IEEE INFOCOM 2021 - IEEE Conference on Computer Communications | 978-1-6654-0325-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/INFOCOM42981.2021.9488806

Mobility- and Load-Adaptive Controller Placement and Assignment in LEO Satellite Networks

Long Chen, Feilong Tang, Xu Li
Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
Emails: nirver1994@sjtu.edu.cn, tang-fl@cs.sjtu.edu.cn, jerry-lx@sjtu.edu.cn

***Abstract*—Software-defined networking (SDN) based LEO satellite networks can make full use of satellite resources through flexible function configuration and efficient resource management of controllers. Consequently, controllers have to be carefully deployed based on dynamical topology and time-varying workload. However, existing work on controller placement and assignment is not applicable to LEO satellite networks with highly dynamic topology and randomly fluctuating load. In this paper, we first formulate the *adaptive controller placement and assignment (ACPA)* problem and prove its NP-hardness. Then, we propose the *control relation graph (CRG)* to quantitatively capture the control overhead in LEO satellite networks. Next, we propose the *CRG-based controller placement and assignment (CCPA)* algorithm with a bounded approximation ratio. Finally, using the predicted topology and estimated traffic load, a lookahead-based improvement algorithm is designed to further decrease the overall management costs. Extensive emulation results demonstrate that the CCPA algorithm outperforms related schemes in terms of response time and load balancing.**

I. Introduction

Software-defined networking (SDN) can efficiently schedule resources and provide programmable function configuration for satellite networks [1] [2]. In SDN-based low-earth-orbit (LEO) satellite networks, multiple controllers are responsible for on-demand route planning [3] [4] and providing customized satellite services [5] [6] for individual control domains. As a result, the *controller placement and assignment* becomes a critical problem since it greatly affects the network performance [7].

Related research has made great progress. However, most schemes place controllers either in geostationary-earth-orbit satellites [8] or on the ground [1] [9], which incur frequent handovers due to the dynamic topology. Although a few of related schemes deploy controllers in the same satellite layer [10], they ignore the time-varying load. Consequently, existing schemes are all inefficient in LEO satellite networks due to the following two new challenges

1) *Highly dynamic topology*. Distance and connectivity between satellites keep changing because LEO satellites orbit the earth at a very fast speed, e.g., the period of Iridium constellations is approximately 100 minutes. Switches need to be dynamically connected to the proper controllers since the continuously-changing and long propagation delay will cause high control overhead. As a result, the controller placement and assignment must be adaptive to the dynamic topology of LEO satellite networks.

Feilong Tang is the corresponding author of this paper.

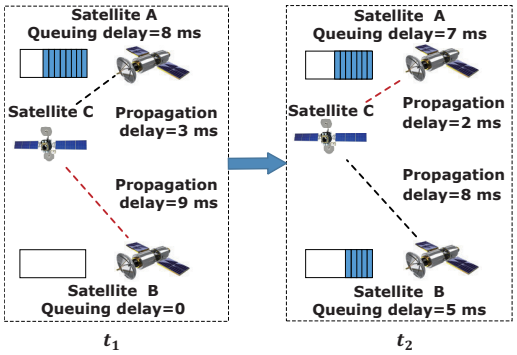


Fig. 1: Adaptive controller placement and assignment

2) *Randomly fluctuating load*. The traffic is unevenly distributed along the Earth's surface, rising when satellites fly over large cities and falling when passing through deserts. The congestion level will also be higher in the daytime due to increased human activities. Choosing a controller with a high load will reduce network management efficiency. Thus, the controller placement and assignment scheme has to adapt to time-varying load.

Hence, the controller placement and assignment in SDN-based LEO satellite networks should jointly consider the mobility and load of satellites. As shown in Fig. 1, satellites A and C act as a controller and a switch, respectively. At t_1, we need to place a new controller in satellite B and assign satellite C to it since the response delay is shorter (9 ms $<$ 3 ms + 8 ms). At t_2, we adaptively assign satellite C to the controller in satellite A since the propagation delays and controllers' loads change (2 ms + 7 ms $<$ 8 ms + 5 ms). The controller in satellite B will be deactivated once the backlog is completed.

In this paper, we investigate how to dynamically place and assign controllers in LEO satellite networks, which adaptively adjusts the number of controllers and management relations in terms of node mobility and traffic load. To estimate the load of controllers, we propose the expected queuing delay model which takes the number and location of switches, as well as the length of backlogs into consideration. Besides, we propose the *control relation graph (CRG)* to quantitatively analyze the management costs of a certain placement and assignment scheme. To the best of our knowledge, this is the first work to jointly consider the effects of mobility and load on the *adaptive controller placement and assignment (ACPA)* problem in LEO satellite networks. The main contributions of

978-1-6654-0325-2/21/$31.00 ©2021 IEEE

this paper are summarized as follows.

1) We formulate the *adaptive controller placement and assignment (ACPA)* problem in SDN-based LEO satellite networks, with an optimization goal of minimizing the management costs tailored for the LEO satellite networks; then, we prove that the ACPA problem is NP-hard.
2) We propose the *control relation graph (CRG)* by encoding the management costs in the ACPA problem, upon which we build the *candidate set* to reduce the problem scale.
3) We propose a *CRG-based controller placement and assignment (CCPA)* algorithm with a bounded approximation ratio, and design a *lookahead-based improvement algorithm* to further utilize the benefits of the predicted topology and traffic load known in advance.

The remainder of this paper is organized as follows. Section II briefly reviews related work. The system model and problem formulation are presented in Section III. In Section IV, the concept of the CRG is proposed. In Section V, we propose a CRG-based controller placement and assignment (CCPA) algorithm. The lookahead-based improvement algorithm is also introduced. Section VI evaluates the CCPA algorithm. We conclude this paper in Section VII.

II. Related Work

The controller placement and assignment problem was first proposed by [11], and existing work falls into two categories.

A. Load-Adaptive Schemes

Most existing load-adaptive schemes were designed for scenario-dependent optimization goals [12] such as control latency and load balancing.

In wide-area networks, authors in [13] first divided the global network topology into a set of subnetworks, and took the minimization of the maximum latency between the switches and controllers in each subnetwork as the goal. To achieve load balancing, [14] and [15] proposed two heuristic algorithms with low complexities, respectively. However, the inter-controller latency plays an important role in the control overhead, especially in elastic distributed controller architectures [16] [17]. Subsequent work such as [18], [19] considered the latency and communication overhead between controllers, but the processing latency of the controller was ignored. A series of methodologies such as s bidirectional matching [20] and the Pareto searching [21], [22] were used to solve the problem, incurring high complexities in large scale networks.

In data center networks, Tao et al. [23] proposed an online placement algorithm to balance the controller's load. The processing latency was carefully modeled based on the queue theory, but they simplified the propagation delays from switches to controllers as hops. In [24], a heuristic placement algorithm was proposed by jointly considering the traffic load and controller's load. As a result, the controller placement in data center networks was mostly converted into the task assignment problem [25] and the virtual machine placement problem [26].

In SDN-enable edge networks, Qin et al. [7] took the controller's load as a key factor to model the synchronization costs, but the controllers were only placed in the static edge nodes. Other related work either focus on the processing latency [27], the maximum response time [28] and average control latency [29], which cannot be applied in satellite networks directly.

In satellite networks, the ground placement schemes [30], [9] caused frequent switches due to the movements of LEO satellites. Bao et al. [8] proposed a redundant scheme to place the controllers in the same layer as the satellite switches, which incurred high control overhead. Authors in [10] proposed a scheme to minimize the flow setup time, but the number of controllers needs to be specified in advance that degrades the adaptation to the varying load.

B. Mobility-Adaptive Schemes

Most existing related work focuses on the user mobility [18] [31] [32]. In [33] and [34], movement patterns of users were taken as key factors of load variance. Authors in [35] proposed a static joint stochastic controller placement and enodeB-controller assignment algorithm based on the variations in the cellular user locations. Liu et al. [36] proposed the controller placement algorithm based on the time-varying propagation delay, but the controller was placed in the ground. To the best of our knowledge, there is little work to place the controller in a dynamic node. Hence, the mobility-adaptive scheme remains an interesting problem to be studied [37] [38].

To conclude, our work in this paper focuses on controller placement and assignment in the LEO satellite networks. We solve it by jointly considering the topology mobility and load. In particular, the locations and number of controllers, as well as the control relation between switches and controllers are dynamically updated.

III. System Model and Problem Formulation

In this section, we first analyze the features of the LEO satellite constellation topology. Then, we introduce the SDN-based LEO satellite network architecture upon which the expected queuing delay model of controllers is proposed. Finally, we analyze three types of management costs under this architecture and formulate the *adaptive controller placement and assignment (ACPA)* problem.

A. LEO Satellite Constellation

1) Basic parameters: The LEO satellite constellation consists of N planes, with M satellites in each plane. Adjacent planes have an angular distance of $\frac{2\pi}{N}$. All nodes in the same plane are separated from each other with an angular distance of $\frac{2\pi}{M}$. We use $\alpha \in (0, \frac{\pi}{2})$ to depict the inclination angle between any plane and the equatorial plane. The *polar area border* $\beta \in (0, \frac{\pi}{2})$ is a predefined threshold to determine the switching behavior of *inter-satellite links (ISLs)*. All nodes move in their planes with a constant angular velocity.

2) Network model: We use the concept of a time-varying graph [39] to model the dynamic network topology as $G = (V, E, \mathcal{T}, \phi, \xi)$, where the last three parameters reflect the time-varying features of satellite networks.

Each node in V represents a satellite, and we have $|V| = NM$. We denote the ith node by $v_i \triangleq v_{(i^p, i^q)}$, where i^p and

i^q are the plane and index numbers, respectively. The angular distance between nodes with the same index in two adjacent planes (e.g., $v_{(i^p,i^q)}$ and $v_{(i^p+1,i^q)}$) is a constant of $\frac{2\pi}{NM}$ [40].

Each edge in E represents an inter-satellite link (ISL). There are at most two intra-plane ISLs and two inter-plane ISLs for a satellite. The intra-plane ISL is established between two adjacent nodes in the same plane and always exists during $\mathbb{T}$. When a satellite's latitude is larger than β or smaller than $-\beta$, its associated inter-plane ISLs are switched off, resulting in topology changes. Based on the predictable satellite topology, we split the period $\mathbb{T}$ into a set of intervals denoted as $\mathcal{T}$. The topology is assumed to be stable during each interval.

The *edge latency* function $\phi : E \times \mathcal{T} \rightarrow \mathbb{R}^+$ gives the time-varying propagation delay for the ISL. The *path latency* function $\xi : V \times V \times \mathcal{T} \rightarrow \mathbb{R}^+$ indicates the shortest time it takes to transmit from one node to another node in terms of *edge latency* along the path. ξ can be implemented by any shortest path algorithm, such as the Dijkstra algorithm.

B. SDN-Based LEO Satellite Network Architecture

We extend the SDN-based satellite network architecture in [41] with three logical layers, as depicted in Fig. 2.

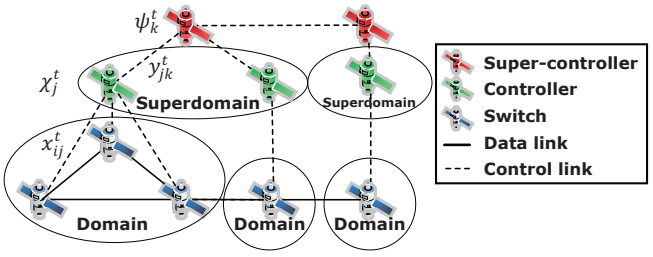


Fig. 2: SDN-based LEO satellite networks

1) Data plane: All satellites act as switches and are connected to ground stations or end hosts directly. We denote them by $S = \{s_1, s_2, ..., s_n\}$, with $n = |V|$[1]. A subset of S forms a *domain* and is managed by an SDN controller.

2) Control plane: This plane contains two kinds of controllers and is located in a subset of satellites in the same constellation as the data plane. We assume that the controller can be installed in all satellites without hardware constraints.

More specifically, controllers provide various network management functions such as routing calculation within domains, denoted by the set $C = \{c_1, c_2, ..., c_m\}$, with $|C| = m$. A set of controllers forms a *superdomain* and is managed by a super-controller, which is another type of controller with greater processing capacity. All super-controllers synchronize with each other at a certain frequency to obtain the state information of other superdomains and provide cross-superdomain services. We denote the set of super-controllers by $SC = \{sc_1, sc_2, ..., sc_p\}$ with $|SC| = p$.

The relations among switches, controllers and super-controllers will be updated at the beginning of each interval of $\mathcal{T}$. We use two binary variables χ_j^t and ψ_k^t to represent whether v_j and v_k are activated as a controller ($\chi_j^t = 1$) and a super-controller ($\psi_k^t = 1$), respectively.

$$\chi_j^t \in \{0,1\}, \psi_k^t \in \{0,1\} \tag{1}$$

Similarly, two other variables, x_{ij}^t and y_{jk}^t are used to indicate whether switch s_i is assigned to controller c_j ($x_{ij}^t = 1$) and whether c_j is assigned to super-controller sc_k ($y_{jk}^t = 1$).

$$x_{ij}^t \in \{0,1\}, y_{jk}^t \in \{0,1\} \tag{2}$$

The data plane is running on the *reactive* mode, which means a packet-in packet will be sent to the controller when a flow fails to match the flow table. We make no assumption on the arrival of routing requests.

We denote the number of new arrival requests at the beginning of each interval $t \in \mathcal{T}$ as γ_i^t. The flow table will be emptied at the end of each interval. Hence, unfinished flows in the last interval are considered as new arrival flows. Using the destination of a request as the classification criterion, for switch s_i, the number of within-domain requests can be calculated as follows

$$\gamma_{1i}^t = |\Upsilon_0|, \Upsilon_0 = \{f \in \Upsilon_i^t | \exists c_j, x_{ij}^t = 1, x_{i'j}^t = 1\}, \tag{3}$$

where Υ_i^t is the set of requests of the switch s_i. The destination of the request f is $s_{i'}$. Hence, the number of cross-domain requests is

$$\gamma_{2i}^t = \gamma_i^t - \gamma_{1i}^t. \tag{4}$$

C. Expected Queuing Delay Model

This model estimates the queuing delay from the perspective of a single switch, which is the duration from when the request is received to when it is processed by the controller. Three key factors are considered, including the current load of the controller, the propagation delay, and the number of switches within the same domain. Similarly, the queuing delay is estimated at the beginning of each interval. In the lookahead-based improvement algorithm, we will extend this model by considering the arrival requests.

We denote the processing capacity by μ_j, which is the number of requests that c_j can process in unit time. The length of backlog is denoted by l_j^t, and we set $l_j^t = 0, \forall c_j$ in the first interval.

In the single-switch scenario, we estimate the queuing delay as $(\frac{l_j^t}{\mu_j} - \xi_{ij}^t)^+$ [2] since it takes at least ξ_{ij}^t for the request from s_i to arrive at c_j.

In multiple-switch scenarios, requests from other switches increase the queue delay of s_i. Since the time of routing calculation is a function of the network size (e.g., the Dijkstra's shortest path algorithm), we estimate the increased queuing delay as a linear function of the square of the number of switches. The expected queuing delay model is expressed as

$$\zeta_{ij}^t = (\frac{l_j^t}{\mu_j} - \xi_{ij}^t)^+ + k(\sum\nolimits_{i'=1}^{n} x_{i'j}^t)^2. \tag{5}$$

[1]We use $|X|$ to represent the size of the set X in this paper

[2]$(a)^+ = \max\{a, 0\}$

The expected queuing delay of the super-controller can be calculated in a similar way as below

$$\zeta_{jk}^t = (\frac{l_k^t}{\mu_k} - \xi_{jk}^t)^+ + k'(\sum\nolimits_{j'=1}^{n} y_{j'k}^t)^2, \tag{6}$$

where k (k') is the added value of the queuing delay for each additional switch (controller) to controller (super-controller). The smaller the k, the stronger the processing capability of the controller. We validate the correctness of this model by experiments in Section VI.

D. Problem Formulation

Our objective is to determine management relations with minimum management costs in each interval of t, which can be divided into three types as detailed below.

1) Operating cost: The operating cost is the sum of propagation delays and queuing delays that all requests have to experience, which can be calculated as follows

$$\begin{aligned} D_O^t = &\sum\nolimits_{j=1}^{m} \chi_j^t \sum\nolimits_{i=1}^{n} x_{ij}^t(\xi_{ij}^t + \zeta_{ij}^t)\gamma_i^t \\ &+ \sum\nolimits_{k=1}^{p} \psi_j^t \sum\nolimits_{i=1}^{n} y_{ij}^t(\xi_{jk}^t + \zeta_{jk}^t)\gamma_{2i}^t \end{aligned} \tag{7}$$

The smaller the operating cost, the smaller the *response time*, which is defined as the duration from when the request is generated to when the response is received by the switch.

2) Switch migration cost: The switch migration cost consists of two parts. First, if the two controllers related to the migrated switch belong to the same super-controller, it is the sum of path latencies between the controller and the super-controller as below

$$D_{M1}^{t^-} = \sum\nolimits_{k=1,k'=1}^{k=p,k'=p} y_{j'k'}^{t-1} y_{jk}^t(\xi_{jk}^t + \xi_{j'k'}^t) \tag{8}$$

Otherwise, the path latency between two super-controllers should be considered as follows

$$D_{M2}^{t^-} = \sum\nolimits_{k=1,k'=2,k\neq k'}^{k=p,k'=p} y_{j'k'}^{t-1} y_{jk}^t \xi_{kk'}^t \tag{9}$$

Hence, the switch migration cost is defined as

$$D_M^{t^-} = \sum\nolimits_{i=1}^{n} \sum\nolimits_{j=1,j'=2,j\neq j'}^{j=m,j'=m} x_{ij'}^{t-1} x_{ij}^t (D_{M1}^{t^-} + D_{M2}^{t^-}) \tag{10}$$

The switch migration cost reflects the load balancing among controllers and indirectly affects the response time due to the queuing delay. As shown in the Formula (10), the switch s_i can be assigned to a new controller c_j in the next slot. Hence, it also captures the cost of initializing a new controller and reassigning different switches.

3) Synchronization cost: The synchronization cost is the sum of path latencies between two super-controllers as

$$D_S^t = \sum\nolimits_{k=1,k'=2,k\neq k'}^{k=p,k'=p} \psi_k^t \psi_{k'}^t \xi_{kk'}^t \tag{11}$$

In the layered architecture introduced above, this kind of cost significantly influences the response time since the synchronization process will delay the processing of cross-domain requests. As a result, the locations of super-controllers and their assignments need to be carefully considered.

Given the management cost model introduced above, we aim to minimize the cost as shown in the Formula (12) by determining the placement ($\{\chi_j^t, \psi_k^t | \forall j,k\}$) and assignment ($\{x_{ij}^t, y_{jk}^t | \forall i,j,k\}$) results at t. The objective precisely and jointly captures the response time improvement and controller load balancing demands in LEO satellite networks. The *adaptive controller placement and assignment (ACPA)* problem is formulated as follows

$$\textbf{minimize} \quad D_O^t + D_S^t + D_M^{t^-} \tag{12}$$

$$s.t. \quad (1) - (11)$$

$$x_{ij}^t \leq \chi_j^t, y_{jk}^t \leq \psi_k^t, \forall i,j,k,t \tag{13}$$

$$\sum\nolimits_{j=1}^{m} x_{ij}^t = 1, \forall i,t \tag{14}$$

$$\sum\nolimits_{k=1}^{p} y_{jk}^t = 1, \forall j,t \tag{15}$$

$$\sum\nolimits_{j=1}^{m} x_{ij}^t(\xi_{ij}^t + \zeta_{ij}^t) \leq L_1^i, \forall i,t \tag{16}$$

$$\sum\nolimits_{k=1}^{p} y_{jk}^t(\sum\nolimits_{j=1}^{m} x_{ij}^t(\xi_{ij}^t + \zeta_{ij}^t) + \xi_{jk}^t + \zeta_{jk}^t) \leq L_2^i, \forall i,t \tag{17}$$

where constraint (13) ensures that only activated controllers and super-controllers can be chosen. Constraint (14) ensures that any switch will be connected to exactly one controller. This is the same between c_j and sc_k in constraint (15). Given a switch s_i, we define L_1^i and L_2^i as two thresholds, representing the maximum tolerable latencies when the request is processed by the controller and by the super-controller, respectively. Constraints (16) and (17) guarantee that the selected controllers and super-controllers satisfy threshold requirements.

Theorem 1. *The ACPA problem is NP-hard.*

Proof. We consider a special case of the ACPA problem in one interval and omit the superscript t . First, we assume that switches will receive at most one request ($\gamma_i \leq 1, \forall i$). Then, we set L_i^i and L_2^i to infinity. The switch migration cost and the synchronization cost are ignored. By treating C and SC as the dominating set over S and C, respectively, the special case of the ACPA problem is equivalent to finding the *minimum weight dominating set (MWDS)*, which is NP-hard [42], by setting weights as $\xi_{ij} + \zeta_{ij}$ and $\xi_{jk} + \zeta_{jk}$. This reduction is done within polynomial time, which completes the proof. □

Considering the NP-hardness, solving the ACPA problem directly by mathematical programming solvers such as CPLEX [43] will involve many variables and is time-consuming. Hence, we will propose an approximation algorithm based on the control relation graph presented as follows.

IV. Control Relation Graph

In this section, we introduce how to build the *control relation graph (CRG)* based on G, which quantitatively reflects the control overhead in LEO satellite networks. We first discuss the construction of candidate sets to reduce the scale of the ACPA problem.

A. Construction of Candidate Set

The candidate set contains the minimum number of nodes including controllers $\widehat{C}$ and super-controllers $\widehat{SC}$ that satisfy the requirements of all switches. However, the selection of controllers and super-controllers is correlated. We first show that minimizing $\widehat{C}$ and $\widehat{SC}$ separately can provide the minimum $|\widehat{C}|+|\widehat{SC}|$, and the order does not matter based on the following analysis. Then, we propose a greedy candidate set construction algorithm.

Definition 1. *Given a directed graph $G(V,E)$, the directed dominating set (DDS) $\mathcal{D}_G \subseteq V$ satisfies that $\forall u \in V$, either $u \in \mathcal{D}_G$ or $\exists v \in \mathcal{D}_G, \overrightarrow{vu} \in E$.*

Definition 2. *Given a directed graph $G(V,E)$ and a set of vertices $V_0 \subseteq V$, the partial dominating set (PDS) $\mathcal{D}_{V_0} \subseteq V$ satisfies that $\forall u \in V_0$, either $u \in \mathcal{D}_{V_0}$ or $\exists v \in D_{V_0}, \overrightarrow{vu} \in E$.*

Lemma 1. $V_0 \subset V_1 \subseteq V$ implies that $\mathcal{D}_{V_0} \subseteq \mathcal{D}_{V_1}$.

Proof. We prove this lemma by contradiction. Suppose that if $V_0 \subset V_1$ then $\mathcal{D}_{V_0} \nsubseteq \mathcal{D}_{V_1}$, which means that $\exists v \in \mathcal{D}_{V_0}$ such that $v \notin \mathcal{D}_{V_1}$.

On the one hand, we know that $\exists u_0 \in V_0$ such that $\overrightarrow{vu_0} \in E$ since $v \in \mathcal{D}_{V_0}$. On the other hand, assume that $u_0 \in V_1$, we have $\overrightarrow{vu_0} \notin E$ since $v \notin \mathcal{D}_{V_1}$, contradicting the fact that $\overrightarrow{vu_0} \in E$. Hence, we have $u_0 \notin V_1$, meaning at least one node u_0 is in V_0 but not in V_1, which contradicts our supposition $V_0 \subset V_1$. Thus, we complete the proof. □

Theorem 2. *$|\widehat{C}|+|\widehat{SC}|$ is minimum if $|\widehat{C}|$ is minimum. Similarly, $|\widehat{C}|+|\widehat{SC}|$ is minimum if $|\widehat{SC}|$ is minimum.*

Proof. We prove this theorem by contradiction. Consider the situation whereby we obtain a solution with $|\widehat{C}| = |\widehat{C}|_{\min} = a, |SC| = b$ and $\min(|C|+|SC|) = a+b$.

Suppose that there is a better solution C', SC' with $C \subset C', |C'| = a' > a, |SC'| = b'$, that satisfies $a'+b' \le a+b$. Since SC and SC' are the PDS of C and C' respectively, e.g., $SC = \mathcal{D}_C$. By lemma 1, we have $SC \subseteq SC'$.

Thus, $a'+b' \le a+b \Rightarrow 0 \le b'-b \le a-a'$, which contradicts the supposition that $a' > a$. In other words, we can never obtain a better solution without finding $|\widehat{C}|_{\min}$.

We omit the proof of the latter half of the theorem since it is similar to the above process. □

Before introducing the construction algorithm, we define some useful concepts as below. $C^-_{i,t}$ and $SC^-_{i,j,t}$ include nodes that are unlikely to satisfy constraints (16) and (17). The control interval matrix CIM records all the intervals during which a node is able to become the controller of another node.

$$C^-_{i,t} = \{v_j | \xi^t_{ij} > L^i_1, \forall v_i, v_j \in V\} \tag{18}$$

$$SC^-_{i,j,t} = \{v_k | \xi^t_{ij} + \xi^t_{jk} > L^i_2, \forall v_i, v_k \in V, v_j \in \{V - C^-_{i,t}\}\} \tag{19}$$

$$CIM[i,j] = \{t | \forall v_i \in V, v_j \in \{V - C^-_{i,t}\}, \forall t\} \tag{20}$$

The construction of $\widehat{C}$ is equivalent to finding the DDS by adding a link $\overrightarrow{v_j v_i}$ between any two nodes v_i and v_j if $CIM[i,j] \neq \phi$. By replacing $C^-_{i,t}$ with $SC^-_{i,j,t}$ in the Formula (20), $\widehat{SC}$ can be obtained by finding the PDS of $\widehat{C}$.

Algorithm 1 Candidate set ($\widehat{C}$) construction

Input: $V, V_c = V, CIM, \mathcal{T}$
Output: $\widehat{C}$
1: **if** $CIM[i,j] \neq \phi, \forall v_i, v_j \in V$ **then**
2: $\quad E_c = E_c \cup \overrightarrow{v_j v_i}$
3: **end if**
4: $I_u = \mathcal{T}, \forall u \in V_c$
5: **for** $v_j \in V$ **do**
6: $\quad d_j = \sum_{i \in V_c : \overrightarrow{v_j v_i} \in E_c} |CIM[i,j]|$
7: **end for**
8: **while** $\exists v_c \in V_c, I_c \neq \emptyset$ **do**
9: $\quad V_0 = \{v_0 | \arg\max_{v_0 \in V} d_{v_0}\}$
10: $\quad$ **if** $V_c \cap V_0 \neq \emptyset$ **then**
11: $\quad\quad v_j = \arg\max_{v_j \in V_c \cap V_0} |I_j|, I_j = \emptyset$
12: $\quad$ **else**
13: $\quad\quad v_j = \arg\max_{v_j \in V_0} |I_j|, I_j = \emptyset$
14: $\quad$ **end if**
15: $\quad$ **for** $v_i \in V_c, \overrightarrow{v_j v_i} \in E_c$ **do**
16: $\quad\quad I_i = I_i - CIM[i,j]$
17: $\quad\quad$ **for** $\overrightarrow{v_k v_i} \in E_c, v_k \in V$ **do**
18: $\quad\quad\quad d_k = d_k - |I_i \cap CIM[i,k]|$
19: $\quad\quad$ **end for**
20: $\quad\quad$ **if** $\exists\ \overrightarrow{v_i v_j} \in E_c$ **then**
21: $\quad\quad\quad d_i = d_i - |CIM[j,i]|$
22: $\quad\quad$ **end if**
23: $\quad$ **end for**
24: $\quad \widehat{C} = \widehat{C} \cup v_j$
25: **end while**

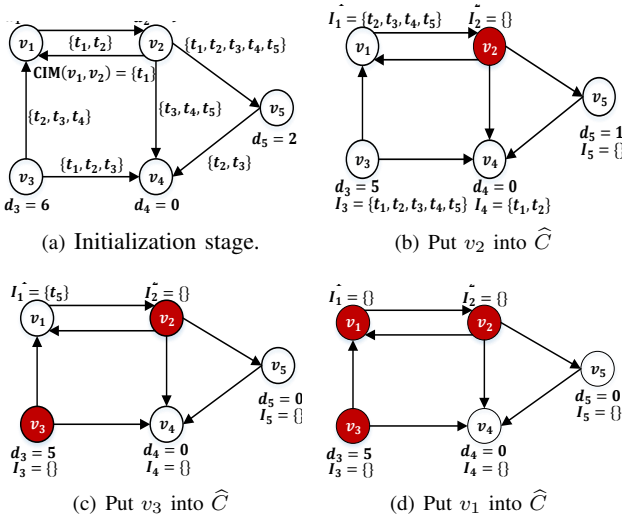
We introduce the construction of $\widehat{C}$ in Algorithm 1, which greedily adds a node v into $\widehat{C}$ until all nodes in V_c are being controlled. By replacing $V_c = V$ with $V_c = \widehat{C}$ in Algorithm 1, the $\widehat{SC}$ can be constructed in a similar manner.

In the initialization phase (lines 1-7), we construct the set of directed edges E_c based on CIM. The *idle interval* I_i is the set of intervals in which v_i is not under control, and we set it to $\mathcal{T}$ for all nodes. The *dominating degree* d_i is the total number of basic intervals during which v_i can be candidate controllers of nodes in V_c.

In the searching phase (lines 9-14), we build the node set V_0 with maximum dominating degree in line 9. Then, depending on whether we are constructing the $\widehat{C}$ ($V_c \cap V_0 \neq \emptyset$) or the $\widehat{SC}$ ($V_c \cap V_0 = \emptyset$), we choose the node with the maximum I_j and empty the set I_j.

The update phase (lines 15-23) includes three steps. The first step is the update of I_i since v_i is able to be controlled by v_j during $t \in CIM[i,j]$. Then, the dominating degrees of the other candidate controllers v_k may decrease. Finally, if v_i is also a candidate controller for v_j, we simply subtract d_i from $CIM[j,i]$ since v_j can always control itself.

We use an example in Fig. 3 to show the construction of $\widehat{C}$. Each directed edge is marked with its corresponding value in CIM. We set V as $\{v_1, v_2, v_3, v_4, v_5\}$ and $\mathcal{T}$ as



(a) Initialization stage. (b) Put v_2 into $\widehat{C}$

(c) Put v_3 into $\widehat{C}$ (d) Put v_1 into $\widehat{C}$

Fig. 3: Construction of $\widehat{C}$

$\{t_1, t_2, t_3, t_4, t_5\}$ in the initialization stage. In Fig. 3(b), we set d_1 as 0 since no node will be controlled by v_1 after v_2 is put into $\widehat{C}$. Furthermore, we put v_1 into $\widehat{C}$ in the last step since it is idle in t_5.

B. Construction of the Control Relation Graph

The *control relation graph (CRG)* consists of $|\mathcal{T}|$ subgraphs represented as $\{\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t), \forall t \in \mathcal{T}\}$. For each subgraph, we denote the two types of nodes in $\mathcal{V}_t$ by $\mathcal{V}_t^c$ and $\mathcal{V}_t^s$. The control node in $\mathcal{V}_t^c$ contains a pair of node indexes from $\widehat{C}$ and $\widehat{SC}$ respectively, with $|\mathcal{V}_t^c| = |\widehat{C}| \cdot |\widehat{SC}|$. The set $\mathcal{V}_t^s$ consists of all switch nodes. They can be represented using

$$\mathcal{V}_t^c = \{v_i | v_i = (j,k) = (v_i^c, v_i^{sc}), \forall v_j \in \widehat{C}, v_k \in \widehat{SC}\} \quad (21)$$

$$\mathcal{V}_t^s = \{v_i | v_i \in V \backslash \{\widehat{C} \cup \widehat{SC}\}\} \quad (22)$$

Links are added between nodes of the above two sets and the weight $\mathcal{W}_{j,i}$ of $\overrightarrow{v_j v_i}$ is encoded based on TABLE I, where v_i^c and v_i^{sc} represent the the first and second element of $v_i \in \mathcal{V}_t^c$, respectively. Symbols with a comma in the subscript have the same value as their previous definitions, e.g., $\xi_{i,j}^t = \xi_{ij}^t$.

We will first consider the estimation of four assignment-dependent variables, upon which the feasibility of weight calculation is analyzed.

First, we discuss the estimation of γ_{2i}^t since its value is influenced by the assignment relations between the controller and other switches. Given a candidate controller c_j in $\widehat{C}$, the number of switches it can connect to is

$$\hat{x}_j^t = |S_j^t|, S_j^t = \{s_i | c_j \notin C_{i,t}^-, s_i \in S\} \quad (23)$$

By assuming that all nodes in the network have the same chance to become destinations of a request, we estimate γ_{2i}^t as a function of the c_j's index as below

$$\hat{\gamma}_{2i}^t(j) = \frac{(n - \hat{x}_j^t) \cdot \gamma_i^t}{n}, \quad (24)$$

where n is the total number of switches.

TABLE I: Calculation of the link weight $\mathcal{W}_{j,i}$

Values	Conditions
$\xi_{v_i^{sc}, v_j^{sc}}^t$	$v_i, v_j \in \mathcal{V}_t^c$
$(\xi_{ij}^0 + \hat{\zeta}_{ij}^0)\gamma_i^0 + \ell_j^0 \hat{\gamma}_{2i}^0(v_j^c)$	$v_i \in \mathcal{V}_0^s, v_j \in \mathcal{V}_0^c$
$(\xi_{i,v_j^c}^t + \hat{\zeta}_{i,v_j^c}^t)\gamma_i^t + \ell_j^t \hat{\gamma}_{2i}^t(v_j^c)$	$v_i \in \mathcal{V}_t^s, v_j \in \mathcal{V}_t^c, x_{i,v_j^c}^{t-1} = 1$
$(\xi_{i,v_j^c}^t + \hat{\zeta}_{i,v_j^c}^t)\gamma_i^t + \ell_j^t \hat{\gamma}_{2i}^t(v_j^c)$ $+\xi_{v_j^c, v_j^{sc}}^t + \xi_{v_{j'}^c, v_{j'}^{sc}}^t$	$v_i \in \mathcal{V}_t^s, v_j \in \mathcal{V}_t^c, v_{j'} \in \mathcal{V}_{t-1}^c,$ $x_{i,v_{j'}^c}^{t-1} = 1, v_j^{sc} = v_{j'}^{sc}$
$(\xi_{i,v_j^c}^t + \hat{\zeta}_{i,v_j^c}^t)\gamma_i^t + \ell_j^t \hat{\gamma}_{2i}^t(v_j^c) +$ $\xi_{v_j^c, v_j^{sc}}^t + \xi_{v_{j'}^c, v_{j'}^{sc}}^t + \xi_{v_j^{sc}, v_{j'}^{sc}}^t$	$v_i \in \mathcal{V}_t^s, v_j \in \mathcal{V}_t^c, v_{j'} \in \mathcal{V}_{t-1}^c,$ $x_{i,v_{j'}^c}^{t-1} = 1, v_j^{sc} \neq v_{j'}^{sc}$

Then, the expected queuing delay is also dependent on the settings of x_{ij}^t. We pessimistically assume that all the candidate switches will connect to the controller. Hence, we replace $\sum_{i'=1}^n x_{i'j}^t$ with $\hat{x}_j^t$ in the Formula (5) to represent the expected queuing delay as

$$\hat{\zeta}_{ij}^t = (\frac{l_j^t}{\mu_j} - \xi_{ij}^t)^+ + k(\hat{x}_j^t)^2 \quad (25)$$

The queuing delays of super-controllers can be estimated in a similar manner by replacing $\sum_{j'=1}^n y_{j'k}^t$ with $\hat{y}_k^t$ in the Formula (6). Finally, we denote the node weight of $v_i \in \mathcal{V}_t^c$ by ℓ_i^t, which can be calculated using

$$\ell_i^t = \xi_{v_i^c, v_i^{sc}}^t + \hat{\zeta}_{v_i^c, v_i^{sc}}^t \quad (26)$$

Theorem 3. *Finding the MWDS on $\mathcal{G}_t$ gives feasible solutions to the ACPA problem.*

Proof. Finding the MWDS on $\mathcal{G}_t$ means that a set of nodes V' is chosen from $\mathcal{V}_t^c$ with minimum total link weights, such that for each node in $\mathcal{V}_t^s$ there exists a directed link from the node of V'.

We discuss the cost encoding process first. When both nodes belong to $\mathcal{V}_t^c$, we set the weight as the synchronization cost between two super-controllers. For links between $\mathcal{V}_t^c$ and $\mathcal{V}_t^s$, if in the first interval, we do not have to count the migration cost; otherwise, we consider whether two controllers after migration belong to the same super-controller and add $\xi_{v_j^{sc}, v_{j'}^{sc}}^t$ to the link weight if they do not ($v_j^{sc} \neq v_{j'}^{sc}$).

Hence, starting from the first interval, we find the MWDS on $\mathcal{G}_t$ and obtain the corresponding placement and assignment results, e.g. if there is a link between $v_j \in \mathcal{V}_t^c$ and $v_i \in \mathcal{V}_t^s$, we can set $\chi_{v_j^c}^t = 1, x_{i,v_j^c}^t = 1, \psi_{v_j^{sc}}^t = 1$ and $y_{v_j^c, v_j^{sc}}^t = 1$, which completes the proof. □

V. Adaptive Controller Placement and Assignment Based on CRG

In this section, the CRG-based controller placement and assignment (CCPA) algorithm is proposed to approach the optimal solution of the ACPA problem, with a bounded approximation ratio. Then, we design the lookahead-based algorithm to further improve the CCPA algorithm by taking advantage of the predicted satellite topology and estimated traffic load.

A. Controller Placement and Assignment Algorithm

We use a binary $1 \times m$ matrix $\boldsymbol{\chi^t}$=$\{\chi_j^t\}$, and a binary $1 \times p$ matrix $\boldsymbol{\psi^t}$=$\{\psi_k^t\}$ to represent the placement results. Similarly, $\boldsymbol{X^t}$=$\{x_{ij}^t\}$ and $\boldsymbol{Y^t}$=$\{y_{jk}^t\}$ are two $n \times m$ and $m \times p$ binary matrices used to indicate the assignment results.

As shown in Algorithm 2, the CCPA algorithm takes $\mathcal{G}_t$ and $\widehat{C}$ as inputs. Firstly, we greedily select the node in $\mathcal{V}_t^c$ with minimum average link weight in line 2, where $deg(v)$ represents the degree of v. Then, we place the controller to v_j^c and assign switches to v_j^c from line 3 to line 8. Next, we place the super-controller in v_j^{sc} and assign v_j^c to super-controller v_j^{sc}. Finally, $\mathcal{V}_t^c$ is updated because a controller can only be connected to a super-controller. the CCPA algorithm repeats until every switch is connected to a controller.

Theorem 4. *The approximation ratio of the CCPA algorithm is* $1+\log(n(nR_{\max}^p + \frac{n^2}{X_{\min}} + 1 + nR_{\max}^m))$*, where* $R_{\max}^p$ *and* $R_{\max}^m$ *are the maximum ratios of the path latency and the migration cost, respectively.* $X_{\min}$ *is the minimum number of switches connected to an activated controller.*

Proof. First, we analyze the management costs gap between the one incurred by the CCPA algorithm and the optimal solution, which is caused by estimations from Formulas (23) to (26) during link weight calculations. Denote the management cost without estimation by adding an overline on the notations, e.g. $\overline{D_O^t}$. We omit indicator variables and derive the gap as

$$\frac{\tilde{D}_O^t + \tilde{D}_S^t + \tilde{D}_M^{t-}}{\overline{D_O^t} + \overline{D_S^t} + \overline{D_M^{t-}}} \leq \frac{\tilde{D}_O^t}{\overline{D_O^t}} + \frac{\tilde{D}_S^t}{\overline{D_S^t}} + \frac{\tilde{D}_M^{t-}}{\overline{D_M^{t-}}} \tag{27a}$$

$$\leq \frac{\tilde{D}_O^t}{\overline{D_O^t}} + 1 + \frac{\sum_i(\tilde{D}_{M1}^{t-} + \tilde{D}_{M2}^{t-})}{\sum_i(\overline{D_{M1}^{t-}} + \overline{D_{M2}^{t-}})} \tag{27b}$$

$$\leq \frac{\sum_i(\tilde{\xi}_{ij}^t + \tilde{\zeta}_{ij}^t)\gamma_i^t}{\sum_i(\overline{\xi_{ij}^t} + \overline{\zeta_{ij}^t})\gamma_i^t + a\gamma_{2i}^t} + 1 + nR_{\max}^m \tag{27c}$$

$$\leq \sum_i(\frac{\tilde{\xi}_{ij}^t}{\overline{\xi_{ij}^t}} + \frac{\tilde{\zeta}_{ij}^t}{\overline{\zeta_{ij}^t}}) + 1 + nR_{\max}^m \tag{27d}$$

$$\leq nR_{\max}^p + \frac{n^2}{X_{\min}} + 1 + nR_{\max}^m, \tag{27e}$$

where (27a) holds since $\frac{\sum_i a_i}{\sum_i b_i} \leq \sum_i \frac{a_i}{b_i}, \forall a_i, b_i > 0$. Based on the assumptions in (23), the worst case occurs when each controller can manage all switches and the maximum number of activated super controllers is 1, e.g., $\hat{x}_j^t = n$ and $\hat{\gamma}_{2i}^t(j) = 0, \forall c_j \in \widehat{C}$. As a result, (27b) and (27c) hold. The (27d) holds if we ignore γ_{2i}^t and (27e) comes from $\frac{\tilde{\xi}_{ij}^t}{\xi_{ij}^t} \leq \frac{n}{\sum_i x_{ij}} \leq \frac{n}{X_{\min}}$.

Then, the CCPA algorithm solves the ACPA problem by finding the MWDS about link weights on $\mathcal{G}_t$ with an approximation ratio of $1 + \log(\Delta|OPT|)$ [44], where Δ is the maximum degree of $\mathcal{G}_t$ and OPT is the optimal solution. Hence, the approximation ratio of CCPA is $1+\log(n(nR_{\max}^p + \frac{n^2}{X_{\min}} + 1 + nR_{\max}^m))$. □

B. Lookahead-Based Improvements

Recent studies on satellite network show that traffic requests show a certain pattern in time [45]. Therefore, assume that the number of requests can be estimated within a lookahead window of one interval, which is denoted as $\hat{\gamma}_i^t$. We design the following lookahead-based improvement algorithm to further improve the CCPA algorithm.

As shown in Algorithm 3, we take results of the CCPA algorithm and the traffic load in t as inputs. First, we estimate the length of c_j's backlog at $t+1$ using

$$\hat{l}_j^{t+1} = l_j^t + (\sum\nolimits_{i=0}^{n} \hat{\gamma}_i^t \chi_j^t x_{ij}^t - \mu_j \times \Delta t)^+, \tag{28}$$

where Δt is the duration of t. χ_j^t and x_{ij}^t can be obtained from results of the CCPA algorithm.

Then, we build $\mathcal{G}_{t+1}$ based on the predicted satellite topology, $\{\hat{l}_j^{t+1}|\forall c_j\}$ and the estimated traffic load $\{\hat{\gamma}_i^t|\forall s_i\}$. We run CCPA based on $\mathcal{G}_{t+1}$. Next, we consider the switch s_i that is expected to be migrated to another controller in the next interval, e.g. $x_{ij}^t = 1$ and $x_{ij'}^{t+1} = 1$. Finally, we try to migrate s_i to $c_{j'}$ in advance and accept this migration if the decrease in management costs in $t+1$ is larger than the increase in t, where D^t is the sum of the three types of costs.

Algorithm 2 CRG-based controller placement and assignment

Input: $\mathcal{G}_t, \widehat{C}$
Output: $\{\boldsymbol{\chi^t}, \boldsymbol{\psi^t}, \boldsymbol{X^t}, \boldsymbol{Y^t}\}$
1: **while** $\exists s_i \in \mathcal{V}_t^s, \forall c_j \in \widehat{C}, x_{ij}^t = 0$ **do**
2: $\quad v_j = \arg\min_{v_j \in \mathcal{V}_t^c} \frac{\sum_{v_i \in \mathcal{V}_t^s, \overrightarrow{v_j v_i} \in \mathcal{E}_t} \mathcal{W}_{j,i}}{deg(v_j)}$
3: $\quad$ **for** $v_i \in \mathcal{V}_t^s$ **do**
4: $\quad\quad$ **if** $\overrightarrow{v_j v_i} \in \mathcal{E}_t$ **then**
5: $\quad\quad\quad \chi_{v_j^c}^t = 1, x_{i,v_j^c}^t = 1$
6: $\quad\quad\quad \mathcal{V}_t^s = \mathcal{V}_t^s - \{v_i\}$
7: $\quad\quad$ **end if**
8: $\quad$ **end for**
9: $\quad \psi_{v_j^{sc}}^t = 1, y_{v_j^c, v_j^{sc}}^t = 1$
10: $\quad \mathcal{V}_t^c = \mathcal{V}_t^s - \{v_k | v_k \in \mathcal{V}_t^c, v_k^c = v_j^c\}$
11: **end while**

Algorithm 3 Lookahead-based improvement

Input: $\{\boldsymbol{X^t}, \boldsymbol{Y^t}, \boldsymbol{\chi^t}, \boldsymbol{\psi^t}\}$,$\{\gamma_i^{t+1}|\forall s_i\}, G$
Output: $\{\boldsymbol{X^t}, \boldsymbol{Y^t}, \boldsymbol{\chi^t}, \boldsymbol{\psi^t}\}$
1: **for** $s_i \in S$ **do**
2: $\quad$ Calculate $\{\hat{l}_j^{t+1}|\forall c_j\}$ based on the Formula (28)
3: $\quad$ Built $\mathcal{G}_{t+1}$ and run CCPA
4: $\quad$ Calculate management costs D^t and D^{t+1}
5: $\quad$ **if** $x_{ij}^t = 1, x_{ij'}^{t+1} = 1, \chi_{j'}^t = 1$ **then**
6: $\quad\quad$ Try migration: $\tilde{x}_{ij}^t = 0$ and $\tilde{x}_{ij'}^t = 1$
7: $\quad\quad$ Recalculate $\{\tilde{l}_j^{t+1}|\forall c_j\}$ by the Formula (28)
8: $\quad\quad$ **if** Assign s_i to $c_{j'}$ satisfying (16) and (17) **then**
9: $\quad\quad\quad$ Recalculate management costs $\tilde{D}^t$ and $\tilde{D}^{t+1}$
10: $\quad\quad\quad$ **if** $\tilde{D}^t - D^t < D^{t+1} - \tilde{D}^{t+1}$ **then**
11: $\quad\quad\quad\quad$ Accept this migration: $x_{ij}^t = 0, x_{ij'}^t$=1
12: $\quad\quad\quad$ **end if**
13: $\quad\quad$ **end if**
14: $\quad$ **end if**
15: **end for**

Theorem 5. *The computational complexity of the lookahead-based improvement algorithm is* $O(n^2mp)$.

Proof. The backlog estimation requires $O(mn)$ computations. The construction of $\mathcal{G}_{t+1}$ and the CCPA algorithm incurs $O(nmp)$ computations. The complexity of calculating management costs is $O(nm + mp)$. The total complexity can be represented as $O(n(nmp + nm + mp)) = O(n^2mp)$ since Algorithm 3 will terminate after n iterations. □

VI. Performance Evaluations

A. System Setup

We implement the system based on ONOS [14] on a server running 64-bit Ubuntu 18.04 with an Intel Gold 6148 2.4GHz CPU, which provides an open-source control plane for managing network components. The data plane is implemented by Mininet [46] on a server running 64-bit Ubuntu 18.04 with an Intel E5-2620 2.1GHz CPU. Each satellite is emulated by an Open vSwitch.

The delays of links are updated by the Linux traffic control command tc at the beginning of each interval. Meanwhile, placement and assignment results are obtained from the CCPA algorithm. The LEO satellite constellation consists of 8 planes and 72 nodes. Each plane contains 9 evenly distributed satellites. We set $L_1^i = 40$ ms and $L_2^i = 70$ ms for all satellites. The inclination angle α and polar area border β are set as 86.4^o and 80^o, respectively. All satellites move at a height of 780 km with an orbital period of 94.5 minutes. According to the above settings, $\mathcal{T}$ consist of 140 intervals, with the length of 60 seconds and 21 seconds appearing alternately.

We run a cluster of ONOS docker images to act as controllers and super-controllers. The processing capacities of the controllers and super-controllers are set as 4×10^3 and 5×10^3 requests/s, respectively. We give each controller and super-controller an index corresponding to the satellite to simulate the placement process. The path latency between switch and controller is set based on their relative locations. The link layer discovery protocol (LLDP) is used.

We generate traffic based on the traffic model in [10]. We repeat every experiment 5 times using different random seeds and obtain the average emulation results. We compare the CCPA algorithm against the following two related schemes:

- *FSTP* [10] places SDN controllers in the LEO layer with the minimum average flow setup time, which is calculated by adding propagation delays between the controller and data plane switches along the path. We assume that the controllers will synchronize with each other to obtain the inter-domain information. The number of activated controllers K is fixed and has to be specified in advance.
- *SAA* [36] is the simulated annealing based algorithm to determine the placement of the controller from a set of locations on the ground, and assigns switches to controllers with minimum propagation delays. We set the elevation angle threshold of establishing ground-satellite links as 85^o.

B. Parameter Settings (k and k')

We run a set of experiments to determine the setting of k in the Formula (5), which is a key parameter to reflect the processing capability of the controller. We activate an ONOS controller image and use Mininet to create the linear topology. All nodes are connected to the remote ONOS controller and begin to *ping* other nodes simultaneously. We vary the number of switches and trace the average queuing delay, which is defined in section III-C. We set k to 0.09 according to Fig. 4. Considering the similarity of the measurement of k', we omit it due to space limitation and set k' to 0.05 experimentally.

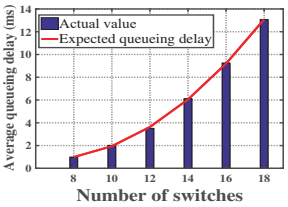


Fig. 4: Parameter setting (k)

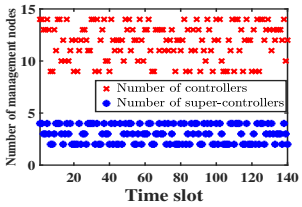


Fig. 5: Number of management nodes with time slots

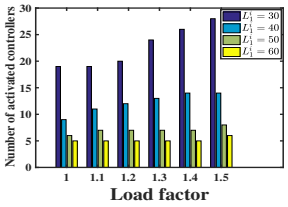


Fig. 6: Number of controllers with load

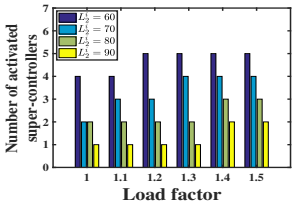


Fig. 7: Number of super-controllers with load

C. Results and Analysis

1) Number of controllers and super-controllers: This set of experiment observes the number of controllers and super-controllers. We multiply the total number of requests by the load factor to reflect the overall load level, which is a real number larger than 1. As shown in Fig. 5, the number of controllers ranges from 9 to 14 and the number of super-controllers varies between 2 and 4. Furthermore, the number of controllers and super controllers increases as the load increases, as depicted in Fig. 6 and Fig. 7.

These results show that the CCPA algorithm can adapt to the dynamic network topology and fluctuating traffic load well by changing the number of management nodes. Note that the CCPA algorithm is more sensitive to the changes of the load when L_1^i is smaller, which indicates the CCPA algorithm can also adapt to the requirements of switches. For example, when the load factor increases and $L_1^i = 30$ ms, the number of activated controllers increases dramatically from 19 to 28. We will show the performance of the CCPA algorithm in terms of average response time and load balance, some of which come from the adjustment of the number of management nodes.

2) Average response time: We compare the performance of CCPA with FSTP and SAA in terms of response time. We use two topologies called Nsfnet and Nsfcnet from Topology Zoo [47] with 13 and 10 nodes, respectively. As shown in Fig. 8,

the average response time of FSTP and SAA increases along with time, while CCPA remains stable.

On the one hand, the activated controller's load varies. On the other hand, the position between the controller and the switch keeps changing over time. However, CCPA has a relatively stable average response time by jointly considering mobility and load. More specifically, when the current path latency or load level cannot meet the management requirements of new arrival requests, CCPA will activate new controllers/super-controllers or update the management relations between nodes.

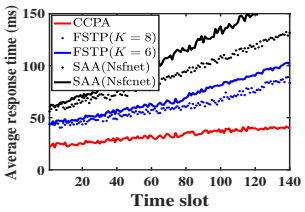


Fig. 8: Response time with time slots

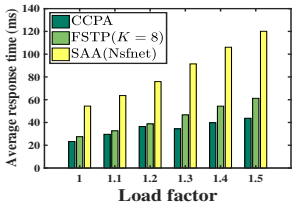


Fig. 9: Response time with load

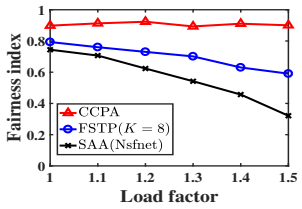


Fig. 10: Fairness index with load

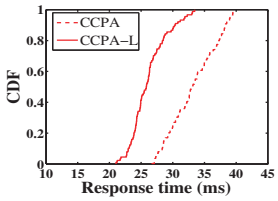


Fig. 11: Lookahead-based improvements on response time

On the contrary, FSTP experiences a higher response delay than CCPA for two reasons. The overall load of controllers under the FSTP scheme is higher than CCPA because FSTP needs to calculate placement and assignment results from a fixed number of nodes. Particularly, FSTP ($K = 6$) suffers a faster increase than FSTP ($K = 8$) about response time. Moreover, the switch may be connected to an overloaded controller since FSTP ignores the queuing delay in the controller.

In the SAA scheme, the packet-in request must be transmitted from the source to the satellite above the elevation angle of the controller first, and then be forwarded to the ground controller. Hence, SAA has a higher overall average response time than FSTP. SAA has the same growth trend as FSTP because it only considers the path latency in terms of propagation delays. At the first interval, CCPA has a lower response delay, even if the controller load is very low. This result reflects the advantages of a layered architecture. In this case, the average number of switches that each controller connects to is less than FSTP and SAA.

Fig. 9 shows the response time with different traffic load. We can see that the average response time of CCPA increases slightly, while SAA (Nsfnet) grows sharply, indicating that CCPA can better adapt to the changing traffic loads. The reason is that the number and locations of candidate controllers are fixed for SAA (Nsfnet). Loads of controllers under SAA keep increasing due to the unreasonable propagation-delay-based assignment strategy. Furthermore, the FSTP scheme has relatively slower growth in response time since it dynamically chooses controllers in each time slot. However, the response time of FSTP gets worse with the increase of load since the number of controllers is fixed.

3) Load balance: We use the fairness index

$$FI = \sum_{t \in \mathcal{T}} \frac{(\sum_{c_j \in C} l_j^t)^2}{m \sum_{c_j \in C} (l_j^t)^2} / |\mathcal{T}| \tag{29}$$

to show the load distribution among multiple controllers. The greater the fairness index, the more uniform the load distribution. As shown in Fig. 10, CCPA significantly outperforms the FSTP and SAA schemes. These results attribute to two reasons. First, CCPA takes the load of the controller into consideration during placement and assignment while the other two methods do not. Second, additional controllers and super-controllers can be dynamically activated by CCPA to balance the overloaded management nodes.

4) Performance gain from lookahead-based improvements: The cumulative distribution function of the response time depicted in Fig. 11 shows that the lookahead-based improvements algorithm (CCPA-L) brings about 25% decrease in the response time. Specifically, the maximum response time decreases from 40 ms to 33 ms.

The reason is that the lookahead-based improvement algorithm makes switch migrations in advance to reduce overall management costs between consecutive time slots. As a result, smaller response delays can be achieved based on the estimated traffic in the next interval and predicted topology.

VII. Conclusion

This paper investigates the *adaptive controller placement and assignment* problem in SDN-based LEO satellite networks. First, with requests arriving randomly in the long propagation delay environment, we propose the expected queuing delay model to estimate the load of the controller. Then, to jointly consider the effects of mobility and load, we propose the concept of *control relation graph (CRG)* to quantitatively represent the control overhead in SDN-based LEO satellite networks. Finally, a *CRG-based controller placement and assignment algorithm* is proposed with a bounded approximation ratio. By taking advantage of predicted topology, we design a *lookahead-based improvement algorithm* to further decrease the total management costs. Extensive emulation results further demonstrate that our CCPA algorithm outperforms other related schemes in terms of response time and load balancing.

Acknowledgements

This work was supported in part by the National Natural Science Foundation of China projects (Nos. 61832013 and 61672351), in part by STCSM (Science and Technology Commission of Shanghai Municipality) AI project (No. 19511120300), in part by Aerospace Fund (No. 20GFC-JJ02-012), in part by the National Key Research and Development Program of China (No. 2019YFB2102204), in part by Alibaba Innovative Research project (No. SCCT622019010803), and in part by the Huawei Technologies Co., Ltd projects (Nos. YBN2019105155 and YBN2020085026). Feilong Tang is the corresponding author of this paper.

REFERENCES

[1] T. Li, H. Zhou, H. Luo, and S. Yu, "Service: A software defined framework for integrated space-terrestrial satellite communication," *IEEE Transactions on Mobile Computing*, vol. 17, no. 3, pp. 703–716, 2017.

[2] F. Tang, H. Zhang, and L. T. Yang, "Multipath cooperative routing with efficient acknowledgement for leo satellite networks," *IEEE Transactions on Mobile Computing*, vol. 18, no. 1, pp. 179–192, 2018.

[3] M. Sheng, Y. Wang, J. Li, R. 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, 2017.

[4] F. Tang, L. T. Yang, C. Tang, J. Li, and M. Guo, "A dynamical and load-balanced flow scheduling approach for big data centers in clouds," *IEEE Transactions on Cloud Computing*, vol. 6, no. 4, pp. 915–928, 2016.

[5] J. Liu, X. Hou, and F. Tang, "Fine-grained machine teaching with attention modeling." in *AAAI 2020*, pp. 2585–2592.

[6] F. Tang, "Bidirectional active learning with gold-instance-based human training." in *IJCAI 2019*, pp. 5989–5996.

[7] G. I. Qiaofeng Qin, Konstantinos Poularakis and L. Tassiulas, "Sdn controller placement at the edge: Optimizing delay and overheads," *IEEE INFOCOM 2018*, pp. 684–692.

[8] J. Bao, B. Zhao, W. Yu, Z. Feng, C. Wu, and Z. Gong, "Opensan: a software-defined satellite network architecture," *ACM SIGCOMM Computer Communication Review*, 2014.

[9] Z. Tang, B. Zhao, W. Yu, Z. Feng, and C. Wu, "Software defined satellite networks: Benefits and challenges," *IEEE Conference on Computing, Communications and IT Applications*, pp. 127–132, 2014.

[10] A. Papa, T. de Cola, P. Vizarreta, M. He, C. Mas Machuca, and W. Kellerer, "Dynamic sdn controller placement in a leo constellation satellite network," *IEEE GLOBECOM 2018*, pp. 1–7.

[11] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," *ACM HotSDN 2012*, pp. 473–478.

[12] G. Wang, Y. Zhao, J. Huang, and W. Wang, "The controller placement problem in software defined networking: A survey," *IEEE Network*, vol. 31, no. 5, pp. 21–27, 2017.

[13] G. Wang, Y. Zhao, J. Huang, Q. Duan, and J. Li, "A k-means-based network partition algorithm for controller placement in software defined network," *IEEE ICC 2016*, pp. 1–6.

[14] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "Onos: towards an open, distributed sdn os," *ACM HotSDN 2014*, pp. 1–6.

[15] A. Ruiz-Rivera, K.-W. Chin, and S. Soh, "Greco: an energy aware controller association algorithm for software defined networks," *IEEE Communications Letters*, vol. 19, no. 4, pp. 541–544, 2015.

[16] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama *et al.*, "Onix: A distributed control platform for large-scale production networks." *OSDI 2010*, pp. 1–6.

[17] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an elastic distributed sdn controller," *ACM SIGCOMM 2013*, pp. 7–12.

[18] A. Ksentini, M. Bagaa, T. Taleb, and I. Balasingham, "On using bargaining game for optimal placement of sdn controllers," *IEEE ICC 2016*, pp. 1–6.

[19] J.-M. Sanner, Y. Hadjadj-Aoul, M. Ouzzif, and G. Rubino, "An evolutionary controllers' placement algorithm for reliable sdn networks," *IFIP ManSDNNFV 2017*.

[20] T. Hu, P. Yi, Z. Guo, J. Lan, and J. Zhang, "Bidirectional matching strategy for multi-controller deployment in distributed software defined networking," *IEEE Access*, vol. 6, pp. 14 946–14 953, 2018.

[21] M. Tanha, D. Sajjadi, and J. Pan, "Enduring node failures through resilient controller placement for software defined networks," *IEEE GLOBECOM 2016*, pp. 1–7.

[22] S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, D. Hock, M. Jarschel, and M. Hoffmann, "Heuristic approaches to the controller placement problem in large scale sdn networks," *IEEE Transactions on Network and Service Management*, vol. 12, no. 1, pp. 4–17, 2015.

[23] T. Wang, F. Liu, and H. Xu, "An efficient online algorithm for dynamic sdn controller assignment in data center networks," *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 2788–2801, 2017.

[24] W. Kim, J. Li, J. W.-K. Hong, and Y.-J. Suh, "Hes-cop: Heuristic switch-controller placement scheme for distributed sdn controllers in data center networks," *International Journal of Network Management*, vol. 28, no. 3, p. e2015, 2018.

[25] F. Tang, "Optimal complex task assignment in service crowdsourcing," in *IJCAI 2020*.

[26] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *IEEE INFOCOM*, 2010, pp. 1–9.

[27] S. Auroux and H. Karl, "Flow processing-aware controller placement in wireless densenets," in *IEEE PIMRC*, 2014, pp. 1294–1299.

[28] M. J. Abdel-Rahman, E. A. Mazied, A. MacKenzie, S. Midkiff, M. R. Rizk, and M. El-Nainay, "On stochastic controller placement in software-defined wireless networks," in *IEEE WCNC*, 2017, pp. 1–6.

[29] E. Borcoci, T. Ambarus, and M. Vochin, "Distributed control plane optimization in sdn-fog vanet," *ICN 2017*.

[30] S. Xu, X.-W. Wang, and M. Huang, "Software-defined next-generation satellite networks: Architecture, challenges, and solutions," *IEEE Access*, vol. 6, pp. 4027–4041, 2018.

[31] F. Tang, H. Zhang, and J. Li, "Joint topology control and stable routing based on pu prediction for multihop mobile cognitive networks," *IEEE Transactions on Wireless Communications*, vol. 17, no. 3, pp. 1713–1726, 2017.

[32] F. Tang, M. Guo, S. Guo, and C.-Z. Xu, "Mobility prediction based joint stable routing and channel assignment for mobile ad hoc cognitive networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 3, pp. 789–802, 2013.

[33] H. Selvi, S. Güner, G. Gür, and F. Alagöz, "The controller placement problem in software defined mobile networks (sdmn)," *Software defined mobile networks (SDMN): beyond LTE network architecture*, pp. 129–147, 2015.

[34] F. Tang, H. Zhang, L. Fu, and X. Li, "Distributed stable routing with adaptive power control for multi-flow and multi-hop mobile cognitive networks," *IEEE Transactions on Mobile Computing*, vol. 18, no. 12, pp. 2829–2841, 2018.

[35] M. J. Abdel-Rahman, E. A. Mazied, K. Teague, A. B. MacKenzie, and S. F. Midkiff, "Robust controller placement and assignment in software-defined cellular networks," in *IEEE ICCCN*, 2017, pp. 1–9.

[36] J. Liu, Y. Shi, L. Zhao, Y. Cao, W. Sun, and N. Kato, "Joint placement of controllers and gateways in sdn-enabled 5g-satellite integrated network," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 2, pp. 221–232, 2018.

[37] L. Chen, F. Tang, X. Li, L. T. Yang, L. Cao, L. Fu, Z. Li, and L. Kong, "Dynamical control domain division for software-defined satellite-ground integrated vehicular networks," *IEEE Transactions on Network Science and Engineering*, early access. doi:10.1109/TNSE.2021.3050213.

[38] B. P. R. Killi and S. V. Rao, "Controller placement in software defined networks: A comprehensive survey," *Computer Networks*, vol. 163, p. 106883, 2019.

[39] A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro, "Time-varying graphs and dynamic networks," *International Conference on Ad-Hoc Networks and Wireless*, pp. 346–359, 2011.

[40] J. Wang, L. Li, and M. Zhou, "Topological dynamics characterization for leo satellite networks," *Computer Networks*, vol. 51, no. 1, pp. 43–53, 2007.

[41] F. Tang, "Dynamically adaptive cooperation transmission among satellite-ground integrated networks," in *IEEE INFOCOM 2020*, pp. 1559–1568.

[42] S. Guha and S. Khuller, "Approximation algorithms for connected dominating sets," *Algorithmica*, vol. 20, no. 4, pp. 374–387, 1998.

[43] I. ILOG, "Cplex optimizer," *En ligne]. Available: http://www-01. ibm. com/software/commerce/optimization/cplex-optimizer*, 2012.

[44] V. Chvatal, "A greedy heuristic for the set-covering problem," *Mathematics of operations research*, vol. 4, no. 3, pp. 233–235, 1979.

[45] Z. Zhang, C. Jiang, S. Guo, Y. Qian, and Y. Ren, "Temporal centrality-balanced traffic management for space satellite networks," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 5, pp. 4427–4439, 2017.

[46] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010.

[47] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.