Computation and Communication Co-Design for Real-Time Monitoring and Control in Multi-Agent Systems

Vishrant Tripathi*, Luca Ballotta†, Luca Carlone*, and Eytan Modiano*
*Laboratory for Information and Decision Systems, MIT
†Department of Information Engineering, University of Padova

***Abstract*—We investigate the problem of co-designing computation and communication in a multi-agent system (*e.g.*, a sensor network or a multi-robot team). We consider the realistic setting where each agent acquires sensor data and is capable of local processing before sending updates to a base station, which is in charge of making decisions or monitoring phenomena of interest in real time. Longer processing at an agent leads to more informative updates but also larger delays, giving rise to a *delay-accuracy trade-off* in choosing the right amount of local processing at each agent. We assume that the available communication resources are limited due to interference, bandwidth, and power constraints. Thus, a scheduling policy needs to be designed to suitably share the communication channel among the agents. To that end, we develop a general formulation to jointly optimize the local processing at the agents and the scheduling of transmissions. Our novel formulation leverages the notion of *Age of Information* to quantify the freshness of data and capture the delays caused by computation and communication. We develop efficient resource allocation algorithms using the Whittle index approach and demonstrate our proposed algorithms in two practical applications: multi-agent occupancy grid mapping in time-varying environments, and ride sharing in autonomous vehicle networks. Our experiments show that the proposed co-design approach leads to a substantial performance improvement ($18-82\%$ in our tests).**

***Index Terms*—wireless networks; Age of Information; distributed computing; robotics; networked control systems.**

I. Introduction

Monitoring and control of dynamical systems are fundamental and well-studied problems. Many emerging applications involve performing these tasks over communication networks. Examples include: sensing for IoT applications, control of robot swarms, real-time surveillance, and environmental monitoring by sensor networks. Such systems typically involve multiple agents collecting and sending information to a central entity where data is stored, aggregated, analyzed, and then used to send back control commands. Due to the dramatic improvements both in on-device and edge computing, and in wireless communication over the past two decades, there has been a rapid growth in the size and scale of such networked systems.

This has motivated the design of scalable architectures, both for computation and communication. Two key directions of innovation involve a) pushing the computation to be distributed across the network, such that all agents perform local processing of the collected measurements, and b) designing scheduling algorithms that efficiently share limited communication resources across all devices and ensure timely delivery of information. However, existing work on communication scheduling [1]–[5] disregards distributed processing, while related work on sensor fusion [6]–[8] focuses on designing distributed algorithms, rather than allocating computational resources at each node.

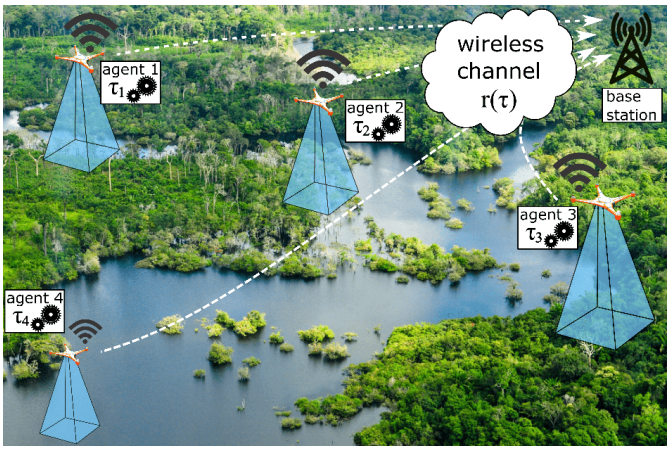

Fig. 1. Example: four drones monitor different regions and send updates to a base station over a wireless channel. Each agent spends time τ_i processing the collected measurements before sending. A scheduling algorithm prioritizes transmissions to the base station. This paper focuses on the co-design of the processing times τ_i and the scheduling policy.

In this work, we explore the joint optimization of computation and communication resources for monitoring and control tasks. We consider a multi-agent system where each agent is in charge of monitoring a time-varying phenomenon and sending information to a central base station. For instance, this setup can model a team of robots mapping a dynamic environment and sending map updates to a base station, which aggregates a global map for centralized decision-making (Fig. 1).

The agents are capable of local processing before transmitting the acquired information. This could involve operations such as refining, denoising, or compressing the data or simply gathering more informative updates. We assume that the more time an agent spends in processing locally, the higher the quality of the generated update. However, longer processing also induces a delay in between subsequent updates. This yields a *delay-accuracy trade-off*: is it better to send outdated but high-quality updates, or to reduce the overall latency by communicating low-quality information?

The first two authors contributed equally to this paper.

This work was funded by NSF Grant CNS-1713725, by Army Research Office (ARO) grant no. W911NF-17-1-0508, by the Office of Naval Research under the ONR RAIDER program (N00014-18-1-2828), by the CARIPARO Foundation Visiting Programme HiPeR, and by the Italian Ministry of Education, University and Research through the PRIN project no. 2017NS9FEY and through the initiative "Departments of Excellence" (Law 232/2016).

We consider the realistic scenario where the total communication resources available are limited due to interference, limited bandwidth, and/or power constraints. Thus, in any given time-slot, only one of the agents is allowed to communicate with the base station. The communication constraints mean that, in addition to optimizing the local processing times, a *scheduling policy* needs to be designed to specify which agents can communicate in every time-slot.

Therefore, the goal of this work is to develop a general framework to determine the optimal amount of local processing at each agent in the network and design a scheduling policy to prioritize communication in order to maximize performance.

Related Work. Over the past few years, there has been a rapidly growing body of work using *Age of Information* (AoI) as a metric for designing scheduling policies in communication networks [3]–[5] and for control-driven tasks in networked control systems [9]–[12]. AoI captures the timeliness of received information at the destination (see [13], [14] for recent surveys). Our processing and scheduling co-design problem is motivated by recent advances in embedded electronics, as well as the development of efficient estimation and inference algorithms for real-time applications on low-powered devices [15]–[18]. The output accuracy of such algorithms increases with the runtime, in line with the delay-accuracy trade-off we consider in this paper. Another application of such a trade-off involves deciding on computation offloading in cloud robotics, which has been the focus of recent works on real-time inference by resource-constrained robots [19], [20]. In this context, sending raw data can induce long transmission delays, but allow better inference by shifting the computational burden to the cloud.

Contributions. We address the computation and communication co-design problem and develop a) a scheduling policy that ensures timely delivery of updates, and b) an algorithm to determine the optimal amount of local processing at each agent. To do so, we use AoI to measure the lag in obtaining information for monitoring and control of time-critical systems. Our contribution is threefold. First, we develop a general framework to jointly optimize computation and communication for real-time monitoring and decision-making (Section II). This framework extends existing work [21] by a) considering joint optimization of scheduling in addition to processing, and b) addressing a general model that goes beyond linear systems.

Second, we develop low-complexity scheduling and processing allocation schemes that perform well in practice (Sections III-IV). The co-design problem is a multi-period resource allocation problem and is hard to solve in general due to its combinatorial nature. We resolve this by considering a Lagrangian relaxation that decouples the problem into multiple single-agent problems, which can be solved effectively. To solve the scheduling problem, we generalize the Whittle index framework proposed in [5] for sources that generate updates at different rates and of different sizes.

Finally, we demonstrate the benefits of using our methods in two practical applications from robotics and autonomous systems: multi-agent occupancy grid mapping in time-varying environments, and ride-sharing systems with local route optimization (Section V). Our simulations show that we can achieve performance improvements of $18-35\%$ in the mapping application and $75-82\%$ in the ride-sharing application with respect to baseline approaches.

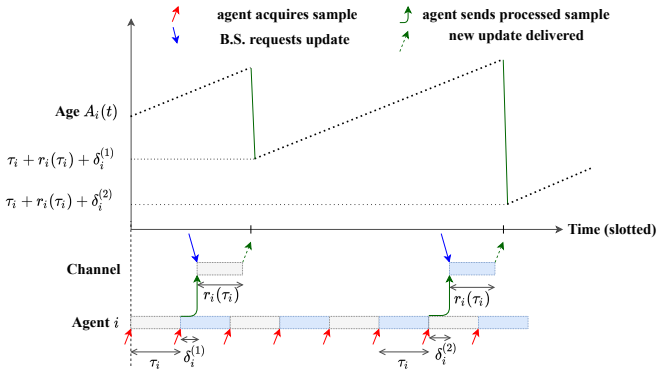

Fig. 2. AoI evolution for agent i. The agent acquires and processes new samples every τ_i time-slots. When the base station (B.S.) requests a new update, the agent sends the most recent sample that has finished processing, taking $r_i(\tau_i)$ time-slots for transmission. The variable $\delta_i^{(k)}$ represents the waiting time in the buffer for update k. Upon a new update delivery, the AoI at the base station $A_i(t)$ drops to the age of the delivered update.

II. Problem Formulation

We consider a discrete-time setting with N agents in a networked system, where each agent is in charge of monitoring a time-varying phenomenon and sending information updates to a base station. Each agent processes the collected measurements locally, before sending its updates. The i-th agent spends τ_i time slots to process a new update. We refer to this quantity as the *processing time* associated with agent i.

We assume that sensing and processing happen sequentially at each agent. Thus, agent i acquires a new sample every τ_i time slots. Further, each agent stores in a buffer the freshest processed measurement. We will assume that the processing time allocations $\tau_i, \forall i$ are constant during operation.

To communicate the acquired and processed updates, the agents use a wireless communication channel. We assume that, due to interference and bandwidth constraints, only one of the agents can transmit to the base station in any given time-slot. At every transmission opportunity, the base station polls one of the agents regarding the state of its system and receives the most recent measurement that has been processed.

Scheduling decisions are modeled as indicator variables $u_i(t)$ where $u_i(t) = 1$ if the i-th agent is scheduled at time t and zero otherwise. We assume that a transmission from the i-th agent takes $r_i(\tau_i)$ time slots, with $r_i(\cdot)$ a monotone sequence. This captures one aspect of the delay-accuracy trade-off, namely that the size of the update depends on the amount of time spent in processing it. When the agents spend local processing to collect more detailed information, *e.g.,* in exploration tasks, the measurements get larger overtime and $r_i(\cdot)$ is increasing. Conversely, when the agents compress the collected data, *e.g.,* extracting visual features from images, $r_i(\cdot)$ is decreasing.

To measure the freshness of the information at the base station, we use a metric called Age of Information (AoI). The AoI $A_i(t)$ measures how old the information at the base station

is regarding agent i at time t. Upon receiving a new update, it drops to the age of the delivered update. Otherwise, it increases linearly. The evolution is described below:

$$A_i(t+1) = \begin{cases} \tau_i + r_i(\tau_i) + \delta_i^{(k)}, & \text{if update } k \text{ is delivered,} \\ A_i(t) + 1, & \text{otherwise.} \end{cases} \tag{1}$$

Here $\delta_i^{(k)}$ is the *waiting time* spent by the k-th update from agent i in the buffer, *i.e.,* the delay from the time the update was processed to the time it was actually transmitted. Since a new processed update is generated every τ_i time-slots, the waiting time $\delta_i^{(k)}$ ranges from 0 to $\tau_i - 1$ time-slots. Note that, as we are dealing with centralized scheduling decisions, the agents do not know in advance when they will be asked for an update. Therefore, under our assumption of constant processing time τ_i, the agents cannot set to zero such $\delta_i^{(k)}$ by just choosing when to sample. Fig. 2 depicts the AoI process for agent i. Observe that the lowest value that the AoI can drop to is $\tau_i + r_i(\tau_i)$ time-slots, since every update spends time τ_i in processing and $r_i(\tau_i)$ in communication.

The AoI evolution in (1) is involved since it requires analyzing waiting times that vary with each update. To simplify the analysis, while still capturing the relevant features of the AoI dynamics, we assume that the sequences $\delta_i^{(k)}$ are constant over time, *i.e.,* $\delta_i^{(k)} \equiv \delta_i \, \forall k$, $\forall i \in \mathcal{V}$. Each δ_i accounts for the *average waiting time* accumulated by a processed measurement before it is sent by the i-th agent. This assumption is reasonable since the waiting time's contribution to the overall AoI is negligible on average (being upper bounded by τ_i) as compared to the time between subsequent requests from the base station, which grows linearly with the number of agents N [3]. The smallest AoI for agent i is defined as $\Delta_i \triangleq \tau_i + r_i(\tau_i) + \delta_i$, which is the value that AoI resets to upon a new update delivery.

It has been shown in recent works [9]–[12] that real-time monitoring error for linear dynamical systems can be seen as an increasing function of the AoI. Intuitively, fresher updates lead to higher monitoring accuracy and better control performance. Motivated by this, we assume that each agent has an associated cost function $J_i(\tau_i, A_i(t))$ that maps the processing time and the current AoI to a cost that reflects how useful the current information at the base station is for monitoring or control.

Assumption 1 (Delay-Accuracy Trade-off)**.** The cost functions $J_i(\tau_i, A_i(t))$ are increasing with the AoI $A_i(t)$ and decreasing with the processing time τ_i. Thus, longer processing leads to more useful measurements (for a fixed age), while fresher information induces a lower cost than outdated information.

Remark 1 (Task-related cost function)*.* The specific functional form of $J_i(\tau_i, A_i(t))$ depends on the underlying dynamics of the system i and on the impact of agent processing on the quality of updates. These functions are typically estimated using domain knowledge or learned from data offline. The approach in this paper holds for any functions $J_i(\tau_i, A_i(t))$ that satisfy the above assumption. We discuss numerical examples in Section V.

Our goal is to design a causal scheduling policy π and find the processing times $\tau_1, ..., \tau_N$ for every agent so as to minimize the sum of the time-average costs.

Problem 1 (Computation and Computation Co-design)**.** Given the set of agents $\mathcal{V} = \{1, \ldots, N\}$, cost functions $\{J_i(\cdot,\cdot)\}_{i\in\mathcal{V}}$, and AoI evolution (1), find the processing times $\{\tau_i\}_{i\in\mathcal{V}}$ and the scheduling policy π that minimize the infinite-horizon time-averaged cost:

$$\begin{aligned} \min_{\substack{\tau_i \in \mathcal{T}_i \, \forall i \in \mathcal{V} \\ \pi \in \Pi}} \quad & \sum_{i\in\mathcal{V}} \limsup_{T\to+\infty} \mathbb{E}_\pi \left[\frac{1}{T} \sum_{t=t_0}^{T} J_i\left(\tau_i, A_i^\pi(t)\right) \right] \\ \text{s.t.} \quad & \sum_{i\in\mathcal{V}} u_i^\pi(t) \leq 1, \forall t \end{aligned} \tag{P1}$$

where Π is the set of causal scheduling policies, $u_i^\pi(t) = 1$ if policy π schedules agent i at time t and $u_i^\pi(t) = 0$ otherwise. $\mathcal{T}_i$ is the set of admissible processing times for agent i, and $A_i^\pi(t)$ is the AoI of the i-th agent at time t under policy π.

Finding the optimal processing times requires iterating over the combinatorial space $\mathcal{T}_i \times ... \times \mathcal{T}_N$, while finding the optimal scheduling policy requires solving a dynamic program which suffers from the curse of dimensionality.

III. A Lagrangian Relaxation

We now discuss a relaxation of Problem 1 that enables us to develop efficient algorithms. This approach is motivated by the work of Whittle [22] and its applications to network scheduling [5]. The relaxation will be useful not only for finding a scheduling policy, but also in optimizing the processing times.

We start by considering a relaxation of (P1) where the scheduling constraint is to be satisfied on average, rather than at each time slot. The relaxed problem is given by

$$\begin{aligned} \min_{\substack{\tau_i \in \mathcal{T}_i \, \forall i \in \mathcal{V} \\ \pi \in \Pi}} \quad & \sum_{i\in\mathcal{V}} \limsup_{T\to+\infty} \mathbb{E}_\pi \left[\frac{1}{T} \sum_{t=t_0}^{T} J_i\left(\tau_i, A_i^\pi(t)\right) \right] \\ \text{s.t.} \quad & \sum_{i\in\mathcal{V}} \limsup_{T\to+\infty} \frac{\sum_{t=t_0}^{T} u_i(t)}{T} \leq 1. \end{aligned} \tag{2}$$

To solve (2), we introduce a Lagrange multiplier $C > 0$ for the average scheduling constraint. The Lagrange optimization is given by the following equation:

$$\max_{C>0} \min_{\substack{\tau_i \in \mathcal{T}_i \, \forall i \in \mathcal{V} \\ \pi \in \Pi}} \sum_{i\in\mathcal{V}} \bar{J}_i(\tau_i, C) - C \tag{3}$$

$$\bar{J}_i(\tau_i, C) \triangleq \limsup_{T\to+\infty} \mathbb{E}_\pi \left[\frac{1}{T} \sum_{t=t_0}^{T} \left(J_i\left(\tau_i, A_i^\pi(t)\right) + C u_i(t) \right) \right]$$

Due to the Lagrangian relaxation, the inner minimization can be decoupled as the sum of N independent problems.

Problem 2 (Decoupled Problem i)**.** Given a constant cost $C > 0$, find a scheduling policy $\pi_i = \{u_i(t)\}_{t \geq t_0}$ and a processing time $\tau_i \in \mathcal{T}_i$ that minimize the infinite-horizon time-averaged cost of agent i:

$$\min_{\substack{\tau_i \in \mathcal{T}_i \\ \pi \in \Pi}} \limsup_{T \to +\infty} \mathbb{E}_\pi \left[\frac{1}{T} \sum_{t=t_0}^{T} \Big(J_i\left(\tau_i, A_i^{\pi_i}(t)\right) + C u_i(t) \Big) \right] \tag{P2}$$

In Problem 2, the multiplier C can be interpreted as a transmission cost: whenever $u_i(t) = 1$, agent i has to pay a cost of C for using the channel. Further, transmitting an entire update costs $Cr_i(\tau_i)$, since i transmits for $r_i(\tau_i)$ time-slots. In the next section, we look at the single-agent problem (P2) in greater detail, and show how to solve it exactly. Since the problem involves a single agent, it is much easier to solve than the original combinatorial formulation. The solution also provides key insights in choosing both the scheduling policy and the processing times for the original problem (P1).

A. Solving the Decoupled Problem

We now solve Problem 2 for each agent separately. First, we characterize the structure of the optimal scheduling policy π_i^* given a fixed value of τ_i. Then, we optimize over the latter.

Theorem 1. *The solution to Problem 2, given a fixed value of τ_i, is a stationary threshold-based policy: let $\widetilde{H}_i \triangleq H_i + r_i(\tau_i)$ and suppose there exists an age H_i that satisfies*

$$J_i(\tau_i, \widetilde{H}_i - 1) \leq J_i^W(\tau_i, H_i) \leq J_i\big(\tau_i, \widetilde{H}_i\big) \tag{4}$$

where

$$J_i^W(\tau_i, H_i) \triangleq \frac{\sum_{h=\Delta_i}^{\widetilde{H}_i - 1} J_i(\tau_i, h) + C r_i(\tau_i)}{\widetilde{H}_i - \Delta_i}. \tag{5}$$

Then, an optimal scheduling policy π_i^* is to start sending an update whenever $A_i(t) \geq H_i$ and to not transmit otherwise. If no such H_i exists, the optimal policy is to never transmit. The quantity $J_i^W(\tau, H_i)$ represents the time-average cost of using a threshold policy with the AoI threshold H_i.

Proof: See Appendix A in the technical report [23]. ■

The structure of the optimal scheduling policy π_i^* according to Theorem 1 is intuitive, due to the monotonicity of the cost functions $J_i(\tau_i, \cdot)$ in the AoI. If it is optimal to transmit and pay the cost C for $r_i(\tau_i)$ time-slots at a particular AoI, it should be also be optimal to do so when the AoI is higher, since the gain from AoI reduction would be even more. Given τ_i and C , a way to compute the optimal threshold is to start from $H = \Delta_i$ and increase H until condition (4) is satisfied. Let the value that this procedure terminates at be denoted by $H_i(\tau_i)$. Then, $H_i(\tau_i)$ is an optimal threshold for agent i.

Next, we look at how to compute the optimal processing time τ_i^* to solve Problem 2. To do so, given the admissible set $\mathcal{T}_i$, we find the value of $\tau_i \in \mathcal{T}_i$ that induces the lowest time-averaged cost for agent i by enumerating over the set,

$$\tau_i^* = \arg\min_{\tau_i \in \mathcal{T}_i} \tilde{J}_i^W(\tau_i) \tag{6}$$

where $\tilde{J}_i^W(\tau_i) \triangleq J_i^W\big(\tau_i, H_i(\tau_i)\big)$. The optimal processing times τ_i^* and policies π_i^*, with thresholds $H_i(\tau_i^*)$, computed for each decoupled problem provide an optimal solution to the inner minimization of (3).

B. Optimizing Processing Times in Problem 1

Leveraging the solution of the decoupled problems found in Section III-A, we now design a procedure to optimize the processing times for the original multi-agent Problem 1.

Given a cost $C > 0$, we can use (4) and (6) to compute the optimal processing times τ_i^* and the corresponding AoI thresholds $H_i(\tau_i^*)$ for the N decoupled problems in (3). Further, observe that, for the i-th decoupled problem, the optimal scheduling policy for agent i chooses to send a new update every time the AoI exceeds $H_i(\tau_i^*)$ and the AoI drops to Δ_i after each update delivery. Thus, the fraction of time that agent i occupies the channel (on average) is given by

$$f_i(\tau_i^*) = \frac{r_i(\tau_i^*)}{H_i(\tau_i^*) + r_i(\tau_i^*) - \Delta_i}. \tag{7}$$

The total channel utilization given the Lagrange multiplier C is $f = \sum_{i \in \mathcal{V}} f_i(\tau_i^*)$. From (2), f must lie in the interval $[0, 1]$ to represent a feasible allocation of computation and communication resources. If not, then more than one agent is transmitting in every time-slot *on average*, which is not possible given the (relaxed) interference constraint.

This suggests a natural way to optimize over both the Lagrange cost C and the processing times τ_i, which is presented in Algorithm 1. In particular, we optimize the processing times τ_i by using (4) and (6) (line 4 in Algorithm 1), and update C via a dual-ascent scheme (lines 6–7) using the average channel utilization f_{curr}.

Algorithm 1 Optimizing Processing Times

Input: Costs $J_i^W(\cdot)$, set of admissible processing times $\mathcal{T}_i$ for each agent $i \in \mathcal{V}$, stepsize $\alpha > 0$.
Output: Locally optimal processing times $\{\tau_i^*\}_{i \in \mathcal{V}}$.

1: $C \leftarrow C_0$;
2: **loop**
3: **for** sensor $i \in \mathcal{V}$ **do** // optimization (6)
4: $\tau_i^* \leftarrow \arg\min_{\tau_i \in \mathcal{T}_i} \tilde{J}_i^W(\tau_i)$;
5: **end for**
6: $f_{curr} \leftarrow \sum_{i \in \mathcal{V}} f_i(\tau_i^*)$;
7: $C \leftarrow C + \alpha(f_{curr} - 1)$;
8: **end loop**
9: **return** $\{\tau_i^*\}_{i \in \mathcal{V}}$.

Intuitively, the algorithm keeps increasing the virtual communication cost (quantified by the Lagrange multiplier C) until the processing times computed at step 4 become compatible with the scheduling constraint. The decoupling reduces the

complexity of finding the optimal processing times from combinatorial $O\left(\prod_{i\in\mathcal{V}}|\mathcal{T}_i|\right)$ to linear search $O\left(\sum_{i\in\mathcal{V}}|\mathcal{T}_i|\right)$.

IV. Whittle-index Scheduling

In the previous section, we established a threshold structure for the optimal scheduling policy of the relaxed problem (2), where each agent transmits when its AoI exceeds $H_i(\tau_i^*)$. Next, we exploit this threshold structure to design an efficient scheduling policy for Problem 1. Given the processing times τ_i^* computed via Algorithm 1, we need to solve:

$$\begin{aligned}\min_{\pi\in\Pi} \quad & \sum_{i\in\mathcal{V}}\limsup_{T\to+\infty}\mathbb{E}_\pi\left[\frac{1}{T}\sum_{t=t_0}^{T}J_i\left(\tau_i^*,A_i^\pi(t)\right)\right]\\ \text{s.t.} \quad & \sum_{i\in\mathcal{V}}u_i(t)\leq 1,\forall t.\end{aligned} \tag{8}$$

Minimizing the time-average of increasing functions of AoI was considered in [5]. There, the authors introduced a low-complexity near-optimal scheduling policy using the Whittle index approach. Unlike the setting in [5], our agents generate updates at different rates (every τ_i time-slots for agent i) and induce different communication delays ($r_i(\tau_i)$ time-slots). We now generalize the Whittle index approach for our setting.

The Whittle index approach consists of four steps: 1) converting the problem into an equivalent restless multi-armed bandit (RMAB) formulation, 2) decoupling the problem via a Lagrange relaxation, 3) establishing a structural property called *indexability* for the decoupled problems, and 4) using this structure to formulate a Whittle index policy for the original scheduling problem. We go through these steps below.

Step 1. We first need to establish (8) can be equivalently formulated as a restless multi-armed bandit problem. We do so in Appendix B in the technical report [23].

Step 2. As we observed in Section III, the original scheduling problem can be split into N decoupled problems of the form (P2) via a Lagrange relaxation. Further, through Theorem 1, we know that the optimal scheduling policy for each decoupled problem has a threshold structure, *i.e.,* agent i should transmit only if its associated AoI $A_i(t)$ exceeds the threshold $H_i(\tau_i^*)$.

Step 3. Whittle showed in [22] that when there is added structure in the form of a property called *indexability* for the decoupled problems, then the RMAB admits a low-complexity solution called the Whittle index, that is known to be near optimal [24]. The *indexability* property for the i-th decoupled problem requires that, as the transmission cost C increases from 0 to ∞, the set of AoI values for which it is optimal for agent i to transmit must decrease monotonically from the entire set (all ages $A_i(t)\geq\Delta_i$) to the empty set (never transmit). In other words, the optimal threshold $H_i(\tau_i^*)$ should increase as the transmission cost C increases. Next, we use Theorem 1 and the monotonicity of the cost functions $J_i(\tau_i^*,\cdot)$ to establish that the decoupled problems are indeed indexable.

Lemma 1. *The indexability property holds for the decoupled problems* (2)*, given an allocation of processing times* τ_i*.*

Proof: See Appendix C in the technical report [23]. ∎

Step 4. Having established indexability for the decoupled Problem 2, we can derive a functional form for the Whittle index which solves the scheduling for the original Problem 1.

Definition 1. For the i-th decoupled problem, the Whittle index $W_i(H)$ is defined as the minimum cost C that makes both scheduling decisions (transmit, not transmit) equally preferable at AoI H. Let $\widetilde{H}\triangleq H+r_i(\tau_i)$. The expression for $W_i(H)$, given a processing time τ_i, is:

$$W_i(H)\triangleq\frac{\left(\widetilde{H}-\Delta_i\right)J\left(\tau_i,\widetilde{H}\right)-\sum_{k=\Delta_i}^{\widetilde{H}-1}J(\tau_i,k)}{r_i(\tau_i)}. \tag{9}$$

We derive the expression above in Appendix C. Using (9), we can now design the Whittle index policy to solve (8). Whenever the channel is unoccupied, the agent with the most critical update should be asked for an update. This leads to the scheduling policy presented in Algorithm 2. The Whittle index policy chooses the agent with the highest index (line 4), since it represents the minimum cost each agent would be willing to pay to transmit at the current time-slot. When the channel is occupied, no other transmission is allowed (line 6). The variable z keeps track of ongoing communication and drops to zero when a new transmission can be scheduled.

Algorithm 2 Whittle Index Scheduling

Input: Processing time τ_i, communication delay $r_i(\cdot)$, and cost $J_i(\cdot,\cdot)$ for each agent $i\in\mathcal{V}$, time horizon T.

1: $t=t_0$, $z=0$;
2: **while** $t\leq T$ **do**
3: **if** $z=0$ **then** // schedule transmission at time t
4: $\pi\leftarrow\arg\max_{i\in\mathcal{V}} W_i(A_i(t))$; // trigger agent π
5: $z\leftarrow r_\pi(\tau_\pi)-1$;
6: **else** // continue ongoing transmission
7: $z\leftarrow z-1$;
8: **end if**
9: **end while**

The Whittle index is known to be asymptotically optimal as $N\to\infty$, if a fluid limit condition is satisfied [24], [25]. These results, along with our simulations, suggest that the Whittle index is a very good low-complexity heuristic for scheduling in real-time monitoring and control applications.

V. Applications

We demonstrate our co-design algorithms in two applications: multi-agent occupancy grid mapping in time-varying environments (Section V-A), and ride sharing in autonomous vehicle networks (Section V-B). The results show that we can achieve performance improvements of $18-35\%$ for grid mapping and $75-82\%$ for ride-sharing compared to baseline approaches. We also provide a video briefly summarizing and visualizing our simulation results [26].

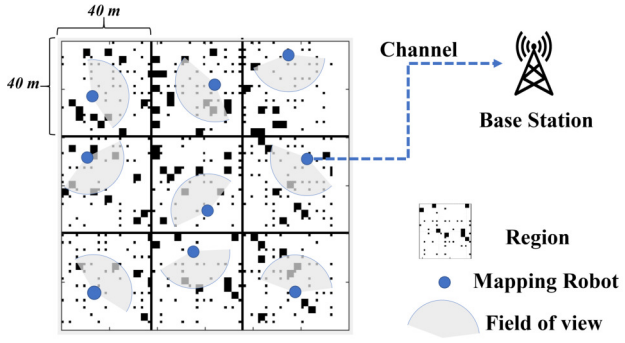

Fig. 3. **Multi-agent mapping over 9 regions:** each agent monitors and builds a local grid map of a region, and sends map updates to a base station. The occupancy in the regions is time-varying. A scheduling policy specifies how to share the communication channel among the agents. Processing times specify how much time each agent spends in generating new map updates.

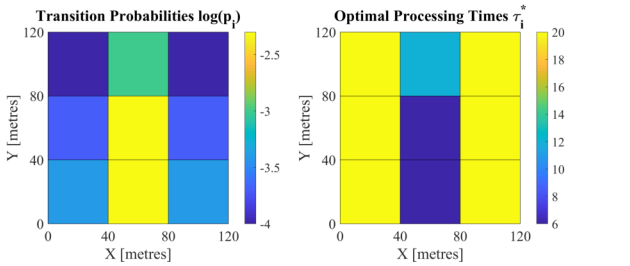

Fig. 4. Transition probabilities and optimal processing time allocations plotted for each region. The probabilities are plotted on a logarithmic scale while the processing times are plotted in number of time-slots.

A. *Multi-agent Mapping of Time-Varying Environments*

Setup. We co-design computation and communication for a multi-agent mapping problem. We assume there are N separate regions each of which is being mapped by an agent. The agents send updates —in the form of occupancy grid maps of their surroundings— to a base station over a single communication channel, where the local maps are aggregated into a global map for centralized monitoring (Fig. 3).

In our tests, each region is $40\text{m} \times 40\text{m}$ in size and is represented by an occupancy grid map with $1\text{m} \times 1\text{m}$ cells. The state of each cell can be either *occupied* (1) or *unoccupied* (0). We consider a dynamic environment where the state of each cell within region i evolves according to a Markov chain, with cells remaining in their original state with probability $1 - p_i$ and switching from occupied to unoccupied and vice-versa with probability p_i. This is a common model for grid mapping in dynamic environments in the robotics community [27], [28].

Each agent is equipped with a range-bearing sensor (*e.g.,* lidar), with a fixed maximum scanning distance (25m) and angular range $[-\pi/2, \pi/2]$. The agents move around the regions randomly, taking scans of the area round them. Scanning an entire region takes an agent multiple time-slots. We use the Navigation toolbox in MATLAB to create sensors such that the resolution of the readings θ_{min} improves with the processing time. We set $\theta_{\text{min}} = 0.5/\tau$. We also set the noise variance in angle and distance measurements to be inversely proportional to τ. These settings capture the delay-accuracy trade-off. We further set the update communication times to increase linearly with the amount of processing, *i.e.,* $r(\tau) = 5 + \lceil \tau/2 \rceil$.

The base station maintains an estimate of the current map for each region based on the most recent update it received and the Markov transition probabilities $\{p_i\}_{i \in \mathcal{V}}$ associated with each region. As is common in mapping literature [29], [30], we measure uncertainty at the base station in terms of entropy of the current estimated occupancy grid map for each region and set the cost functions $J_i(\cdot, \cdot)$ to be the entropy of region i. In Appendix D in the technical report [23], we show that the entropy cost increases monotonically with the AoI of a region and satisfies the assumptions of our framework. It drops to a lower value if more time was spent in processing, since the base station is more certain about the quality of the received update. Our goal is to minimize the time-average of the entropies summed across each region through the joint optimization of processing times and the scheduling policy.

Results. Fig. 4 shows an example of transition probabilities p_i (for each of the 9 regions) and the corresponding optimal processing times τ_i^* found using Algorithm 1. We observe that for regions that change quickly (*i.e.,* have large value of p_i), the corresponding processing time allocated is smaller. This is because there is not much benefit to spending large amounts of time generating high quality updates if they become outdated very quickly. Conversely, for slowly changing regions (with low values of p_i), Algorithm 1 assigns much longer processing times. In this case, high quality useful updates can be created by taking longer time since the regions don't change quickly.

Further, we compare the performance of various scheduling algorithms in Fig. 5. We consider the setting where the processing times τ_i are fixed to be the same parameter τ for every region (uniform processing allocation). We then plot the performance of three scheduling algorithms –a uniform stationary randomized policy, a round-robin policy, and the proposed Whittle index-based policy– for different values of τ. We also plot the performance of the Whittle index policy and the stationary randomized policy under the optimized processing times, computed using Algorithm 1, shown via dotted lines in Fig. 5. We observe that Algorithm 1 can find processing times that perform well in practice. We also observe that the Whittle index policy outperforms the two "traditional" classes of scheduling policies for every value of the parameter τ.

Overall, choosing the processing times using Algorithm 1 and using the Whittle schedule from Algorithm 2 together leads to a performance improvement of $28 - 35\%$ over the baseline versions of randomized policies. Similarly, our proposed approach leads to a performance improvement of $17 - 28\%$ over the baseline versions of round-robin policies.

B. *Smart Ride Sharing Control in Vehicle Networks*

Setup. We consider the scenario in which a ride-sharing taxi fleet serves a city coordinated by a central scheduler, which receives riding requests and assigns them to the drivers. Assigned requests are enqueued into a FIFO-like queue for each driver. In particular, a rider is matched to the driver whose predicted route has the shortest distance to the pick-up location.

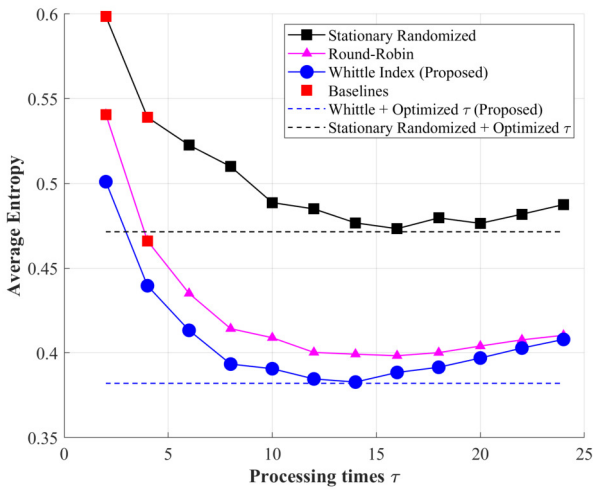

Fig. 5. Performance of different scheduling policies vs. processing times τ. Solid lines represent performance of different classes of scheduling policies as the processing time τ varies. The dotted lines represent the scheduling performance with processing times computed using Algorithm 1.

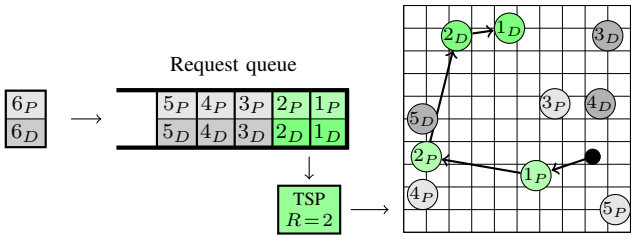

Fig. 6. Drivers calculate their route by processing the oldest R requests (green queue portion). The TSP solver starts from the current driver location and involves pick ups (P) and drop offs (D) of the processed requests.

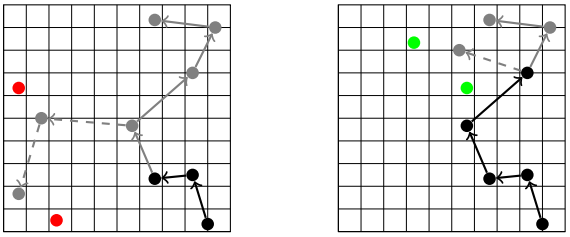

Fig. 7. Left: long processing causes large gaps between the route followed by the driver (solid gray) and the one stored at the scheduler (dashed gray), yielding bad matches (red dots). Right: with short computations, the matched requests are closer to the actual route (green dots). The traveled path is black.

In our setup, routes are calculated locally by drivers and transmitted on demand to the scheduler, which uses this information to match future requests. Such distributed processing for route optimization is different from current architectures, which are usually centralized. However, it allows for much greater scalability and is envisioned as a key component in increasing efficiency and scale of future ride-sharing systems [31]–[33].

Given communication constraints, only one driver can transmit at a time. Drivers update their route periodically to embed real-time road conditions and remove served requests from the queue. Routes are calculated via the Travelling Salesman Problem (TSP) involving the first R pick-up and drop-off locations in the request queue (Fig. 6). Processing many requests ensures more efficient paths for enqueued riders, thus shortening their *travel time* from pick up to drop off. Conversely, the computational complexity of the TSP (*i.e.,* its delay) increases with the amount of processed requests R. As a consequence, the information collected by the scheduler gets outdated more quickly (Fig. 7), inducing larger gaps with the actual route followed by the driver. This leads to worse driver-request matching and increases the *waiting time* experienced by riders before they are actually picked up. Since the overall Quality of Service (QoS) is measured through the *service time*, given by the sum of travel and waiting times of riders, the drivers face a trade-off: processing many requests shortens the travels, while processing few reduces the waiting time.

In our tests, we model the city as a 200-node graph where each driver travels one edge per time slot. Requests are randomly generated according to a Poisson process of unit intensity and assigned immediately to the matching driver by the scheduler. Each request contributes one time slot to the processing time of the TSP (*e.g.,* $\tau = 2$ corresponds to processing two requests) and we set $r(\tau) = \tau$ (longer processing yields a longer route to transmit). To exploit the advantage of the Whittle index, we simulate an heterogeneous fleet with five "myopic" drivers, which can only process the oldest request ($\tau_m = 1$), and five "smart" drivers whose processing can be designed: in particular, we assign the same processing time τ_s to all such "smart" drivers. The cost of each driver, given by its *average service time* (AST), is modeled as

$$J_i\left(\tau_i, A_i(t)\right) = P_i(\tau_i) + A_i(t) \tag{10}$$

where the estimated contribution of the local processing (TSPs)

$$P_i(\tau_i) \doteq \left(2 + 2\mathrm{e}^{-0.2\tau_i}\right) \hat{q}_i(t) \tag{11}$$

was fitted from simulations with an initial queue and no assignments. Because the number of enqueued requests affects the AST but cannot be computed offline, we modeled $P_i(\tau_i)$ as linear with the queue length. The scheduler approximates the queue length at time t with the latest received value $\hat{q}_i(t)$. The dependence on $A_i(t)$ is hard to assess and we let it linear.[1]

Results. We compute statistics over 1000 Monte Carlo runs. Fig. 8 shows the AST with 10000 requests assigned during the simulation for $\tau_s \in \{1, ..., 7\}$. The circles refer to the performance obtained with the Whittle index policy, while the squares to Stationary Randomized which is used as a benchmark. Combining Whittle index-based scheduling with processing optimization (green circle) yields a striking improvement of the QoS (AST $= 41$) compared to the Stationary Randomized with standards policies (red squares) such as FIFO request service ($\tau_s = 1$, AST $= 225$), or back-to-back trips [34] ($\tau_s = 2$, AST $= 165$). In particular, the minimum at $\tau_s^* = 5$ indicates that it is optimal to process the five oldest requests in the queue. Also, the Whittle index outperforms Stationary Randomized for all values of the

[1]Other cost functions decreasing with τ_i and increasing with $A_i(t)$ also yield good performance, suggesting that our approach is indeed robust.