Distributed and Distribution-Robust Meta Reinforcement Learning (D^2-RMRL) for Data pre-storage and Routing in Cube Satellite Networks

Ye Hu, Xiaodong Wang, *Fellow, IEEE*, and Walid Saad, *Fellow, IEEE*

***Abstract*—In this paper, the problem of data pre-storage and routing in dynamic, resource-constrained cube satellite networks is studied. In such a network, each cube satellite delivers requested data to user clusters under its coverage. A group of ground gateways will route and pre-store certain data to the satellites, such that the ground users can be directly served with the pre-stored data. This pre-storage and routing design problem is formulated as a decentralized Markov decision process (Dec-MDP) in which we seek to find the optimal strategy that maximizes the pre-store hit rate, i.e., the fraction of users being directly served with the pre-stored data. To obtain the optimal strategy, a distributed distribution-robust meta reinforcement learning (D^2-RMRL) algorithm is proposed that consists of three key ingredients: value-decomposition for achieving the global optimum in distributed setting with minimum communication overhead, meta learning to obtain the optimal initial to reduce the training time under dynamic conditions, and pre-training to further speed up the meta training procedure. Simulation results show that, using the proposed value decomposition and meta training techniques, the satellite networks can achieve a 31.8% improvement of the pre-store hits and a 40.7% improvement of the convergence speed, compared to a baseline reinforcement learning algorithm. Moreover, the use of the proposed pre-training mechanism helps to shorten the meta-learning procedure by up to 43.7%.**

***Index Terms*—Cube satellite network, data pre-storage, routing, multi-agent reinforcement learning, value decomposition, actor-critic, meta learning.**

I. Introduction

Square-shaped miniature cube satellites operating on low earth orbit (LEO) can provide an endurable, reliable, and accessible data service to users in wireless disadvantaged areas. Compared to traditional large satellites on mid earth and geosynchronous orbit, cube satellites are more affordable, flexible, and can provide wireless service with speeds of up to hundreds of megabits per second [1] and [2]. However, deploying cube satellites for low latency information access is still an important open problem, because satellites have to serve unforeseeable data requests with their limited available on-board resources. In particular, it is challenging for cube satellites to serve unpredictable, diverse data needs from users around the globe using only limited contact chances in the network.

This research was supported by the U.S. National Science Foundation under Grant CNS-1909372.

Y. Hu is with the Department of Electrical Engineering, Columbia University, New York, NY, USA, 24061, and the Wireless@VT, Bradley Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, VA, USA, 24061, Emails: yh3453@columbia.edu.

X. Wang is with the Department of Electrical Engineering, Columbia University, New York, NY, USA, 24061, Email: xw2008@columbia.edu.

W. Saad is with the Wireless@VT, Bradley Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, VA, USA, 24061, Emails: walids@vt.edu.

A. Related Works and their Limitations

Prior works [3]–[5] studied a number of problems related to routing design in resource constrained cube satellite networks. The work in [3] studied the problem of contact plan design in LEO satellite networks while jointly considering the satellites' on-board energy capacity and their stochastic solar energy infeed. The authors in [4] designed a contact plan for a resource constrained LEO satellite network to deliver the satellites' data to a fixed ground base station. In [5], the problem of scalable battery aware contact plan design in mega LEO satellite constellations was treated using mixed integer linear programing. Despite their promising results, these existing on-demand routing solutions [3]–[5] started routing data only after that data is requested, which required extra processing time within the satellite communication system. To reduce such processing time, some works [6]–[8] applied in-network caching in satellite networks. The authors in [6] proposed a cache-enabled satellite-UAV-vehicle system for energy efficient data delivery services, and formulated the cache placement problem as an optimization problem. In [7], the problem of cache placement within information-centric satellite networks was investigated based on a profile of users' interests in different topics. The work in [8] proposed a stochastic model to predict content popularity to help the satellite system feed caches in advance. However, these works only considered known, fixed service requests. Indeed, the optimization-based solutions in [3]–[8] may not be suitable for the design of contact plan or cache placement in real-world, highly dynamic satellite networks with dynamic, unpredictable user requests.

More recently, there has been significant interest in realizing dynamic resilient satellite networking by employing machine learning tools [9]–[15]. In particular, the work in [9] employed a machine learning method to predict the future service needs in satellite communication networks. The authors in [10] developed a centralized machine learning algorithm that enables real-time estimation of the environment, specifically, the ever-changing rain intensity on broadband satellite communication links. In [11], a deep reinforcement learning (DRL) solution was developed for intelligent satellite communications within the national aeronautics and space administration (NASA)'s space communication and navigation testbed. Yet when managing operations on different satellites, a centralized solution such as the one in [11] can cause significant communication overhead, especially for high latency satellite systems. Thus, distributed solutions are more desirable [16]. In this regard, the authors in [12] developed a distributed hierarchical classification solution using deep learning to intelligently classify the dynamic Internet traffic flows on satellites with low overhead. The work in [13] employed a distributed extreme learning

machine solution to route data among LEO satellites based on the forecasted dynamic traffic density. In [14], the authors developed a multi agent reinforcement learning (MARL) solution that enables multiple satellites to cooperatively manage their spectrum use.

Even though the machine learning based solutions developed in [9]–[14] are capable of accomplishing certain networking tasks with unknown service needs or unknown communication environments, the developed solutions were mostly designed in a way to overfit to the target tasks. In particular, in the learning solutions of [9]–[14], the models were trained to serve specific data needs within specific communication environment. When serving new and unseen data needs, the models must be retrained, which incurs excessive computational costs. As the data needs change constantly in practical applications, the satellite networks must spent a great amount of time and energy on training the machine learning based networking solutions.

To reduce such cost for dynamic resilient operations, the notion of *meta learning* was introduced to generalize the machine learning solutions for a family of tasks that specified by a certain distribution [17]. In particular, a meta-learning model was trained over sample tasks from this distribution that serves as the initial during the regular model training for a specific unseen task, such that the regular training can be accomplished within a small number of epochs [15], [17], [18]. In this paper, we will employ such meta learning technique to reduce the training cost. However, the satellite communication system is deployed to serve users around the globe, whose needs are different, dynamic and may follow diverse distributions, which motivates generalized meta-learning solutions that can efficiently obtain meta initials for a large number of tasks distributions. We will develop such a cost effective solution for the satellite networks.

B. Contributions

The main contribution of this paper is a novel meta reinforcement learning framework for dynamic resilient prestorage and routing design in cube satellite networks. While prior works such as [6]–[8] used in-network caching to provide on-demand data service with satellite networks, they have not considered the resource constraints and environmental dynamics within the network. In contrast, here, we propose a dynamic resilient solution tailored to the cube satellite system whose goal is to optimize the service hit rate under unpredictable user requests when accounting for the resource budget of each satellite. Our key contributions include:

- We develop a novel framework to pre-store and route data in cube satellite networks. In particular, we consider a cube satellite communication system in which the satellites must serve ground users with dynamic and unpredictable needs. Within this system, the ground gateways will selectively store data to satellites, by either directly uploading data to the target satellites, or by routing data through neighboring satellites. Subsequently, the satellites can provide data service to the target users once the data is requested. To achieve a high pre-store hit rate, a policy needs to be designed for determining what data should be stored on each satellite and how the data is routed from the gateway to the destination satellite using the limited contact chances in the system. We formulate this pre-storage and routing design problem as a decentralized Markov decision process (Dec-MDP), and seek to find the optimal pre-storage and routing strategy which maximizes the fraction of user requests being directly served with the pre-stored data.
- We then propose a distributed distribution-robust meta reinforcement learning (D^2-RMRL) algorithm that is shown to reach a high pre-store hit rate of dynamic service needs, with low communication overhead and computation cost. In particular, to reduces the communication overhead in distributed learning, we use the value decomposition technique to reinforce the team benefit on each data flow without exchanging their action choices and environmental observation. To reduce the learning cost in the system, we use the meta training mechanism to initialize the learning procedure based on the prior information on possible data needs at different service occasions. Moreover, we use the pre-training technique to implement a shortened meta training procedure that obtains the meta initial models for a large number of service distributions.

Simulation results show that the proposed value decomposition technique can lead to a 31.8% improvement of the pre-store rate achieved by the distributed reinforcement learning algorithm. The meta learning technique can find learning initials that results in a convergence speedup by 40.7%. Furthermore, the meta learning procedure is shortened by up to 43.7% with the proposed meta pre-training scheme under multiple service distributions.

The rest of this paper is organized as follows. The system model and problem formulation are described in Section II. In Section III, the proposed algorithm, including the value-decomposition-based actor-critic reinforce learning algorithm, the meta training algorithm, and the pre-training algorithm are presented. In Section IV, simulation results are analyzed. Finally, conclusions are drawn in Section V.

II. System Model and Problem Formulation

Consider a cube satellite network (CSN) that consists of N_{S} LEO cube satellites, and N_{G} distributed ground gateways. At each time slot, the satellites serve N_{U} user clusters, each of which represents a group of users that falls within the pre-store hits of a satellite, as shown in Fig. 1. Meanwhile, at each time slot, a gateway or a user cluster can only connect to one satellite. There are totally N_{F} content files available in the system. At each time slot, each user cluster requests its associated satellite to deliver some content files. At this point, if the requested files are stored on the associated satellite already, users in the cluster can directly download them. Otherwise, the satellite must seek the requested files from neighboring satellites or ground gateways, such that the

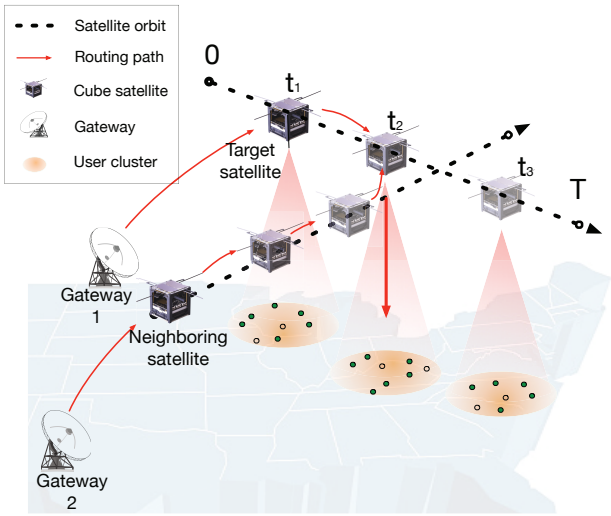

Figure 1. Topology of a cube satellite network (CSN).

content requests can be served, although with higher latency. Thus, the CSN system will pre-store the content files of interest on the satellites to serve user requests with minimum latency.

In the considered system, the gateways determine the content files that should be pre-stored on the satellites, and optimize routing path of these content files. Note that, when pre-storage content files to the satellites, the gateways do not know which files will be requested by the user clusters. This is not only because that the user requests happen in the future, but also the users' interests on content files are highly dynamic (i.e., the interests follow unknown distributions, which also change over time and user locations [19]). On the other hand, due to the limited storage capacity, the satellites cannot store all content files that are of probable interest to the users. Thus, the gateways must selectively pre-store content files on satellites. Yet, the orbiting satellites may not be able to receive all content files of interests either, as they only have limited chances to communicate with the gateways. Then, the gateways should store some content files on the neighboring satellites that can be offloaded to the target satellites as in Fig. 1. That is, when there are enough chances for the gateways and the target satellite to communicate, the gateways can directly store all files of interest on the target satellites. Otherwise, the gateways store some of the content files of interest on the target satellite, and the rest on the neighboring satellites while specifying how they are routed to the target satellite. In summary, the gateways in the CSN system determine how the content files are pre-stored on and routed to the satellites, based on the time-evolving CSN topology, network resource limitations, and user needs. Next, the transmission opportunities with storage limitations in the CSN system are modeled as an time-unrolled directed graph. Then, the problem formulation is given.

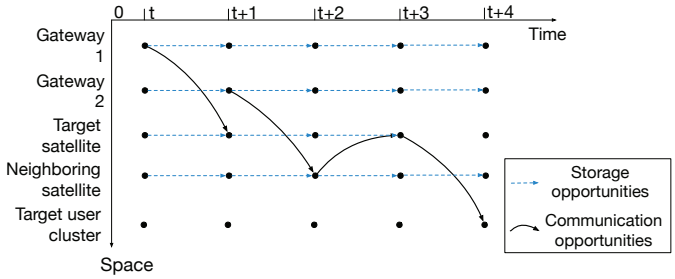

Figure 2. Time-unrolled graph for modeling transmissions and storage opportunities in the CSN.

A. Data Transmission Graph

We use a time-unrolled graph to characterize the transmission graph evolution and storage limitations within the CSN system. The t-th layer of this graph represents the transmission opportunities at time slot t. The vertices at the t-th layer, i.e., $\mathcal{U}(t) \cup \mathcal{S}(t) \cup \mathcal{G}(t)$, correspond to the replicas of user clusters, satellites and gateways at time slot t. In particular, $u \in \mathcal{U}(t)$, $s \in \mathcal{S}(t)$ and $g \in \mathcal{G}(t)$ denote, respectively, a user cluster, satellite and gateway in the CSN system at time slot t.

An edge in the graph connects a node at layer t to another node at layer $t+1$, representing either transmitting or storing a file during time slot t. In particular, the set of edges $\mathcal{E}^{\mathrm{G}}(t) = \{g \in \mathcal{G}(t) \rightarrow g \in \mathcal{G}(t+1)\}$ implies that a file that resides in a gateway g at the beginning of slot t, will remain in this gateway during slot t, and therefore, it remains in the same gateway g, at the beginning of slot $t+1$. Similarly, the set of edges $\mathcal{E}^{\mathrm{S}}(t) = \{s \in \mathcal{S}(t) \rightarrow s \in \mathcal{S}(t+1)\}$ denotes that a file stays in a satellite s during time slot t. There are three types of file transmissions during time slot t: the set of edges $\mathcal{E}^{\mathrm{GS}}(t) = \{g \in \mathcal{G}(t) \rightarrow s \in \mathcal{S}(t+1)\}$ denotes that a file is transmitted from gateway g to satellite s; the set of edges $\mathcal{E}^{\mathrm{SS}}(t) = \{s \in \mathcal{S}(t) \rightarrow s' \in \mathcal{S}(t+1)\}$ models that a file is transmitted from satellite s to a neighboring satellite s'; while the set $\mathcal{E}^{\mathrm{SU}}(t) = \{s \in \mathcal{S}(t) \rightarrow u \in \mathcal{U}(t+1)\}$ models that a file is transmitted from satellite s to user cluster u. Note that, the communication opportunities in the CSN system are constrained by the communication range, and transponders on the satellites. Thus, a device can only communicate with a target satellite when it falls within this satellite's communication range, and the target satellite is not transmitting or receiving content files at current time slot t. In the considered CSN system, the cube satellite constellation can provide globally seamless coverage, which means that, at any time slot t, for any satellite $s \in \mathcal{S}(t)$, there always exists one and only one incoming edge $g \rightarrow s$ from a gateway $g \in \mathcal{G}(t-1)$, as well as one and only one outgoing edge $s \rightarrow u$ to an user cluster $u \in \mathcal{U}(t+1)$. An inter-satellite communication edge $s \rightarrow s'$ exists only when the two satellites, $s \in \mathcal{S}(t)$ and $s' \in \mathcal{S}(t+1)$, orbit within each other's communication range. There is no edge linking two different gateways, two different user clusters, or a gateway and a user cluster. Moreover, associated with each satellite $s \in \mathcal{S}(t)$ there is a variable storage capacity Γ_s, which denotes the number of files that

are stored in s at the current time slot. Since each satellite has limited storage, we have $0 \leq \Gamma_s \leq \Gamma_{\max}$. Meanwhile, at any time slot, if a node is the source or destination of more than one edges, only one of these edges can be active. Also, notice that the data rates of the communication edges are determined by the surrounding propagation environment [20], and we assume that a communication edge guarantees the completion of a file transmission within one time slot.

B. Pre-store Hit

Given that the goal of the CSN system is to reduce service latency by pre-storage interesting content files on the satellites, the system performance is evaluated by the *pre-store hit*, which is defined as the number of user content requests being directly served with the files pre-stored on satellites. In particular, a user cluster can be served with content files on satellites only if its content files of interest are already pre-stored on its associated satellite, by the time it requests these content files. For each user cluster u, define an $N_{\mathrm{F}} \times 1$ vector $\boldsymbol{x}_u(t) = \left[x_u^1(t), \ldots, x_u^{N_{\mathrm{F}}}(t)\right]$, where

$$x_u^f(t) = \begin{cases} 1, & \text{if } u \text{ requests file } f \text{ at the beginning of time slot } t \\ 0, & \text{otherwise} \end{cases}. \tag{1}$$

We assume that if a file f is transmitted to a satellite s in time slot t, then it is stored in s, at the beginning of time slot $t+1$. For each satellite s, define an $N_{\mathrm{F}} \times 1$ vector $\boldsymbol{y}_s = \left[y_s^1, \ldots, y_s^{N_{\mathrm{F}}}\right]$, where y_s^f denotes the "age" of file f at s, i.e.,

$$y_s^f = \begin{cases} k, & \text{if } f \text{ is stored on } s \text{ at the beginning of time slot } k \\ \infty, & \text{if } f \text{ is not stored on } s \end{cases}, \tag{2}$$

with $k = 1, 2, \ldots$. Then, the total hits at time slot t is

$$h(t) = \sum_{f=1}^{N_{\mathrm{F}}} \sum_{u \in \mathcal{U}(t)} \sum_{\substack{s \in \mathcal{S}(t), \\ s \to u \in \mathcal{E}^{\mathrm{SU}}(t)}} \mathbb{1}_{\left\{x_u^f(t)=1, y_s^f \neq \infty\right\}}, \tag{3}$$

with $\mathbb{1}_{\chi} = 1$ when χ is true, otherwise, $\mathbb{1}_{\chi} = 0$. Thus, $\mathbb{1}_{\left\{x_u^f(t)=1, y_s^f \neq \infty\right\}} = 1$ represents that user cluster u can directly download content file f from satellite s, at slot t. The pre-storage and routing process in the considered CSN system aims at the highest number of pre-store hits, under the network topology evolution and resource limitations. This pre-storage and routing process is modeled in the next subsection.

C. MDP Modeling for a Single Content File

In our system model, the gateways make plans on how each content file is pre-stored and routed within a CSN system. In particular, only the content files that are deemed of interest to the ground users should be pre-stored on the target satellites, or neighboring satellites, using the limited communication and storage opportunities in the system. However, the resulting pre-store hits is partially determined by the gateways' decisions, and partially determined by the dynamic, and unforeseeable user requests in the system. The routing process of a given content file f can be formulated as a Markov decision process (MDP) $\left\langle \Omega^f, \mathcal{A}^f, R^f \right\rangle$ [21]. We next describe the state space Ω^f, the action space $\mathcal{A}^f$ and the reward function R^f.

1) State: Ω^f is the state space of content file f. The state is a pair $\sigma^f(t) = [v, x]$, where for a regular state $v \in \mathcal{G}(t) \bigcup \mathcal{S}(t)$, and $x \in \{0, 1, \emptyset\}$. Specifically, v denotes the location of file f at time t. If $v = g \in \mathcal{G}(t)$, i.e., file f is in gateway g, then $x = \emptyset$. On the other hand, if $v = s \in \mathcal{S}(t)$, i.e., file f is in satellite s at time t, then $x = x_u^f(t)$ indicates whether or not file f is requested by the user cluster u currently connected to s, i.e., $\{s \to u\} \in \mathcal{E}^{\mathrm{SU}}(t)$. Moreover, there is an initial state denoted as $[\mathsf{I}, \emptyset]$ and a terminal state denoted as $[\mathsf{T}, \emptyset]$. A file always starts from the initial state and stays there until it is picked by a gateway, and it terminates when the satellite that stores it decides to discard it.

2) Action: $\mathcal{A}^f$ is the action space of content file f. The possible actions $a^f(t)$ corresponding to the state $\sigma^f(t)$, and the resulting state $\sigma^f(t+1)$ are as follows; and we denote the corresponding probabilistic policy as $\pi^f\left(a^f(t) \,|\sigma^f(t)\right)$.

- For the initial state $\sigma^f(t) = [\mathsf{I}, \emptyset]$, there are two possible actions:
  (a) $a^f(t) = \{\mathsf{I} \to \mathsf{I}\}$, i.e., the file stays at the initial state, and hence $\sigma^f(t+1) = [\mathsf{I}, \emptyset]$.
  (b) $a^f(t) = \{\mathsf{I} \to g\} \in \mathcal{G}(t)$, i.e., the file is picked by gateway g, and hence $\sigma^f(t+1) = [g, \emptyset]$.
- For a regular state $\sigma^f(t) = [g, \emptyset]$, there are two possible actions:
  (a) $a^f(t) = \{g \to g\} \in \mathcal{E}^{\mathrm{G}}(t)$, i.e., file f stays in gateway g, leading to the next state $\sigma^f(t+1) = [g, \emptyset]$.
  (b) $a^f(t) = \{g \to s\} \in \mathcal{E}^{\mathrm{GS}}(t)$, i.e., file f is transmitted from gateway g to satellite s, leading to the next state $\sigma^f(t+1) = \left[s, x_u^f(t+1)\right]$, with $\{s \to u\} \in \mathcal{G}^{\mathrm{SU}}(t+1)$. As part of this action, we record the time file f is stored in satellite s with $y_s^f \leftarrow t+1$, and update the occupancy of s as $\Gamma_s \leftarrow \Gamma_s + 1$. Now, if $\Gamma_s = \Gamma_{\max}$, i.e., the storage capacity is reached, then the oldest file on s is discarded, i.e., $y_s^{\hat{f}} \leftarrow \infty$, where $\hat{f} = \arg\min_{f \in \mathcal{F}} y_s^f$, and $\Gamma_s \leftarrow \Gamma_s - 1$. Moreover, the file $\hat{f}$ reaches the terminal state, i.e., $\sigma^{\hat{f}}(t+1) = [\mathsf{T}, \emptyset]$. Hence, the action $a^f(t)$ of file f can affect the state $\sigma^{\hat{f}}(t+1)$ of another file $\hat{f}$.
- For a regular state $\sigma^f(t) = [s, x]$, there are three possible actions:
  (a) If $x = 1$, i.e., file f is requested by user cluster u that is connected to s, then $a^f(t) = \{s \to u\} \in \mathcal{E}^{\mathrm{SU}}(t)$, i.e., file f is transmitted from satellite s to user cluster u, leading to the next state $\sigma^f(t+1) = \left[s, x_u^f(t+1)\right]$, with $\{s \to u\} \in \mathcal{E}^{\mathrm{SU}}(t+1)$.
  (b) If $x = 0$, then $a^f(t)$ can be $\{s \to s\} \in \mathcal{E}^{\mathrm{S}}(t)$, i.e., file f stays in satellite s, leading to the next state $\sigma^f(t+1) = \left[s, x_u^f(t+1)\right]$, with $\{s \to u\} \in \mathcal{E}^{\mathrm{SU}}(t+1)$.
  (c) Or if $x = 0$, $a^f(t)$ can also be $\{s \to s'\} \in \mathcal{E}^{\mathrm{SS}}(t)$, i.e., file f is transmitted from satellite s to its neighboring satellite s', leading to the next state $\sigma^f(t+1) = \left[s', x_u^f(t+1)\right]$, with $\{s' \to u\} \in \mathcal{E}^{\mathrm{SU}}(t+1)$. As part

of this action, we record the time file f is stored in satellite s' with $y_{s'}^{f} \leftarrow t+1$, and set $y_{s}^{f} \leftarrow \infty$, since file f is moved out of satellite s. We update the occupancy of s as $\Gamma_s \leftarrow \Gamma_s - 1$, and the occupancy of s' as $\Gamma_{s'} \leftarrow \Gamma_{s'} + 1$. Now, if $\Gamma_{s'} = \Gamma_{\max}$, i.e., the storage capacity is reached, then the oldest file on s' is discarded, i.e., $y_{s'}^{\hat{f}} \leftarrow \infty$, where $\hat{f} = \arg\min_{f \in \mathcal{F}} y_{s'}^{f}$, and $\Gamma_{s'} \leftarrow \Gamma_{s'} - 1$. Moreover, the file $\hat{f}$ reaches the terminal state, i.e., $\sigma^{\hat{f}}(t+1) = [\mathsf{T}, \emptyset]$.

- For the terminal state $\sigma^{f}(t) = [\mathsf{T}, \emptyset]$, no action is needed and the state remains terminal.

3) Reward: The reward function $R^{f}(t) = R^{f}\left(a^{f}(t), \sigma^{f}(t)\right)$ evaluates action choices $a^{f}(t) \in \mathcal{A}^{f}$ at different states $\sigma^{f}(t) \in \Omega^{f}$. Since the system performance metric is the total number of hits in (3), we get a unit reward at time t corresponding to a hit if and only if the action is a file transmission from a satellite to its user cluster, i.e., $R^{f}(t) = \mathbb{1}_{\{\sigma^{f}(t)=[s,1]\}}$.

D. Problem Formulation

The routing process in the CSN system considers multiple content files. The routing decisions on one file f can directly affect those on some other content files, since the files compete for the communication or storage resources of the satellites. Thus, the routing of multiple content files should be cooperatively arranged by the gateways. To model these interdependent decision making processes, we formulate a Dec-MDP defined by $\langle \mathcal{F}, \Omega, \mathcal{A}, R \rangle$ [21], where

- $\mathcal{F} = \{1, ..., N_{\mathrm{F}}\}$ is the set of content files to be routed in the CSN system.
- $\Omega = \Omega^{1} \times \ldots \times \Omega^{N_{\mathrm{F}}}$ is the state space. The state of the multiple content routing process at time slot t is captured as $\boldsymbol{\sigma}(t) = \left(\sigma^{1}(t), \ldots, \sigma^{N_{\mathrm{F}}}(t)\right)$.
- $\mathcal{A} \subset \mathcal{A}^{1} \times \ldots \times \mathcal{A}^{N_{\mathrm{F}}}$ is the action space. The action of the multiple content routing process at time slot t is captured as $\boldsymbol{a}(t) = \left(a^{1}(t), \ldots, a^{N_{\mathrm{F}}}(t)\right)$. Note that, due to the possible conflicts in the system, e.g., multiple routing processes compete for one communication or storage chance, the action space $\mathcal{A}$ is a strict subset of $\mathcal{A}^{1} \times \ldots \times \mathcal{A}^{N_{\mathrm{F}}}$, and captures only the viable action choices under resource limitations. In particular, at time slot t, if every file f follows its own local policy $\pi^{f}\left(a^{f}(t) | \sigma^{f}(t)\right)$, then for any give file f, its action $a^{f}(t) \in \mathcal{A}^{f}$ might be in conflict with the action $a^{f'}(t) \in \mathcal{A}^{f'}$ of another file f' based on its local policy $\pi^{f'}\left(a^{f}(t) | \sigma^{f}(t)\right)$, as follows
  - For the initial state $\sigma^{f}(t) = [\mathsf{I}, \emptyset]$, the two possible actions $\mathsf{I} \rightarrow \mathsf{I}$ and $\mathsf{I} \rightarrow g$ are not in conflict with the action of any other file.
  - For a regular state $\sigma^{f}(t) = [g, \emptyset]$,
    (a) Action $a^{f}(t) = \{g \rightarrow g\} \in \mathcal{E}^{\mathrm{G}}(t)$ is not in conflict with others.
    (b) Action $a^{f}(t) = \{g \rightarrow s\} \in \mathcal{E}^{\mathrm{GS}}(t)$ is in conflict with actions $a^{f'}(t)$ of another file f' that take gateway g or satellite s as the transmitter or receiver, i.e.,

$$a^{f'}(t) = \begin{cases} \{g \rightarrow s'\} \in \mathcal{E}^{\mathrm{GS}}(t), & \text{if } \sigma^{f'}(t) = [g, \emptyset] \\ \{g' \rightarrow s\} \in \mathcal{E}^{\mathrm{GS}}(t), & \text{if } \sigma^{f'}(t) = [g', \emptyset] \\ \{s' \rightarrow s\} \in \mathcal{E}^{\mathrm{SS}}(t), & \text{if } \sigma^{f'}(t) = [s', 0] \\ \{s \rightarrow s'\} \in \mathcal{E}^{\mathrm{SS}}(t), & \text{if } \sigma^{f'}(t) = [s, 0] \\ \{s \rightarrow u\} \in \mathcal{E}^{\mathrm{SU}}(t), & \text{if } \sigma^{f'}(t) = [s, 1] \end{cases}, \tag{4}$$

    for all $f' \in \mathcal{F} \backslash \{f\}$.
  - For a regular state $\sigma^{f}(t) = [s, 1]$, the only possible $a^{f}(t) = \{s \rightarrow u\} \in \mathcal{E}^{\mathrm{SU}}(t)$ is in conflict with the following actions:

$$a^{f'}(t) = \begin{cases} \{g \rightarrow s\} \in \mathcal{E}^{\mathrm{GS}}(t), & \text{if } \sigma^{f'}(t) = [g, \emptyset] \\ \{s' \rightarrow s\} \in \mathcal{E}^{\mathrm{SS}}(t), & \text{if } \sigma^{f'}(t) = [s', \emptyset] \\ \{s \rightarrow s'\} \in \mathcal{E}^{\mathrm{SS}}(t), & \text{if } \sigma^{f'}(t) = [s, 0] \\ \{s \rightarrow u\} \in \mathcal{E}^{\mathrm{SU}}(t), & \text{if } \sigma^{f'}(t) = [s, 1] \end{cases}, \tag{5}$$

    for all $f' \in \mathcal{F} \backslash \{f\}$.
  - For a regular state $\sigma^{f}(t) = [s, 0]$,
    (a) Action $a^{f}(t) = \{s \rightarrow s\} \in \mathcal{E}^{\mathrm{S}}(t)$ is not in conflict with others.
    (b) Action $a^{f}(t) = \{s \rightarrow s'\} \in \mathcal{E}^{\mathrm{SS}}(t)$, is in conflict with the following actions:

$$a^{f'}(t) = \begin{cases} \{g \rightarrow s\} \in \mathcal{E}^{\mathrm{GS}}(t), & \text{if } \sigma^{f'}(t) = [g, \emptyset] \\ \{g \rightarrow s'\} \in \mathcal{E}^{\mathrm{GS}}(t), & \text{if } \sigma^{f'}(t) = [g, \emptyset] \\ \{s'' \rightarrow s\} \in \mathcal{E}^{\mathrm{SS}}(t), & \text{if } \sigma^{f'}(t) = [s'', 0] \\ \{s'' \rightarrow s'\} \in \mathcal{E}^{\mathrm{SS}}(t), & \text{if } \sigma^{f'}(t) = [s'', 0] \\ \{s \rightarrow s''\} \in \mathcal{E}^{\mathrm{SS}}(t), & \text{if } \sigma^{f'}(t) = [s, 0] \\ \{s' \rightarrow s''\} \in \mathcal{E}^{\mathrm{SS}}(t), & \text{if } \sigma^{f'}(t) = [s', 0] \\ \{s \rightarrow u\} \in \mathcal{E}^{\mathrm{SU}}(t), & \text{if } \sigma^{f'}(t) = [s, 1] \\ \{s' \rightarrow u\} \in \mathcal{E}^{\mathrm{SU}}(t), & \text{if } \sigma^{f'}(t) = [s', 1] \end{cases}, \tag{6}$$

    for all $f' \in \mathcal{F} / \{f\}$.
- $R(\boldsymbol{a}(t), \boldsymbol{\sigma}(t)) = \sum_{f \in \mathcal{F}} R^{f}(t) = \sum_{f \in \mathcal{F}} \mathbb{1}_{\{\sigma^{f}(t)=[s \in \mathcal{S}(t), 1]\}}$ is the reward function that evaluates the action choice $\boldsymbol{a}(t)$ at state $\boldsymbol{\sigma}(t)$, which simply counts the number of hits at time slot t[1].

Here, our goal is to find a global policy $\pi\left(\boldsymbol{a}(t) | \boldsymbol{\sigma}(t)\right)$ that provides a probabilistic mapping from each joint state $\boldsymbol{\sigma}(t)$ to the corresponding joint action $\boldsymbol{a}(t)$ that is conflict-free. However, since the number of content files N_{F} is typically large, the underlying MDP is very high-dimensional and hard to solve or implement in practice. Therefore, in this work, we target at distributed probabilistic policies together with a conflict resolution scheme. In particular, associated with each file f there is a local policy $\pi^{f}\left(a^{f}(t) | \sigma^{f}(t)\right)$ that is a probabilistic mapping from a local state $\sigma^{f}(t)$ to the corresponding action $a^{f}(t) \in \mathcal{A}^{f}$. Note that, such distributed

[1]Note that, here, $R(\boldsymbol{a}(t), \boldsymbol{\sigma}(t)) = h(t)$, since state $\boldsymbol{\delta}(t)$ captures the existence of each file on every satellites, and the MDP formulation will directly push the requested files to the target user cluster u if these files are available on the serving satellite s.

policies may result in actions $\boldsymbol{a}(t) = \left[a^1(t), \ldots, a^{N_\mathrm{F}}(t)\right]$ that are in conflict, which must be resolved when being implemented in the considered CSN system. Since our goal is to maximize the pre-store hits, we should retain as many "$\{s \to u\}$" actions, and disable the routing actions that are in conflict with "$\{s \to u\}$". Hence, given the probabilistic local policies $\pi^f\left(a^f(t)\,|\sigma^f(t)\right)$, $f = 1, \ldots, N_\mathrm{F}$ and the states $\boldsymbol{\sigma}(t) = \left[\sigma^1(t), \ldots, \sigma^{N_\mathrm{F}}(t)\right]$, the procedure for resolving the conflicts is summarized in Algorithm 1. The procedure incrementally form the conflict-free action set $\mathcal{T}$. First, in Lines 1-9, $\mathcal{T}$ consists of all $\{s \to u\}$ actions and here the only possible conflict is that one satellite may be scheduled to transmit more than one file to the user cluster it connects to. In that case, we allow only one file to be transmitted and let the other files remain on the satellite. Then in Lines 10-15, we examine each action not in $\mathcal{T}$. If it is in conflict with any action in $\mathcal{T}$, it must be either $\{g \to s\}$ corresponding to Eq. (4), or $\{s \to s'\}$ corresponding to Eq. (6), and we simply keep the in-conflict files on their current location, i.e., gateway g, or satellite s.

Remark 1. *Note that from the view point of MDP, in our approach, the global policy is decomposed into independent local policies, i.e.,* $\pi\left(\boldsymbol{a}(t)\,|\boldsymbol{\sigma}(t)\right) = \prod_{f=1}^{N_F} \pi^f\left(a^f(t)\,|\sigma^f(t)\right)$. *Then for each file f, the conflict-resolving Algorithm 1 plays the role of the environment that determines the next state given the current state and action, i.e.,* $p\left(\sigma^f(t+1)\,|\sigma^f(t), a^f(t)\right)$.

Our problem is then to design the local probabilistic policies such that when employed by the above distributed routing algorithm, the maximum pre-store hit rate can be achieved.

In the considered CSN system, the gateways have access to all content files in $\mathcal{F}$ as they are connected to the core network. Targeting at the highest number of pre-store hits, we solve the Dec-MDP to obtain the optimal policy for the routing of content files. Then, the gateways will transmit the files to the satellites, along with the commands on these files' routing strategies. However, we notice that the user requests are hard to satisfy, since 1) the user requests $\mathbf{X} = \{[\boldsymbol{x}_1(t), \ldots, \boldsymbol{x}_{N_\mathrm{U}}(t)], t = 1, \ldots, T\}$, are unknown at the time when the gateways make decisions on the routing processes; 2) the user requests $\mathbf{X}$ are dynamic, as they follow some unknown distributions, $p(\mathbf{X})$; and 3) the user requests are unpredictable, as their distribution $p(\mathbf{X})$ may vary. Thus, traditional MDP solutions such as decision tree search or dynamic programming can not solve this Dec-MDP. The reinforcement learning (RL) algorithms such as Q learning, policy gradient, and echo state networks [22]–[24] can help learn routing strategies in the unknown environment, but are still not suitable for solving this high dimensional Dec-MDP. This is because for each possible realization of user requests $\mathbf{X}$, we need to run the RL algorithm to obtain the corresponding optimal pre-storage and routing strategy, which makes such a solution approach prohibitively complex. Thus, to solve the time sensitive pre-storage and routing problem for dynamic, unpredictable content needs, we propose a distributed distribution-robust meta reinforcement learning (D^2-RMRL) solution with pre-trained meta learning capability.

Algorithm 1 Procedure for resolving conflicts among actions produced by local policies.

Input: Joint state $\boldsymbol{\sigma}(t) = [\boldsymbol{\sigma}_1(t), \ldots, \boldsymbol{\sigma}_{N_\mathrm{U}}(t)]$, local policies $\pi^f\left(a^f(t)\,|\sigma^f(t)\right)$, $f = 1, \ldots, N_\mathrm{F}$.
Init: $\mathcal{T} = \emptyset$.
1: **for** $f = 1, \ldots, N_\mathrm{F}$ **do**
2: Generate local actions $a^f(t)$ according to $\pi^f\left(a^f(t)\,|\sigma^f(t)\right)$.
3: **if** $a^f(t) = \{s \to u\}$ **then**
4: $\mathcal{T} = \mathcal{T} \bigcup \{f\}$.
5: **end if**
6: Check all files in $\mathcal{T}$:
7: **if** more than one file $f_1, \ldots, f_k$ are located on the same satellite s **then**
8: Randomly select one file f_j to be moved to u, and the rest stay on s, , i.e., $\sigma^{f_j}(t+1) = u, \sigma^{f_{j'}}(t+1) = s$, $j' \in \{1, ..., k\} \setminus \{j\}$.
9: **end if**
10: **for** each file $f \notin \mathcal{T}$ **do**
11: **if** $a^f(t)$ is in conflict with any action in $\mathcal{T}$ **then**
12: **if** $a^f(t) = \{g \to s\}$ **then**
13: File f stays on gateway g, i.e. $\sigma^f(t+1) = g$.
14: **else if** $a^f(t) = \{s \to s'\}$ **then**
15: File f stays on satellite s, i.e. $\sigma^f(t+1) = s$.
16: **end if**
17: **else**
18: $\mathcal{T} = \mathcal{T} \bigcup \{f\}$.
19: **end if**
20: **end for**
21: **end for**
22: **return** Next state $\boldsymbol{\sigma}(t+1)$ resulting from actions $\boldsymbol{a}(t) = \left[a^1(t), \ldots, a^{N_\mathrm{F}}(t)\right]$, at state $\boldsymbol{\sigma}(t)$.

III. Distribution Robust Meta Reinforcement Learning Algorithm

We now introduce the D^2-RMRL algorithm, which integrates the techniques of value decomposition [25], model agnostic meta-learning [26], pre-training [27], with the actor-critic RL framework. The value decomposition effectively converts the original problem of finding the global policy $\pi(\boldsymbol{a}(t)|\boldsymbol{\sigma}(t))$ to the one of finding the set of local policies $\left\{\pi_0^f\left(a^f(t)\,|\sigma^f(t)\right), f = 1, \ldots, N_\mathrm{F}\right\}$, thus significantly reduces the computational complexity. The meta training scheme further reduces the cost of computing the optimal policy for every possible service request realization $\boldsymbol{X}$, by first computing a meta policy that can serve as a good initialization for service requests following distribution $p(\boldsymbol{X})$, and then the optimal policy for any realization $\boldsymbol{X} \sim p(\boldsymbol{X})$ can be obtained by a small number of gradient updates. Finally, to address varying user request distributions, the pre-training scheme is adopted that employs the meta training procedure with a parameter transfer technique. In what follows, we first explain how to apply the actor-critic RL algorithm to solve the Dec-MDP using the value decomposition technique. Then we explain

how this RL solution is meta-trained for faster convergence on serving different user requests. Finally, we introduce the pre-training scheme to make the performance of the proposed D^2-RMRL robust to different user request distributions.

A. Actor Critic Method for Computing the Global Policy for a Given User Request Realization $\boldsymbol{X}$

The goal of RL is to find a probabilistic policy π^*: $\Omega \rightarrow \mathcal{A}$ that provides a mapping from the CSN system's states to the actions yielding the highest cumulative discounted reward, i.e., $\pi^* = \arg\max_\pi \mathbb{E}\left[\sum_{t=1}^{T} \gamma^t R(\boldsymbol{a}(t), \boldsymbol{\sigma}(t))\, \pi\left(\boldsymbol{a}(t) \,|\boldsymbol{\sigma}(t)\right)\right]$ where γ is the discount factor and $[\boldsymbol{\sigma}(0), \boldsymbol{a}(0), \boldsymbol{\sigma}(1), \boldsymbol{a}(1), \ldots]$ is a state-action trajectory generated by the policy π. Thus, the action choices in the CSN system must consider the instantaneous reward and the discounted future rewards. So, here we define the (state) value function with a given policy π as

$$V^{\pi}(\boldsymbol{\sigma}(t)) = \mathbb{E}\left[\sum_{\tau=t}^{T} \gamma^{\tau} R(\boldsymbol{a}(\tau), \boldsymbol{\sigma}(\tau))\, \pi\left(\boldsymbol{a}(\tau) \,|\boldsymbol{\sigma}(\tau)\right)\right], \tag{7}$$

which encodes the expected cumulative reward when starting in state $\boldsymbol{\sigma}(t)$ and following the policy π thereafter. The optimal value over all possible policies is

$$V^*(\boldsymbol{\sigma}(t)) = \max_{\pi:\Omega\rightarrow\mathcal{A}} V^{\pi}(\boldsymbol{\sigma}(t)). \tag{8}$$

Then the optimal policy π^* can be obtained by always picking actions $\boldsymbol{a}^*(t)$ that are greedy with respect to V^*, i.e., $\boldsymbol{a}^*(t) = \arg\max_{\boldsymbol{a}(t)\in\mathcal{A}} R(\boldsymbol{a}(t), \boldsymbol{\sigma}(t)) + \gamma V^*(\boldsymbol{\sigma}(t+1))$.

In the actor-critic RL (ACRL) method, both the policy function π_θ and value function V_ψ are deep neural networks parameterized by θ and ψ, respectively. Recall that under the conventional RL, for each realization of user request $\mathbf{X} = \{[\boldsymbol{x}_1(t), \ldots, \boldsymbol{x}_{N_\mathrm{U}}(t)], t = 1, \ldots, T\}$, we need to compute the corresponding policy π. Starting with randomly chosen parameters $\theta^{(0)}$ and $\psi^{(0)}$, the i-th epoch of the ACRL training algorithm consists of the following steps:

1) Sampling: First, using the policy $\pi_{\theta^{(i-1)}}$ from the previous epoch, the algorithm records a sampled state-action-reward trajectory and stores it as $\boldsymbol{\eta} = \{\boldsymbol{\sigma}(t), \boldsymbol{a}(t), R(\boldsymbol{a}(t), \boldsymbol{\sigma}(t)), t \in [1, \ldots, T]\}$.

2) Computing the learning error: The temporal difference (TD) error is defined as [28]

$$\begin{aligned} A(\boldsymbol{a}(t), \boldsymbol{\sigma}(t)) =& R(\boldsymbol{a}(t), \boldsymbol{\sigma}(t)) \\ &+ \gamma V_{\psi^{(i-1)}}(\boldsymbol{\sigma}(t+1)) - V_{\psi^{(i-1)}}(\boldsymbol{\sigma}(t)), \end{aligned} \tag{9}$$

which measures the difference between the achieved reward based on the samples in $\boldsymbol{\eta}$ and the estimated reward, and reveals how actions in $\boldsymbol{a}(t)$ are better than other action choices at the current state $\boldsymbol{\sigma}(t)$. Meanwhile, the error on the routing policy is set as the expected advantage with policy $\pi_{\theta^{(i)}}$ as in

$$\sum_{t=1}^{T} A(\boldsymbol{a}(t), \boldsymbol{\sigma}(t))\, \pi_{\theta^{(i)}}\left(\boldsymbol{a}(t) \,|\boldsymbol{\sigma}(t)\right). \tag{10}$$

3) Updating networks: The critic network $V_{\psi^{(i)}}$, i.e., the value function, is updated toward the opposite direction of the gradient of the squared TD error, $A^2(\boldsymbol{a}(t), \boldsymbol{\sigma}(t))$, for accurate future reward estimation, as

$$\begin{aligned} \psi^{(i)} &= \psi^{(i-1)} - \alpha_c^{(i)} \nabla_\psi \sum_{t=1}^{T} A^2(\boldsymbol{a}(t), \boldsymbol{\sigma}(t)) \\ &= \psi^{(i-1)} + 2\alpha_c^{(i)} \sum_{t=1}^{T} A(\boldsymbol{a}(t), \boldsymbol{\sigma}(t))\, \nabla_\psi V_{\psi^{(i-1)}}(\boldsymbol{\sigma}(t)), \end{aligned} \tag{11}$$

with $\alpha_c^{(i)}$ being the value update step size at the i-th iteration. Notice that, $R(\boldsymbol{a}(t), \boldsymbol{\sigma}(t)) + \gamma \tilde{V}_{\psi^{(i)}}(\boldsymbol{\sigma}(t+1))$ is a supervised term whose partial derivative will not be counted in the gradient update step defined in (11) [29].

At the same time, based on the the policy gradient theorem [24], the ACRL algorithm updates policy function $\pi_{\theta^{(i)}}(\boldsymbol{a}(t) \,|\boldsymbol{\sigma}(t))$ by updating $\theta^{(i)}$ in the direction of the gradient of expected advantage as defined in

$$\begin{aligned} \theta^{(i)} =& \theta^{(i-1)} \\ &+ \alpha_a^{(i-1)} \sum_{t=1}^{T} A(\boldsymbol{a}(t), \boldsymbol{\sigma}(t))\, \nabla_\theta \log \pi_{\theta^{(i-1)}}(\boldsymbol{a}(t) \,|\boldsymbol{\sigma}(t)), \end{aligned} \tag{12}$$

where $\alpha_a^{(i)}$ is the policy update step size, at the i-th iteration of ACRL update. Note that, the ACRL algorithm updates policy and value functions with a mini-batch training procedure, i.e., it implements update after collecting a whole action-state-reward trajectory to reduce the variance on the system's learning performance caused by the action sampling, and the need on storing a big dataset in the system. The ACRL will repeat this trial, error and update procedure until a convergence to the optimal policy π_{θ^*} is reached. However, as we noticed, solving the high dimensional Dec-MDP in such a centralized way has high computational complexity. In the next subsection, we explain how our proposed D^2-RMRL algorithm solves the Dec-MDP with distributed routing policies π_{θ_f} on each file $f \in \mathcal{F}$, using the value decomposition technique.

B. Value Decomposition for Computing the Local Policies for a Given User Request Realization $\boldsymbol{X}$

Our goal is to obtain optimal local policies π_{θ_f} at each file f, which is a deep neural network parametrized by θ_f. Each policy function π_{θ_f} takes file f's local state $\sigma^f(t) \in \Omega^f$ as input, and outputs the probability of action $a^f(t) \in \mathcal{A}^f$ at current state. To update these distributed policies locally for each file f, we decompose the original value function $V_\psi(\boldsymbol{\sigma}(t))$ as the sum of local value functions, given by

$$V_\psi(\boldsymbol{\sigma}(t)) = \sum_{f\in\mathcal{F}} \tilde{V}_{\psi_f}\left(\sigma^f(t)\right), \tag{13}$$

where $\tilde{V}_{\psi_f}\left(\sigma^f(t)\right)$, parametrized by ψ_f, is the local value function associated with file f. Moreover, $\eta_f = \left\{a^f(t), \sigma^f(t), R^f\left(a^f(t), \sigma^f(t)\right), t = 1, \ldots, T\right\}$ is the action-state-reward trajectory generated by the single file routing process at file f under current policy π_{θ_f}. Then

Algorithm 2 Value decomposition-based ACRL algorithm for computing the local policies for a given user request realization $\boldsymbol{X}$.

Input: User service requests $\boldsymbol{x}_u(t)$, $t = 1, \ldots, T$, $u = 1, \ldots, N_{\mathrm{U}}$.

Init: Initialize value functions $\tilde{V}_{\psi_f^{(0)}}$, and policy functions $\pi_{\theta_f^{(0)}}$, for $f = 1, \ldots, N_{\mathrm{F}}$.

1: **for** D^2-RMRL training epoch $i = 1 : I$ **do**
2: Generate sample trajectories of state-action-reward $\eta_f = \left\{\sigma^f(t), a^f(t), R^f\left(a^f(t), \sigma^f(t)\right), t = 1, \ldots, T\right\}$ for each $f = 1, \ldots, N_{\mathrm{F}}$ using the local policies $\pi_{\theta_f^{(i-1)}}\left(a^f(t) \left| \sigma^f(t)\right.\right)$ and Algorithm 1.
3: Calculate TD errors $A\left(\boldsymbol{a}(t), \boldsymbol{\sigma}(t)\right)$, $t = 1, \ldots, T$ in (9).
4: **for** each file $f = 1 : N_{\mathrm{F}}$ **do**
5: Update the local value function parameters $\psi_f^{(i)}$ using (14).
6: Update the local policy function parameters $\theta_f^{(i)}$ using (15).
7: **end for**
8: **end for**
9: **return** Optimal value and policy functions, i.e., ψ_f^*, and θ_f^*.

the update of each value function is given by

$$\begin{aligned}\psi_f^{(i)} &= \psi_f^{(i-1)} - \alpha_c^{(i)} \nabla_{\psi_f} \sum_{t=1}^{T} A^2\left(\boldsymbol{a}(t), \boldsymbol{\sigma}(t)\right) \\ &= \psi_f^{(i-1)} + 2\alpha_c^{(i)} \sum_{t=1}^{T} A\left(\boldsymbol{a}(t), \boldsymbol{\sigma}(t)\right) \nabla_{\psi_f} \tilde{V}_{\psi_f^{(i-1)}}\left(\sigma^f(t)\right).\end{aligned} \tag{14}$$

Thus, using the value decomposition assumption in (13), the gradient of each individual value function consists of a local term that depends on the local state $\sigma^f(t)$ and a global term $A\left(\boldsymbol{a}(t), \boldsymbol{\sigma}(t)\right)$ that depends on the global actions $\boldsymbol{a}(t)$ and states $\boldsymbol{\sigma}(t)$. It is through this global term that the updates of all local value functions are coupled. Even though the update of value function at file f still requires the routing process to know the value of the team advantage $A\left(\boldsymbol{a}(t), \boldsymbol{\sigma}(t)\right)$, the proposed mini-batch training procedure allows the satellite system to calculate the team advantage value, by collecting the value of the collected team reward $R\left(\boldsymbol{a}(t), \boldsymbol{\sigma}(t)\right)$ and original value function $V_\psi\left(\boldsymbol{\sigma}(t)\right)$ without knowing the global states and actions at each time step t. This is because the system only have to update their policies after serving one service occasion. Also, note that, $R\left(\boldsymbol{a}(t), \boldsymbol{\sigma}(t)\right) + \gamma \sum_{f \in \mathcal{F}} \tilde{V}_{\psi_f^{(i-1)}}\left(\sigma^f(t+1)\right)$ is a supervised term whose partial derivative will not be counted in the gradient update step defined in (14). Moreover, based on the the policy gradient theorem [24], the update on file f's policy function parameters is given by

$$\begin{aligned}\theta_f^{(i)} &= \theta_f^{(i-1)} \\ &\quad + \alpha_a^{(i-1)} \sum_{t=1}^{T} A\left(\boldsymbol{a}(t), \boldsymbol{\sigma}(t)\right) \nabla_{\theta_f} \log \pi_{\theta_f^{(i-1)}}\left(a^f(t) \left| \sigma^f(t)\right.\right).\end{aligned} \tag{15}$$

Again we see that the updates of these local policies are coupled due to the global terms $A\left(\boldsymbol{a}(t), \boldsymbol{\sigma}(t)\right)$.

In the considered system, the distributed policy and value functions are stored and optimized on a central controller. The resulting routing policy of file f will be distributed to gateway g, if the routing process of file f starts at gateway g. As summarized in Algorithm 2, starting with initial policy functions $\pi_{\theta_f^{(0)}}$ and value functions $\tilde{V}_{\psi_f^{(0)}}$, the algorithm uses the achieved overall pre-store hits (i.e., achieved reward) and the estimated reward (i.e., values of $\tilde{V}_{\psi_f}\left(\sigma^f(t)\right)$) to calculate the TD error $A\left(\boldsymbol{a}(t), \boldsymbol{\sigma}(t)\right)$. It then uses a mini-batch training mechanism to update policies and reward estimations (value functions) on each file f, independently, with (14) and (15), based a sample trajectory η_f[2]. Such trial, error, then update procedure will be repeated, until the convergence is reached. Recall that the training process needs to be performed for each realization of the user request trajectory $\boldsymbol{X}$. To make the training process more efficient, we next resort to the technique of meta learning, to obtain a good initial network parameters $\left\{\theta_f^{(0)}, \psi_f^{(0)}, f = 1, \ldots, N_{\mathrm{F}}\right\}$, for all $\boldsymbol{X}$ that follow a certain distribution $p(\boldsymbol{X})$, such that starting from such meta-trained iniital parameters, Algorithm 2 will converge in just a few gradient iterations for any user request realization $\boldsymbol{X} \sim p(\boldsymbol{X})$.

C. Meta Training of Model Initials for a Given User Request Distribution $p(\boldsymbol{X})$

In the previous subsection, we gave the ACRL algorithm for a given realization of the user request $\boldsymbol{X}$. In reality, $\boldsymbol{X}$ follows certain statistical distribution $p(\boldsymbol{X})$. It is certainly not feasible to compute the policy for each possible realization of $\boldsymbol{X}$. To that end, we resort to meta learning. The basic idea is to train the networks' initial parameters $\left(\bar{\theta}_f, \bar{\psi}_f\right)$, $f = 1, \ldots, N_{\mathrm{F}}$, for the given distribution $p(\boldsymbol{X})$, such that for any user request realizations $\boldsymbol{X} \sim p(\boldsymbol{X})$, the corresponding optimal networks can be obtained from $\left(\bar{\theta}_f, \bar{\psi}_f\right)$, $f = 1, \ldots, N_{\mathrm{F}}$, through a few gradient update steps with a small amount of training data. In particular, the meta training procedure finds the learning initials, i.e., initial policy functions $\pi_{\bar{\theta}_f}$ and value functions $\tilde{V}_{\bar{\psi}_f}$, $f = 1, \ldots, N_{\mathrm{F}}$, that are already close to the optimal strategies and value estimations of all user request realizations following the distribution $p(\boldsymbol{X})$. Starting from such learning initials, given a user request realization $\boldsymbol{X}$, the corresponding optimal policy and value functions can be obtained by a few actor-critic updates.

At each meta training update epoch, we use J samples of user request $\boldsymbol{X}$, i.e., $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_J \sim p(\boldsymbol{X})$, to update $\bar{\psi}_f$, and $\bar{\theta}_f$ in a similar way as in (14) and (15), except that now J state-action-reward trajectories are used, one for each $\boldsymbol{X}_j$. In particular, we can write, for $f = 1, \ldots, N_{\mathrm{F}}$,

$$\bar{\psi}_f \leftarrow \bar{\psi}_f + 2\alpha_c \sum_{j=1}^{J} \sum_{t=1}^{T} A\left(\boldsymbol{a}_j(t), \boldsymbol{\sigma}_j(t)\right) \nabla_{\bar{\psi}_f} \tilde{V}_{\bar{\psi}_{f,j}}\left(\sigma_j^f(t)\right), \tag{16}$$

[2]The proposed VD-RL algorithm updates the policy and value functions at each file's routing process based only on this routing process' individual action and state with dimension of $|\mathcal{A}|$ and $|\Omega|$, respectively, which effectively reduces the time complexity of the considered multi-agent problem.

$$\bar{\theta}_f \leftarrow \bar{\theta}_f + \alpha_a \sum_{j=1}^{J}\sum_{t=1}^{T} A\left(\boldsymbol{a}_j(t), \boldsymbol{\sigma}_j(t)\right) \nabla_{\bar{\theta}_f} \log \pi_{\bar{\theta}_{f,j}}\left(a_j^f(t) \middle| \sigma_j^f(t)\right), \tag{17}$$

where $\bar{\psi}_{f,j}$, and $\bar{\theta}_{f,j}$ are, respectively, the value and policy function parameters updated at each user request $\boldsymbol{X}_j$. $\sigma_j^f(t)$, and $a_j^f(t)$, are the state and action sampled at time slot t corresponding to user request $\boldsymbol{X}_j$, respectively. To obtain the network parameters $\bar{\psi}_{f,j}$, $\bar{\theta}_{f,j}$, and the state-action-reward trajectory $\eta_{f,j} = \left\{\sigma_j^f(t), a_j^f(t), R^f\left(a_j^f(t), \sigma_j^f(t)\right), t = 1, \ldots, T\right\}$ for each $\boldsymbol{X}_j$, we proceed as follow. First, we sample the state-action-reward trajectory using the current policy function $\pi_{\bar{\theta}_f}$ and user request $\boldsymbol{X}_j$ to obtain $\bar{\eta}_{f,j} = \left\{\bar{\sigma}_j^f(t), \bar{a}_j^f(t), R^f\left(\bar{a}_j^f(t), \bar{\sigma}_j^f(t)\right), t = 1, \ldots, T\right\}$. Then we update the model parameters using one-step gradient descent as, for , $f = 1, \ldots, N_{\mathrm{F}}$

$$\bar{\psi}_{f,j} = \bar{\psi}_f + 2\alpha_c \sum_{t=1}^{T} A\left(\bar{\boldsymbol{a}}_j(t), \bar{\boldsymbol{\sigma}}_j(t)\right) \nabla_{\bar{\psi}_f} \tilde{V}_{\bar{\psi}_f}\left(\bar{\sigma}_j^f(t)\right), \tag{18}$$

$$\bar{\theta}_{f,j} = \bar{\theta}_f + \alpha_a \sum_{t=1}^{T} A\left(\bar{\boldsymbol{a}}_j(t), \bar{\boldsymbol{\sigma}}_j(t)\right) \nabla_{\bar{\theta}_f} \log \pi_{\bar{\theta}_f}\left(\bar{a}_j^f(t) \middle| \bar{\sigma}_j^f(t)\right). \tag{19}$$

Next, we generate sample trajectory $\eta_{f,j}$ using the updated policy $\pi_{\bar{\theta}_{f,j}}$, so as to update learning initials with (16), and (17). As summarized in Algorithm 3, at each meta training iteration, J user request realizations are sampled from $p(\boldsymbol{X})$. Using each realization, $\boldsymbol{X}_j \sim p(\boldsymbol{X})$, for each file f, we obtain the sample trajectory $\bar{\eta}_{f,j}$ using the current policy $\pi_{\bar{\theta}_f}$, and then perform one-step update on the value and policy functions using (18) and (19). Next, using the updated policy $\pi_{\bar{\theta}_{f,j}}$, we obtain the sample trajectory $\eta_{f,j}$. Finally, the initial policy function $\pi_{\bar{\theta}_f}$ and value function $V_{\bar{\psi}_f}$ are updated based on (17) and (16), respectively.

In essence, the above meta training procedure seeks to find the optimal learning initializations, i.e., $\pi_{\bar{\theta}_f^{(0)}} = \pi_{\bar{\theta}_f}$, and $V_{\bar{\psi}_f^{(0)}} = V_{\bar{\psi}_f}$, that are close to the optimal policies and values for all user request realizations. Starting from these initializations, the proposed D^2-RMRL solution, i.e., Algorithm 2, takes only a few iterations to reach convergence for every possible user requests $\boldsymbol{X} \sim p(\boldsymbol{X})$. Given the meta trained policies $\pi_{\bar{\theta}_f}$ and value functions $V_{\bar{\psi}_f}$, in order to obtain the optimal policy for any given $\boldsymbol{X} \sim p(\boldsymbol{X})$, we simply run Algorithm 2 by initializing $\pi_{\bar{\theta}_f^{(0)}} = \pi_{\bar{\theta}_f}$, and $V_{\bar{\psi}_f^{(0)}} = V_{\bar{\psi}_f}$, $f = 1, \ldots, N_{\mathrm{F}}$. Then the number of training epochs needed for reaching convergence is typically small.

D. Pre-training for Distribution Robust Meta Learning

In practice, the user request distribution may vary, e.g., it depends on different times of the day. Assume that there are totally K user request distributions $p_1(\boldsymbol{X}), \ldots, p_K(\boldsymbol{X})$. Then we can simply apply Algorithm 3 to perform meta training for each one of these distributions to obtain the corresponding initials. However, such independent meta training is time-consuming and we would like to make use of the initials already meta trained for some distributions, to speed up the entire meta training process for all distributions. Specifically, suppose meta training is sequentially performed for $p_1(\boldsymbol{X}), ..., p_K(\boldsymbol{X})$. At the beginning of meta training for $p_k(\boldsymbol{X})$, we already have the meta trained initials $\left(\bar{\psi}^\ell, \bar{\theta}^\ell\right)$ for $\ell = 1, \ldots, k-1$, where $\bar{\psi}^\ell = \left[\bar{\psi}_f^\ell, f = 1, \ldots, N_F\right]$, and $\bar{\theta}^\ell = \left[\bar{\theta}_f^\ell, f = 1, \ldots, N_F\right]$. Then, when we perform meta training for $p_k(\boldsymbol{X})$ using Algorithm 3, instead of randomly initializing $\bar{\psi}$ and $\bar{\theta}$, we start with one of the previous meta trained initials, based on the "distance" between each initial to the optimum under $p_k(\boldsymbol{X})$.

In particular, we obtain J samples of user request realizations $\boldsymbol{X}_j \sim p_k(\boldsymbol{X}), j = 1, \ldots, J$. For each of these realizations $\boldsymbol{X}_j$, we apply each available initial policy functions $\pi_{\bar{\theta}_f^\ell}$ to obtain the state-action-reward trajectories $\bar{\eta}_{f,j}^\ell = \left\{\bar{\sigma}_{f,j}^\ell(t), \bar{a}_{f,j}^\ell(t), R\left(\bar{a}_{f,j}^\ell(t), \bar{\sigma}_{f,j}^\ell(t)\right), t = 1, \ldots, T\right\}$, $f =$

Algorithm 3 Meta training for optimal learning initials for a given user request distribution $p(\boldsymbol{X})$.

Input: User request distribution $p(\boldsymbol{X})$.
Init: Initialize value functions $\tilde{V}_{\bar{\psi}_f}$, and policy functions $\pi_{\bar{\boldsymbol{\theta}}_f}$, for $f = 1, \ldots, N_{\mathrm{F}}$.
for Meta training epoch $i = 1 : I$ **do**
  for $j = 1 : J$ **do**
    Sample user request realization $\boldsymbol{X}_j \sim p(\boldsymbol{X})$.
    Generate sample trajectories of state-action-reward $\bar{\eta}_j^f = \left\{\bar{\sigma}_j^f(t), \bar{a}_j^f(t), R^f\left(\bar{a}_j^f(t), \bar{\sigma}_j^f(t)\right), t = 1, \ldots, T\right\}$ using the initial policy functions $\pi_{\bar{\theta}_f}$ and Algorithm 1, for $f = 1, \ldots, N_{\mathrm{F}}$.
    Calculate $A\left(\bar{\boldsymbol{a}}(t), \bar{\boldsymbol{\sigma}}(t)\right)$ in (18) and (19), $t = 1, \ldots, T$ using (9).
    for Each file $f = 1 : N_{\mathrm{F}}$ **do**
      Perform one-step update on the value and policy functions using (18) and (19), to obtain $\bar{\psi}_{f,j}$, and $\bar{\theta}_{f,j}$.
      Generate the state-action-reward trajectory $\eta_j^f = \left\{\sigma_j^f(t), a_j^f(t), R^f\left(a_j^f(t), \sigma_j^f(t)\right), t = 1, \ldots, T\right\}$ using the updated policys $\pi_{\bar{\theta}_{f,j}}$ and Algorithm 1.
    end for
    Calculate $A\left(\boldsymbol{a}_j(t), \boldsymbol{\sigma}_j(t)\right)$ in (16) and (17), $t = 1, \ldots, T$ using (9).
  end for
  for Each file $f = 1 : N_{\mathrm{F}}$ **do**
    Update initial value parameters $\bar{\psi}_f$ using (16), and policy parameters $\bar{\theta}_f$ using (17).
  end for
end for
return Optimal initial policy functions $\pi_{\theta_f^{(0)}} = \pi_{\bar{\theta}_f}$, and initial value functions $\tilde{V}_{\psi_f^{(0)}} = \tilde{V}_{\bar{\psi}_f}$, for $f = 1, \ldots, N_{\mathrm{F}}$.

Algorithm 4 Pre-training for shortened meta training for different user request distributions.

Input: User request distributions $p_1(\boldsymbol{X}), \ldots, p_K(\boldsymbol{X})$.

Init: $k = 1$: Run Algorithm 3 to obtain the meta trained initial $\left(\bar{\psi}^1, \bar{\theta}^1\right)$ for $p_1(\boldsymbol{X})$.

for $k = 2 : K$ **do**
  for environment sampling epoch $j = 1 : J$ **do**
    Sample a user request realization $\boldsymbol{X}_j \sim P_k(\boldsymbol{X})$.
    for treated user request distributions $\ell = 1 : k-1$ **do**
      Generate sample trajectories of state-action-reward $\bar{\eta}_{f,j}^{\ell} = \left\{\bar{\sigma}_{f,j}^{\ell}(t), \bar{a}_{f,j}^{\ell}(t), R\left(\bar{a}_{f,j}^{\ell}(t), \bar{\sigma}_{f,j}^{\ell}(t)\right), t = 1, \ldots, T\right\}$ using the initial policy functions $\pi_{\bar{\theta}_f^{\ell}}$ and Algorithm 1, for $f = 1, \ldots, N_{\mathrm{F}}$.
      Calculate $A\left(\bar{\boldsymbol{a}}_j^{\ell}(t), \bar{\boldsymbol{\sigma}}_j^{\ell}(t)\right)$ in (20) and (21), $t = 1, \ldots, T$, using (9).
      for Each file $f = 1 : N_{\mathrm{F}}$ **do**
        Perform one-step update on the value and policy functions using (20) and (21), to obtain $\bar{\psi}_{f,j}^{\ell}$, and $\bar{\theta}_{f,j}^{\ell}$.
        Generate the state-action-reward trajectory $\eta_{f,j}^{\ell} = \left\{\sigma_{f,j}^{\ell}(t), a_{f,j}^{\ell}(t), R\left(a_{f,j}^{\ell}(t), \sigma_{f,j}^{\ell}(t)\right), t = 1, \ldots, T\right\}$ using the updated policy functions $\pi_{\bar{\theta}_{f,j}^{\ell}}$ and Algorithm 1.
      end for
      Calculate $A\left(\boldsymbol{a}_j^{\ell}(t), \boldsymbol{\sigma}_j^{\ell}(t)\right)$ in (22) and (23), $t = 1, \ldots, T$ using (9).
    end for
  end for
  Compute (22), (23), and then $D_k(\ell)$, for all $\ell = 1, \ldots, k-1$.
  Compute ℓ^*. Run Algorithm 3 using the initializations $\left(\bar{\psi}^{\ell^*}, \bar{\theta}^{\ell^*}\right)$ to obtain the meta trained initials $\left(\pi_{\bar{\theta}_f^k}, \tilde{V}_{\bar{\psi}_f^k}\right)$, for $f = 1, \ldots, N_{\mathrm{F}}$, and $p_k(\boldsymbol{X})$.
end for
return Optimal meta training initials $\left(\bar{\psi}^k, \bar{\theta}^k\right)$, for $k = 1, \ldots, K$.

$1, \ldots, N_{\mathrm{F}}$. Then, we update the model parameters using one-step gradient descent as

$$\bar{\psi}_{f,j}^{\ell} \leftarrow \bar{\psi}_f^{\ell} + 2\alpha_c \sum_{t=1}^{T} A\left(\bar{\boldsymbol{a}}_j^{\ell}(t), \bar{\boldsymbol{\sigma}}_j^{\ell}(t)\right) \nabla_{\bar{\psi}_f^{\ell}} \tilde{V}_{\bar{\psi}_f^{\ell}}\left(\bar{\sigma}_{f,j}^{\ell}(t)\right), \tag{20}$$

$$\bar{\theta}_{f,j}^{\ell} \leftarrow \bar{\theta}_f^{\ell} + \alpha_a \sum_{t=1}^{T} A\left(\bar{\boldsymbol{a}}_j^{\ell}(t), \bar{\boldsymbol{\sigma}}_j^{\ell}(t)\right) \nabla_{\bar{\theta}_f^{\ell}} \log \pi_{\bar{\theta}_f^{\ell}}\left(\bar{a}_{f,j}^{\ell}(t) \left| \bar{\sigma}_{f,j}^{\ell}(t)\right.\right). \tag{21}$$

At the next step, we generate trajectory $\eta_{f,j}^{\ell} = \left\{\sigma_{f,j}^{\ell}(t), a_{f,j}^{\ell}(t), R\left(a_{f,j}^{\ell}(t), \sigma_{f,j}^{\ell}(t)\right), t = 1, \ldots, T\right\}$, using the updated policy $\pi_{\bar{\theta}_{f,j}^{\ell}}$, so as to calculate the "distance" between each initial $\left(\bar{\psi}^{\ell}, \bar{\theta}^{\ell}\right)$ to the optimum under $p_k(\boldsymbol{X})$ as in

$$d_{\theta,f}^{\ell} \triangleq \sum_{j=1}^{J} A\left(\boldsymbol{a}_j^{\ell}(t), \boldsymbol{\sigma}_j^{\ell}(t)\right) \pi_{\bar{\theta}_{f,j}}\left(a_{f,j}^{\ell}(t) \left| \sigma_{f,j}^{\ell}(t)\right.\right), \tag{22}$$

$$d_{\psi,f}^{\ell} \triangleq \sum_{j=1}^{J} \left(R\left(a_{f,j}^{\ell}(t) \left| \sigma_{f,j}^{\ell}(t)\right.\right) - \tilde{V}_{\bar{\psi}_{f,j}}\left(\sigma_{f,j}^{\ell}(t)\right)\right)^2. \tag{23}$$

The distance between the ℓ-th initial and the optimum for $p_k(\boldsymbol{X})$ is then given by $D_k(\ell) = \sum_{f=1}^{N_F}\left(d_{\theta,f}^{\ell} + d_{\psi,f}^{\ell}\right)$, $\ell = 1, ..., k-1$. Then, the meta training procedure for service distribution $p_k(\boldsymbol{X})$ is initialized by $\left(\bar{\psi}^{\ell^*}, \bar{\theta}^{\ell^*}\right)$, where $\ell^* = \arg\min_{\ell} D_k(\ell)$.

The proposed D^2-RMRL with pre-training is summarized in Algorithm 4. For the first service request distribution $p_1(\boldsymbol{X})$, the algorithm randomly starts a meta training procedure, using Algorithm 3. Then for each subsequent new distributions, the algorithm can achieve a shortened meta training by choosing among the learning initials corresponding to distributions that are already meta trained. In particular, it collects experience on serving unseen user request distributions using the available learning initials from treated distributions, then obtains the update with (20) and (21), and evaluates the update with distance metrics in (22) and (23). Finally, it will use the best one among the available learning initials to start the meta training procedure for the current distribution. Through such a transfer process of learned initials, the overall meta training process over multiple user request distributions can be expedited.

E. Convergence and complexity analysis

Next, we first show how Algorithm 2-4 work together using Fig. 3. From Fig. 3, we can see that, for the first unseen service request distribution $p_1(\boldsymbol{X})$, the proposed solution randomly starts a meta training procedure (i.e. Algorithm 3) to find the optimal learning initials that are close to the optimal policies and values for all user request realizations falling within $p_1(\boldsymbol{X})$. Starting from these initials, the proposed RL solution (i.e., Algorithm 2), takes only a few iterations to reach convergence for every user requests $\boldsymbol{X} \sim p_1(\boldsymbol{X})$. Then, for each subsequent new distributions $p_k(\boldsymbol{X})$, the proposed solution uses Algorithm 4 to achieve a shortened meta training (i.e. Algorithm 3) by choosing among the learning initials corresponding to distributions that are already meta trained, i.e., $p_1(\boldsymbol{X}), \cdots, p_{k-1}(\boldsymbol{X})$. For the user requests following $p_k(\boldsymbol{X})$, the proposed solution will still start Algorithm 2 from the meta trained learning initials, for a shortened learning curve toward convergence.

We then analyze the complexity of the proposed D^2-RMRL algorithm. First, we observe that the complexity of each step of policy and value function update in Algorithm 2-4 is, respectively, of $\mathcal{O}(n_c C)$ and $\mathcal{O}(n_a C)$, with n_c, and n_a being, respectively, the number of elements in policy parameters ψ_f, and value parameters θ_f. C is the time complexity needed to calculate the gradient of each element in the policy and value functions. Thus, the complexity of Algorithm 2 is $\mathcal{O}(\upsilon(n_c + n_a)C)$, with υ being the number of iterations the algorithm takes to converge. Note that, this complexity is reasonable because the numbers of policy and value function elements, i.e., n_c, and n_a, are small. The proposed distribution-robust meta training mechanism further reduces

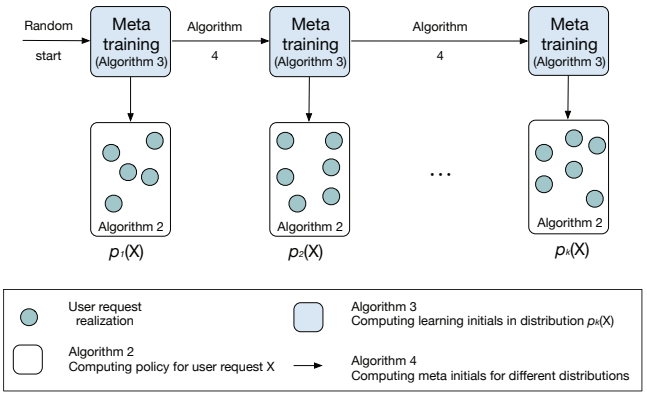

Figure 3. The proposed D^2-RMRL algorithm.

this complexity with faster convergence, i.e., smaller υ. In particular, the distribution-robust meta training mechanism (i.e. Algorithm 3) initializes the RL solution with policy and value functions that are close to the optimal policies and values for all possible user requests at the current user request distribution. The complexity of the distribution-robust meta training procedure (Algorithm 3 associated with Algorithm 4) is $\mathcal{O}\left(2\left(n_c+n_a\right)C\left(J^{\mathrm{P}}+\upsilon_m J\right)\right)$, with $\mathcal{O}\left(2\left(n_c+n_a\right)C\right)$ being the complexity of one step of the meta training on one training sample (i.e., one service experience collected on the gateways). J^{P} and $\upsilon_m J$ are, respectively, the number of service experiences used for pre-training (i.e. Algorithm 4) and meta training (i.e. Algorithm 3) for the current user request distribution. Here, υ_m is the number of iterations that the meta training procedure in Algorithm 3 needs to reach convergence, which is minimized by the pre-training procedure in Algorithm 4. This complexity is reasonable since the offline meta training procedure is implemented only once for every user request distribution $p_k\left(\boldsymbol{X}\right)$, but can speed up the convergence of the VD-RL algorithm for every user requests $\boldsymbol{X}\in p_k\left(\boldsymbol{X}\right)$.

Moreover, we note that the proposed algorithm uses a mini-batch training mechanism that includes an offline training stage and an online usage stage. The offline training is implemented on a central server on the ground. This server knows the coverage dynamics of all satellites and recognizes new user distributions based on user location and service time. No exchange is needed during the offline stage. The ground server runs the algorithm to obtain the local policy for each file f, which is appended to the file. Then during the online stage, wherever the file f is transmitted to a gateway, or a satellite, its host (the gateway or satellite) will read its policy first, and then based on its state σ_f, propose the action $\tilde{a}_f$. It is at this moment some communication overhead will be incurred in order to run Algorithm 1 to resolve the conflicts. Note that, Algorithm 1 can be implemented in two ways. In a centralized approach, all hosts transmit their proposed file actions $\tilde{a}_f$ to a central server, which then runs Algorithm 1 to set the next state for each files on the hosts. On the other hand, Algorithm 1 can also be implemented in a distributed way through exchanges among neighboring hosts, since by definition conflicting actions only occur between adjacent nodes.

IV. Simulation Results

A. Simulation Setup

For our simulations, we consider a scenario with $N_{\mathrm{G}}=5$ gateways serving $N_{\mathrm{U}}=20$ user clusters with the help of a LEO cube satellite constellation at the altitude of 550 km with an inclination of 53°. In particular, these user clusters and gateways fall into the service coverage of $N_{\mathrm{S}}=12$ satellites on 4 intertwined orbits of the constellation. Based on the satellite orbit information in [30] and ground device locations, we construct a time-unrolled data transmission graph to capture the contact chances in the system, within $T=100$ of 10-second time slots. Within this graph, a user cluster or a ground gateway can only contact with its on-duty satellite, i.e., the satellite that is serving their corresponding active cell as in [31]. Meanwhile, we assume that two satellites can communicate only when the distance between them is less than one active cell diameter [30]. Moreover, we assume there are in total $N_{\mathrm{F}}=15$ on-request files in the system. The user request $\boldsymbol{X}=\left\{x_u^f\left(t\right),u=1,\ldots,N_{\mathrm{U}},f=1,\ldots,N_{\mathrm{F}},t=1,\ldots,T\right\}$ are generated as follows. At each time slot, each user cluster u generates m_u file requests, where $m_u\in\{0,1,\ldots,N_F\}$ follows a truncated Poisson distribution with mean $\bar{m}_u$; and these m_u files are random selected out of the N_F files for which we set $x_u^f\left(t\right)=1$. Different user request distributions correspond to different mean values of m.

The value and policy functions of the D^2-RMRL algorithm are both represented by feed forward neural networks, with 2 hidden layers, each is composed of 100 elements. The results of proposed D^2-RMRL algorithm are compared with the ones of the independent actor-critic (IAC) algorithm [32], randomly initialized value decomposition RL solution described in Section III.B (denoted as RL), and meta trained value decomposition RL described in Section III.C (denoted as MRL). Recall that the proposed VD-RL algorithm updates policy and value functions locally at each files based on the global term $A\left(\boldsymbol{a}\left(t\right),\boldsymbol{\sigma}\left(t\right)\right)$, as in (14) and (15). In contrast, the IAC algorithm replaces this global term $A\left(\boldsymbol{a}\left(t\right),\boldsymbol{\sigma}\left(t\right)\right)$ in (14) and (15) with the local term given by $R^f\left(a^f\left(t\right),\sigma^f\left(t\right)\right)+\gamma V_{\psi^f}\left(\sigma^f\left(t+1\right)\right)-V_{\psi^f}\left(\sigma^f\left(t\right)\right)$. Thus, the comparison between IAC and the proposed solution can justify how the proposed value decomposition solution improves distributed data transmission control in the considered satellite network. Meanwhile, the results of the proposed algorithm are also compared to the ones from the RL and MRL solutions, which demonstrate how the proposed distribution-robust meta training mechanism shortens the learning based data transmission design procedures. All statistical results are averaged over a large number of independent runs.

B. Evaluation of Algorithm 2 – Value Decomposition

We first evaluate the performance of value decomposition RL solution in Algorithm 2 for one user request realization $\boldsymbol{X}$. In Fig. 4, we show the convergence behaviors of Algorithm 2 and the IAC method, with the shades indicating results of 1000 runs of the algorithms with random initializations for the same

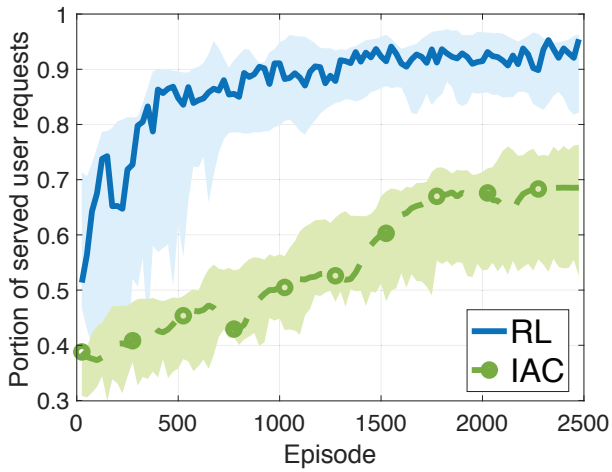

Figure 4. Convergence comparison between Algorithm 2 and the IAC method for a fixed user request realization.

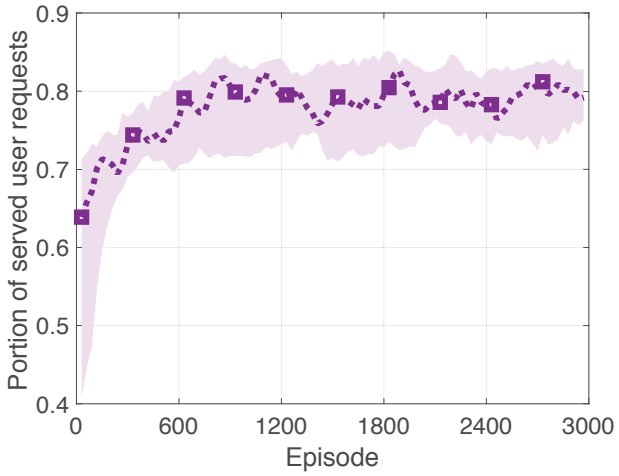

Figure 5. Convergence of Algorithm 3.

user request realization $\boldsymbol{X}$. Fig. 4 shows that, on average, the value decomposition approach proposed in Section III.B yields a 31.8% higher final pre-store hits than the IAC method, as it reinforces strategies that benefit the whole team. On the other hand, the IAC method can only find strategies that maximize the individual utilities. Moreover, from Fig. 4, we also see that Algorithm 2 converges much faster than the IAC method.

C. Evaluation of Algorithm 3 – Meta Training

Next we evaluate how the meta training technique in Algorithm 3 shortens the learning procedure of Algorithm 2 for user requests realizations following $\boldsymbol{X} \sim P(\boldsymbol{X})$, with $\bar{m}_u = 1$ for $u = 1, \cdots, 10$, $\bar{m}_u = 2$ for $u = 11, \cdots, 20$. Firstly, Figure 5 shows the convergence behavior of the meta training procedure, i.e., Algorithm 3 and it is seen that converge can be reached in about 1100 iterations on average. The shades in Figure 5 and Figure 6 indicate results of 1000 independent runs with random sample user requests from distribution $P(\boldsymbol{X})$. Figure 6 shows that, starting from the meta trained learning initials given by Algorithm 3, Algorithm 2 takes about 920 iterations to reach convergence, which improves

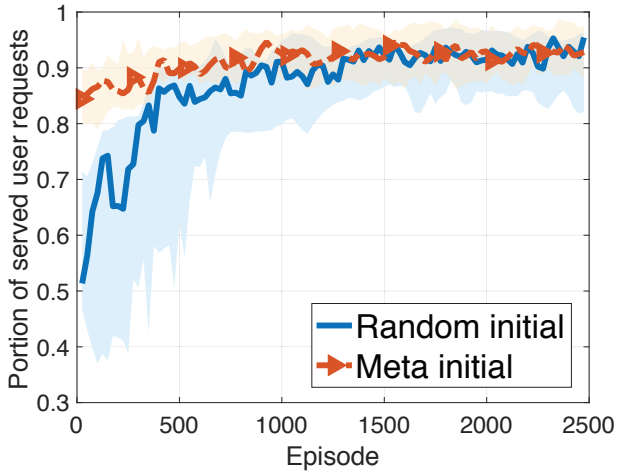

Figure 6. Convergence of Algorithm 2 under meta initial given by Algorithm 3, and random initial.

the convergence speed by up to 40.7% compared to random initials. This stems from the fact that, with the meta training initialization, the RL algorithm can start from policies that are close to the team optimal strategies for the target service task. Moreover, by comparing the shaded areas of the two curves, it is seen that using the meta initial can considerably reduce the variation of the performance of Algorithm 2, in addition to speeding up the convergence.

D. Evaluation of Algorithm 7 – Pre-training

In Fig. 7, we show how Algorithm 4 speeds up the meta training convergence for a family of distributions. We assume that there are $K = 4$ service distributions $P_1(\boldsymbol{X}), \ldots, P_4(\boldsymbol{X})$ and we need to obtain the meta initial for each $P_i(\boldsymbol{X})$. In Fig. 7, we compare the convergence behaviors of two approaches: one is Algorithm 4 and the other is running Algorithm 3 four times one for each $P_i(\boldsymbol{X})$. Recall that in Algorithm 4, for the first distribution $P_1(\boldsymbol{X})$, it simply runs Algorithm 3. Hence the convergence behavior for $P_1(\boldsymbol{X})$ is the same for both approaches. Then, for the other service distributions, $P_2(\boldsymbol{X}), P_3(\boldsymbol{X}), P_4(\boldsymbol{X})$, by making use of the meta initial already obtained for the distribution that is closest to the current one, Algorithm 4, can converge up 43.7% faster than Algorithm 4 which always starts from random initials.

V. Conclusions

In this paper, we have studied the problem of pre-storage and routing data to satellites in a cube satellite network. Using this network, the ground users' dynamic and unforeseeable data requests are served by the cube satellites, which pre-store data from distributed ground gateways, and deliver data service to users in its coverage areas. The design problem is to determine the data to be pre-stored in each satellite and how to route it from a gateway to the satellite. We have formulated this problem as Dec-MDP and have proposed a D^2-RMRL algorithm to solve this problem. The proposed D^2-RMRL algorithm is based on a multi-agent reinforcement learning approach and makes use of the value decomposition technique, so that the agents independently optimize their

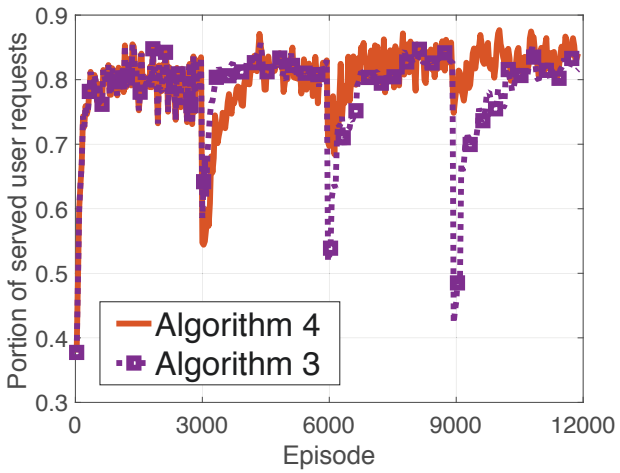

Figure 7. Convergence of meta training for a family of service distributions by Algorithm 4, and Algorithm 3 with random initialization for each distribution.

strategies toward the maximal overall pre-store hits, by sharing only their achieved and estimated reward with each other. To reduce the excessive training cost of this machine learning based solution for different user service requests, we have proposed the meta trainging procedure to obtain initials that can significantly speed up the training process for a given service request distribution, as well as a pre-training procedure for further speedup the meta training procedure for a family of different service request distributions. Simulation results show that the proposed D^2-RMRL algorithm achieves high rate of pre-store hits with fast convergence. Future work could include the consideration of the integration of satellite networks with drone carried aerial networks for extra caching capabilities.

REFERENCES

[1] G. Maral, M. Bousquet, and Z. Sun, *Satellite communications systems: systems, techniques and technology*, John Wiley & Sons, 2020.

[2] M. Sheng, Y. Wang, J. Li, Ru. Liu, D. Zhou, and L. He, "Toward a flexible and reconfigurable broadband satellite network: Resource management architecture and strategies," *IEEE Wireless Communications*, vol. 24, no. 4, pp. 127–133, Aug. 2017.

[3] J. A. Fraire, G. Nies, C. Gerstacker, H. Hermanns, K. Bay, and M. Bisgaard, "Battery-aware contact plan design for LEO satellite constellations: The Ulloriaq case study," *IEEE Transactions on Green Communications and Networking*, vol. 4, no. 1, pp. 236–245, Mar. 2020.

[4] D. Zhou, M. Sheng, X. Wang, C. Xu, R. Liu, and J. Li, "Mission aware contact plan design in resource-limited small satellite networks," *IEEE Transactions on Communications*, vol. 65, no. 6, pp. 2451–2466, Jun. 2017.

[5] J. A Fraire, C. Gerstacker, H. Hermanns, G. Nies, M. Bisgaard, and K. Bay, "On the scalability of battery-aware contact plan design for LEO satellite constellations," *International Journal of Satellite Communications and Networking*, vol. 39, no. 2, pp. 193–204, Mar. 2021.

[6] S. Gu, X. Sun, Z. Yang, T. Huang, W. Xiang, and K. Yu, "Energy-Aware coded caching strategy design with resource optimization for satellite-UAV-Vehicle-Integrated networks," *IEEE Internet of Things Journal*, vol. 9, no. 8, pp. 5799–5811, Mar. 2022.

[7] L. Galluccio, G. Morabito, and S. Palazzo, "Caching in information-centric satellite networks," in *Proc. of IEEE International Conference on Communications (ICC)*, Ottawa,Canada, Jun. 2012, pp. 3306–3310.

[8] A. Armon and H. Levy, "Cache satellite distribution systems: modeling, analysis, and efficient operation," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 2, pp. 218–228, 2004.

[9] X. Liu, "Analysis in big data of satellite communication network based on machine learning algorithms," *Transactions on Emerging Telecommunications Technologies*, vol. 32, no. 7, pp. e3861, Jan. 2021.

[10] A. Gharanjik, M. R. B. Shankar, F. Zimmer, and B. Ottersten, "Centralized rainfall estimation using carrier to noise of satellite communication links," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 5, pp. 1065–1073, May 2018.

[11] P. V. R. Ferreira, R. Paffenroth, A. M. Wyglinski, T. M. Hackett, S. G. Bilen, R. C. Reinhart, and D. J. Mortensen, "Reinforcement learning for satellite communications: From LEO to deep space operations," *IEEE Communications Magazine*, vol. 57, no. 5, pp. 70–75, May 2019.

[12] F. Pacheco, E. Exposito, and M. Gineste, "A framework to classify heterogeneous internet traffic with machine learning and deep learning techniques for satellite communications," *Computer Networks*, vol. 173, pp. 107213, May 2020.

[13] Z. Na, Z. Pan, X. Liu, Z. Deng, Z. Gao, and Q. Guo, "Distributed routing strategy based on machine learning for LEO satellite network," *Wireless Communications and Mobile Computing*, vol. 2018, Jan. 2018.

[14] B. Deng, C. Jiang, H. Yao, S. Guo, and S. Zhao, "The next generation heterogeneous satellite communication networks: Integration of resource management and deep reinforcement learning," *IEEE Wireless Communications*, vol. 27, no. 2, pp. 105–111, Nov. 2020.

[15] Y. Hu, M. Chen, W. Saad, H. V. Poor, and S. Cui, "Distributed multi-agent meta learning for trajectory design in wireless drone networks," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 10, pp. 3177–3192, 2021.

[16] M. Chen, D. GÃ¼ndÃ¼z, K. Huang, W. Saad, M. Bennis, A. V. Feljan, and H. V. Poor, "Distributed learning in wireless networks: Recent progress and future challenges," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 12, pp. 3579–3605, Dec. 2021.

[17] J. Vanschoren, "Meta-learning: A survey," *arXiv preprint arXiv:1810.03548*, 2018.

[18] A. Nichol, J. Achiam, and J. Schulman, "On first-order meta-learning algorithms," *arXiv preprint arXiv:1803.02999*, 2018.

[19] M. Chen, M. Mozaffari, W. Saad, C. Yin, M. Debbah, and C. S. Hong, "Caching in the sky: Proactive deployment of cache-enabled unmanned aerial vehicles for optimized quality-of-experience," *IEEE Journal on Selected Areas on Communications (JSAC), Special Issue on Human-In-The-Loop Mobile Networks*, vol. 35, no. 5, pp. 1046–1061, May 2017.

[20] D. Tse and P. Viswanath, *Fundamentals of wireless communication*, Cambridge university press, 2005.

[21] M. Puterman, *Markov decision processes: discrete stochastic dynamic programming*, John Wiley & Sons, 2014.

[22] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[23] M. Chen, Z. Yang, W. Saad, C. Yin, H. V. Poor, and S. Cui, "A joint learning and communications framework for federated learning over wireless networks," *IEEE Transactions on Wireless Communications*, vol. 20, no. 1, pp. 269–283, Jan. 2021.

[24] R. S Sutton, D. A McAllester, S. P Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proc. of Advances in Neural Information Processing Systems (NIPS)*, Denver, USA, Dec. 2000.

[25] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z Leibo, K. Tuyls, et al., "Value-decomposition networks for cooperative multi-agent learning," *arXiv preprint arXiv:1706.05296*, 2017.

[26] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc. of International Conference on Machine Learning (ICML)*, Sydney, Australia, Aug. 2017.

[27] D. Erhan, A. Courville, Y. Bengio, and P. Vincent, "Why does unsupervised pre-training help deep learning?," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2010, pp. 201–208.

[28] L. C. Baird III, "Advantage updating," Tech. Rep., WRIGHT LAB WRIGHT-PATTERSON AFB OH, 1993.

[29] R S Sutton and A G Barto, *Reinforcement learning: An introduction*, MIT press, 2018.

[30] "Starlink coverage tracker," https://starlink.sx/.

[31] "Starlink daily coverage estimates," https://sebsebmc.github.io/starlink-coverage/index.html.

[32] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018, vol. 32.