and

$$t'_{relax} = \tag{6}$$

The equations above will be justified in Section 4.5.

[TOOD: how are multiple traffic models handled?]

As is depicted in Figure 4, during run time, the SLA solver will get a set of traffic models for different LL flows on the local machine. After solving the corresponding parameter $\Delta b$ from the SLAs, each of them can be viewed as a set of PWs on the timeline. Each PW $(l, r)$ indicates LAHMOL should keep the HT queue clean from time $l$ until time $r$. It is straightforward that any intersecting PWs can be merged, leading to a set of overall PWs.

The SLA solver performs *Inform* operation before the beginning of the first PW, which tells the queue manager about the next time point of LL message transmission. For the following PWs, it performs *Renew* operation at each end, which tells the queue manager the end of the previous PW and the begin of the next PW. *Relax* should be performed at the end of the last PW to tell the queue manager there will be no more foreseen future transmission of LL messages so far, which happens when all the LL flows have stopped.

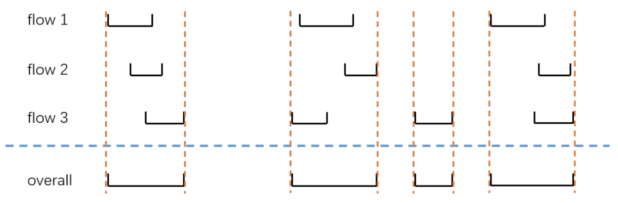

**Figure 4: The PWs processed by SLA solver.**

## 4.4 HT Scheduler

[TODO: the HT scheduler is not that complex, what to write here?]

## 4.5 Solving the SLA

In this section we present the mathematical rationale of how LAHMOL finds out the minimal percentage of messages to protect against Hol blocking and converts the percentage into PWs.

As was mentioned in Section 4.3, LAHMOL assumes two distinct latency distributions for cases with/without HoL blocking. Denote the cumulative distribution function (CDF) as $D_{busy}$ and $D_{free}$, and probability density distribution function (PDF) as $d_{busy}$ and $d_{free}$.

Suppose for each message, there is a probability $p$ that it does not suffer from HoL blocking, and $1 - p$ does. Then the PDF of it follows:

$$d(x) = pd_{free}(x) + (1 - p)d_{busy}(x)$$

And the corresponding CDF is:

$$D(x) = \int_0^x d(x)dx = \int_0^x (p * d_{free}(x) + (1 - p) * d_{busy}(x))dx$$

$$= p * \int_0^x d_{free}(x)dx + (1 - p) * \int_0^x d_{busy}(x)dx = p * D_{free}(x) + (1 - p) * D_{busy}(x)$$

To ensure a latency SLA $P(D_i > T_i) = 1 - D(T_i) \leq C_i$, we have $D(T_i) = p * D_{free}(T_i) + (1 - p) * D_{busy}(T_i) \geq 1 - C_i$, so thers is $p \geq \frac{1-C_i-D_{busy}(T_i)}{D_{free}(T_i)-D_{busy}(T_i)}$. Denote the minimum acceptable p $\frac{1-C_i-D_{busy}(T_i)}{D_{free}(T_i)-D_{busy}(T_i)}$ as $p^*$.

The above solution means that only when $p \geq p^*$, the entire system could satisfy a latency SLA. So LAHMOL has to protect at least a portion $p^*$ of messages from HoL blocking.

As was noted in Section 4.2, the traffic collector is able to predict the future time point of transmission as time $\hat{t}$ with a normal distribution error $N(0, \sigma^2)$. We find out the interval $[\widehat{t} - \Delta b, \widehat{t} + \Delta b]$ so that the probability for the actual time point $t$ within the interval is exactly $p^*$. This is equal to solving the equation $P(|\hat{t} - t| < \Delta b) \geq p^*$.

According to one-variable linear regression, $t_{n+1} \sim N\{\beta_0 + \beta_1 * x_{n+1}, \sigma^2\}, \sigma^2 = \frac{\sum(t_i-\beta_0-\beta_1*x_i)^2}{n}$

$\widehat{t}_{n+1} - t_{n+1} \sim N\{0, \sigma^{*2}\}, \sigma^* = \sigma\sqrt{1 + \frac{1}{n} + \frac{(x_{n+1}-\bar{x})^2}{\sum x_i^2}}$.

$p^* = P(|\widehat{t}_{n+1} - t_{n+1}| < \Delta b) = F(\Delta b) - F(-\Delta b) = \frac{1}{\sigma^* \sqrt{2\pi}} \int_{-\Delta b}^{\Delta b} exp(-\frac{x^2}{2\sigma^{*2}})dx = erf(\frac{\Delta b}{\sqrt{2}\sigma^*}) \geq p_{limit}$

So, $\Delta b \geq \sqrt{2}\sigma^* erf^{-1}(p_{limit})$.

because of SLA2, $max(throuphput) \iff min(\Delta b)$, so we have

$\Delta b = \sqrt{2}\sigma^* erf^{-1}(p_{limit}), p_{limit} = \frac{1-C_i-D_{busy}(T_i)}{D_{free}(T_i)-D_{busy}(T_i)}, \sigma^* = \sigma\sqrt{1 + \frac{1}{n} + \frac{(x_{n+1}-\bar{x})^2}{\sum x_i^2}}$

## 5 IMPLEMENTATION

The queue manager was implemented as a kernel device module with about 600 lines of code based on Linux 5.4.84 with PREEMP_RT patch [7], and provided the *Inform* and *Relax* operation through `ioctl()`. To get the completion time of each packet, the queue manager hooked into two operations of the driver. The operation to enqueue a packet to the hardware queue was hooked to record current number of packets in the hardware queue and the timestamp. The operation to handle packet completion from the hardware queue was hooked to compute the time a packet spent in the hardware queue, and update the CTT as was described in ???. Further, the queue manager hooked into the driver operation of dequeueing packets from mac80211 queues to prevent it from enqueueing too many packets into the hardware queue. Although the exact code of WNIC drivers differ, these three operations are general and necessary. For the evaluation, the hooks were added into the MT76 driver in the kernel source tree with ??? lines of modification, 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()`.

We implemented the traffic collector in the rcl (ROS client library) of ROS2 so as to automatically support a wide range of applications developed by the robotics community. We mapped the notion of flows in LAHMOL as publishers on different topics. This was done by associating a traffic collector for each latency-sensitive flow and hooking the function for publishing messages. For the communication between traffic collectors and the SLA solver, instead of using traditional inter-process communication methods like sockets and shared memory, we chose the ROS2 way, namely publishing the traffic model to a special topic that is subscribed by the SLA solver. This made LAHMOL modular and would enable developing more tools to leverage the information gathered by the traffic collector. It involved around 400 lines of code in the ROS2 rcl.

The SLA solver was implemented as a stand-alone ROS2 application that listens to the topic for traffic model, and controls the queue manager via `ioctl()`.

Ideally the HT scheduler should also be implemented in ROS2. However, currently ROS2 does not provide direct TCP support, and the default UDP support not only limits the size of the message (64KB), but also suffers bandwidth problems in unreliable networks like WiFi [4]. Thus we chose to temporarily implement a reliable message streaming interface with TCP socket, integrated with the HT scheduler. [TODO: maybe some notes about how to implement it with TCP, since it is not trivial.] The global mutex was implemented in a centralized manner for simplicity.

## 6 EVALUATION

Lahmol motivates edge computing and multi-robot applications where the machines or robots are interconnected with wirelessly. To evaluate the effectiveness of Lahmol in the above scenarios, we choose one possible deployment in multi-robot applications, namely cooperative control with decentralized learning. The control part takes as input the sensor data of the robots and generates control commands, requiring low latency of the transmission of sensor data and control commands. The purpose of decentralized learning is to leverage the computing power of the robots and train/fine-tune the control model with collected data, and is the source of high-throughput traffic. There are both centralized control and decentralized control in the robotics research community. Centralized control is generally considered prone to single-unit failure, while decentralized control incurs significant communication overhead. There are also research advances in eliminating explicit communication in decentralized control. Comparing these different paradigms is out of the scope of this paper. For the evaluation, we mainly focus on centralized control, and decentralized control is presented as an aspect of scalability in Section **??**.

To be specific, the workload consists of two parts: control and online learning. The control part can be centralized or decentralized.

*Centralized Control.* A multi-perception application with online distributed learning was developed. The application ran a number of instances distributedly on each host, which we refer to as leader and worker. The leader ran on the desktop that created the hotspot mentioned above, collected perception data from the workers and sent action data to corresponding workers. The workers ran on each of these hosts, kept sending perception data to the leader and receiving action data from the leader, and performed data-parallelism distributed learning in the meantime. The perception data was sampled from the Robonet dataset[5]. To measure the latency of perception and action, the perception data was prepared from the leader with a local timestamp, then it was sent via the Ethernet link so that the latency overhead is minimal and stable. The workers sent the perception data back to the leader via wireless connection right after the reception. The traffic of action data followed a similar pattern, except that they were first sent to the workers via wireless connection, then sent back to the leader via Ethernet connection, because in a real deployment the actions originate from the leader while the perceptions originate from the workers.

*Decentralized Control.*

*Decentralized Learning.* Our evaluation was done using 5 hosts running Ubuntu 20.04 on Linux 5.18.4 patched with Preemp-RT with ROS and ROS2 installed. Two of them were desktops with Intel Core i7-8700 CPU @ 3.20GHz, and three of them were laptops with different models of CPUs, namely Intel Core i7-10510 CPU @ 1.80GHz, i5-7200U CPU @ 2.50GHz and i7-7700HQ CPU @ 2.80GHz respectively. All of them were equipped with MediaTek MT7612U USB WNICs and configured to IBSS mode on channel 36 (5GHz) with 80MHz bandwidth. Additionally, batman-adv [] was configured on all these WNICs so that multi-hop communication can be automatically established in the evaluation of mobility presented in Section **??**. For the measurement of latency, the hosts were connected to a TP-LINK TL-SG108 desktop Ethernet switch, and their system clocks were synchronized with PTP []. The synchronization accuracy was reported to be below 10us by ptp4l, sufficient for measuring wireless latency which is usually above 1ms.

We compare Lahmol with systems that fulfill two requirements. First, they should be designed to achieve optimal latency for prioritized traffic. Second, they should require no modification to the WiFi protocol so as to be practical on commodity WNICs. Thus Pound and SchedWiFi are chosen among related works.

Since Pound was developed for ROS1, we backported the application to ROS1 for its evaluation. The transmission rate was set to 200Mbps according to the speed test result. The delay parameter was configured as 41 microseconds following the calculation in their paper.

SchedWiFi [18] was only implemented and evaluated in simulations in the original paper. To compare its actual performance with Lahmol, we realized SchedWiFi by implementing all its features. ST-Window (STW) to be reserved for each LL flow is calculated through Equation 7, where $L$ is the data length of each message in an LL flow, $SIFS$ is $16\mu s$ according to the standards of IEEE 802.11 [8] and the maximum retransmission time $R$ is set to 7 [1]. $TX_{ack}$ is empirically set to $30\mu s$. SchedWiFi claimed to have the feature of Time-Aware Shaper implemented in the physical layer, which could isolate the transmission of LL flows from HT flows on a single machine but did not demonstrate more details of the design. More importantly, modern WNICs offload most of the MAC layer operations to their closed-source firmware, making it impractical to modify related logic. To achieve the similar effect, we use HT scheduler and queue manager as its Time-Aware Shaper.

We also compare with two additional configurations, referred to as WiFi and EDCA, to give an assessment on the original WiFi. By WiFi we refer to the default configuration of running the application on ROS2, where all the traffic including perception, action and parameter update has the same priority AC_BE by default. For EDCA, the perception and action traffic was specifically configured to use the highest priority AC_VO for low latency.

$$STW = 2\times 25\mu s + (\frac{L}{Bandwidth} + 2\times SIFS + TX_{ack})\times(1+R) \quad (7)$$

We evaluated LAHMOL against WiFi, EDCA, Pound, Pound-HT and SchedWiFi to answer the following questions:

- RQ1: (end-to-end) In terms of throughput and latency, how is LAHMOL compared with existing systems and methods?
- RQ2: (micro-event) Where does the gain of LAHMOL come from?
- RQ3: (mobility) How is the performance LAHMOL to the change of connection quality caused by mobility?
- RQ4: (scalability) How does LAHMOL scale with the number of devices and application instances?

## 6.1 End-to-end Evaluation

## 6.2 Micro-event Analysis

## 6.3 Mobility

*Throughput Drop.*

*Change of Hops.*

## 6.4 Scalability

*Number of Machines.*

*Number of Instances.*

## 7 LIMITATION AND CONCLUSION

## 7.1 Limitations and Future Works

The HT scheduler in LAHMOL eases the contention among bandwidth-hungry flows from different robots and makes the clearance time of WNIC queues comparatively stable and predictable. However, due to the complexity of network status and acceleration techniques like multi-input and multi-output, and frame aggregation in WiFi, the transmission time of a packet still faces minor perturbation. By simply selecting the 99%-th record, queue manager could overestimate or underestimate the transmission time. But as shown in Section 6, the error rate of prediction is small and has no significant impact on solving HoL blocking problem and overall high throughput. A more precise prediction mechanism for queue manager can be one of the future works.

LAHMOL in this paper mainly targets the multi-perception distributed online learning applications on a multi-robot system. However, besides perception and action in our setting, latency-sensitive flows play an important role in many other robotic applications for safety, coordination, continuous control, etc. Many more robotic scenarios where latency-sensitive flows and bandwidth-hungry ones coexist are to be explored and LAHMOL's efficiency on them are to be evaluated in the future.

The HT scheduler of LAHMOL assumes all the robots in the network would contend for the channel with each other, which holds in one-hop networks, like the AP-station topology in our setting. However, in a multi-hop network, not all the robots contend with each other due to the limited transmission range. The exclusive transmission enforced by the HT scheduler ignores the opportunity in multi-hop networks for simultaneous transmission among robots that do not contend with each other. Thus HT scheduler is not efficient enough for multi-hop networks and we leave the improvements for multi-hop networks as one of the future works.

While this paper mainly focus on the latency of periodic traffic, with the modularized design of queue manager, more sophisticated prediction technology can be combined to benefit more applications.

[TODO: handling multi-hop LL?]

## 7.2 Conclusion

In this paper, we identify the simultaneous low-latency and high-throughput demand (i.e., LT-SLA) in multi-robot online learning, and present LAHMOL, the first software level solution to automatically and transparently resolve the HoL blocking problem in WNIC queues and enforce LT-SLAs on modern commodity WNICs. A major feature of LAHMOL is that LAHMOL can be easily transferred to other commodity WNICs, because it is designed on the software level and does not need to modify any internal logic of the WNIC and its driver. Evaluation shows that with LAHMOL integrated, latency-sensitive flows in multi-robot systems for perception, control, coordination, etc., can be transmitted with low latency, while providing high throughput to accommodate bandwidth-hungry flows for training data and parameter updates in distributed learning.

## REFERENCES

[1] [n.d.]. 802.11/MAC/Lower/Retransmissions – WARP Project. https://warpproject.org/trac/wiki/802.11/MAC/Lower/Retransmissions

[2] [n.d.]. AR9280 MAC/BB/Radio Datasheet pdf - MIMO MAC/BB/Radio. Equivalent, Catalog. https://datasheetspdf.com/pdf/705767/Atheros/AR9280/1

[3] [n.d.]. CYW4356/CG8674 Single-Chip 5G WiFi IEEE 802.11ac 2×2 MAC/Baseband/Radio with Integrated Bluetooth 4.1. 002 ([n. d.]), 144.

[4] [n.d.]. DDS tuning information. https://index.ros.org/doc/ros2/Troubleshooting/DDS-tuning/

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

[6] [n.d.]. *ROS 2 Overview.* https://index.ros.org/doc/ros2/

[7] [n.d.]. *RTwiki.* https://rt.wiki.kernel.org/index.php/Main_Page

[8] [n.d.]. Short Interframe Space. https://en.wikipedia.org/w/index.php?title=Short_Interframe_Space&oldid=988080746 Page Version ID: 988080746.

[9] 2021. IEEE 802.11s. https://en.wikipedia.org/w/index.php?title=IEEE_802.11s&oldid=1010283936 Page Version ID: 1010283936.

[10] R. Dubé, A. Gawel, H. Sommer, J. Nieto, R. Siegwart, and C. Cadena. 2017. An online multi-robot SLAM system for 3D LiDARs. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 1004–1011. https://doi.org/10.1109/IROS.2017.8202268 ISSN: 2153-0866.

[11] R. Han, S. Chen, and Q. Hao. 2020. Cooperative Multi-Robot Navigation in Dynamic Environment with Deep Reinforcement Learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 448–454. https://doi.org/10.1109/ICRA40945.2020.9197209 ISSN: 2577-087X.

[12] Fu Li, Shaowu Yang, Xiaodong Yi, and Xuejun Yang. 2018. CORB-SLAM: A Collaborative Visual SLAM System for Multiple Robots. In *Collaborative Computing: Networking, Applications and Worksharing (Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering)*, Imed Romdhani, Lei Shu, Hara Takahiro, Zhangbing Zhou, Timothy Gordon, and Deze Zeng (Eds.). Springer International Publishing, Cham, 480–490. https://doi.org/10.1007/978-3-030-00916-8_45

[13] S. Li, Y. Guo, and B. Bingham. 2014. Multi-robot cooperative control for monitoring and tracking dynamic plumes. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 67–73. https://doi.org/10.1109/ICRA.2014.6906591 ISSN: 1050-4729.

[14] Pinxin Long, Tingxiang Fan, Xinyi Liao, Wenxi Liu, Hao Zhang, and Jia Pan. 2018. Towards Optimally Decentralized Multi-Robot Collision Avoidance via Deep Reinforcement Learning. *arXiv:1709.10082 [cs]* (May 2018). http://arxiv.org/abs/1709.10082 arXiv: 1709.10082.

[15] N. Mahdoui, E. Natalizio, and V. Fremont. 2016. Multi-UAVs network communication study for distributed visual simultaneous localization and mapping. In *2016 International Conference on Computing, Networking and Communications (ICNC)*. 1–5. https://doi.org/10.1109/ICCNC.2016.7440564

[16] Robert Nishihara, Philipp Moritz, Stephanie Wang, Alexey Tumanov, William Paul, Johann Schleier-Smith, Richard Liaw, Mehrdad Niknami, Michael I. Jordan, and Ion Stoica. 2017. Real-Time Machine Learning: The Missing Pieces. *arXiv:1703.03924 [cs]* (May 2017). http://arxiv.org/abs/1703.03924 arXiv: 1703.03924.

[17] Jim Martin Catacora Ocana, Francesco Riccio, Roberto Capobianco, and Daniele Nardi. 2019. Cooperative Multi-Agent Deep Reinforcement Learning in Soccer Domains. (2019), 3.

[18] G. Patti, G. Alderisi, and L. Lo Bello. 2015. 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)*. 1–9. https://doi.org/10.1109/ETFA.2015.7301460 ISSN: 1946-0759.

[19] S. Rashwand and J. Mišić. 2011. IEEE 802.11e EDCA Under Bursty Traffic—How Much TXOP Can Improve Performance. *IEEE Transactions on Vehicular Technology* 60, 3 (March 2011), 1099–1115. https://doi.org/10.1109/TVT.2011.2107928 Conference Name: IEEE Transactions on Vehicular Technology.

[20] Danilo Tardioli, Ramviyas Parasuraman, and Petter Ögren. 2019. Pound: A multi-master ROS node for reducing delay and jitter in wireless multi-robot networks. *Robotics and Autonomous Systems* 111 (Jan. 2019), 73–87. https://doi.org/10.1016/j.robot.2018.10.009