SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

October 2026, Vol. 69, Iss. 10, 202301:1–202301:17
https://doi.org/10.1007/s11432-025-4925-y

CANEG: a polynomial-time graph model for optimal routing in computing-aware networks

Fei LIU[1], Xiushe ZHANG[2*], Peng WANG[3], Hongyan LI[1] & Zonghuan GUO[1]

[1]*State Key Laboratory of Integrated Service Networks, Xidian University, Xi'an 710071, China*
[2]*The 20th Research Institute of China Electronics Technology Group Corporation, Xi'an 710071, China*
[3]*Information Systems Technology and Design, Singapore University of Technology and Design, Singapore 487372, Singapore*

Received 4 December 2025/Revised 12 March 2026/Accepted 16 April 2026/Published online 18 June 2026

Abstract With the rapid advancement in computing technologies, an increasing number of networked applications require processing within the network while data is transmitted from source to destination. This has led to the emergence of computing-aware networks (CANs), which integrate network routing and in-network computation to efficiently serve latency- and compute-sensitive applications. However, jointly optimizing routing and compute node selection in CANs remains a fundamental challenge due to the inherent coupling between data delivery and computation decisions. Existing methods either suffer from high computational complexity or rely on heuristics without optimality guarantees. To address this, we propose the CAN-expanded graph (CANEG), a novel three-layer graph model that provides a complete and lossless representation of all feasible routing and computing options in CANs. Leveraging this structure, we transform the original integer linear programming (ILP) problem into a polynomial-time shortest path problem, solvable optimally using classic algorithms. Numerical experiments demonstrate that our CANEG-based method achieves optimal solutions while significantly reducing computation time by up to 8.1 times compared to ILP-based solvers and up to 3.8 times compared to heuristic approaches.

Keywords computing-aware networks, joint routing and computing, ILP, CAN expanded graph model, latency optimization

Citation Liu F, Zhang X S, Wang P, et al. CANEG: a polynomial-time graph model for optimal routing in computing-aware networks. Sci China Inf Sci, 2026, 69(10): 202301, https://doi.org/10.1007/s11432-025-4925-y

1 Introduction

With the rapid development of chips' computing power, a massive number of computing applications have emerged. Such applications impose stringent requirements on both data transmission and in-network computation, including virtual reality (VR), intelligent transportation, distributed AI training and inference, and so on [1]. Computing-aware network (CAN) emerges as a new network architecture to accommodate such applications, as reflected in the ongoing IETF standardization studies [2]. In a CAN, the network is aware of the load status and computing capabilities of compute sites, thereby enabling joint orchestration of communication and compute resources. In this work, we study the routing problem in CAN, which aims to identify a delivery-compute-delivery (DCD) path for a single application that satisfies the computing requirements of the application while minimizing the end-to-end (E2E) latency [3]. Specifically, in a VR scenario, routing in a CAN entails transmitting raw data from the source to an appropriate compute node for in-network processing, including stitching, extraction, projection, and rendering, and then delivering the processed data to the end users of VR applications. As such, the optimal route in a CAN accounts for both the selection of the compute node and its corresponding data transmission path, different from the existing shortest path problem [4]. Moreover, to support seamless experiences for latency-sensitive applications, the routing scheme must achieve ultra-low and deterministic E2E delay [5,6]. This requires fast and optimal decisions on both routing and compute node selection in CANs.

There are already some studies studying the routing problem, which cannot be both fast and optimal. For instance, the authors in [7–12] focused on minimizing data delivery time, neglecting the impact of compute node selection, leading to a sub-optimal E2E delay in our case. The authors in [13–17] adopted two-stage approaches-either selecting a compute node first and then determining the delivery path [16,17], or identifying candidate paths first and then selecting the optimal compute node along each path [13–15]. While heuristic, such decoupled strategies often lead to suboptimal results due to the lack of joint decision-making. In fact, joint optimization across multiple

* Corresponding author (email: xszhang163@163.com)

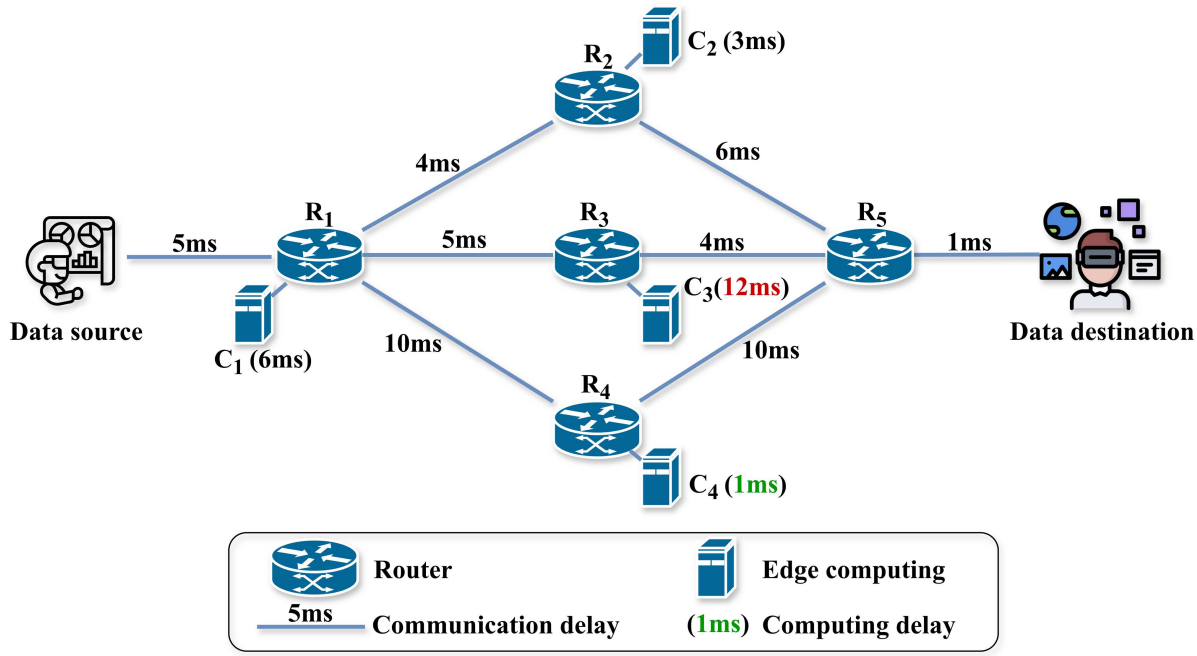


Figure 1 (Color online) An example demonstrating the need for joint routing and computing scheduling.

resources has been shown to significantly improve network performance [18]. As a result, the authors in [19–24] formulated the co-optimization problem as an integer linear programming (ILP) problem or mixed-integer nonlinear programming (MINLP) problem to obtain optimal solutions. However, their high computational complexity may cause significant performance degradation in time-sensitive scenarios.

It is non-trivial to design such a fast and optimal scheme for CAN. Figure 1 shows a simplified example to further illustrate the reasons. Specifically, the considered CAN includes a source, a destination, routers $(R_1, R_2, \ldots, R_5)$ and attached mobile edge computing (MEC) sites (C_1, C_2, C_3, C_4). We label the communication delay on each link and the compute delay in each node of the MEC site. The CAN needs to transmit the data from the source node to one MEC site for processing and then transmit the processed data to the destination node. To quickly solve the problem, one intuitive way is to first adopt the Dijkstra algorithm to find the shortest communication delay path between the source and destination (i.e., source $-$ R_1 $-$ R_3 $-$ R_5 $-$ destination) and then choose the most powerful MEC site (i.e., C_1) along the path to process data, leading to a total delay of (15+6) ms. Another intuitive way is to choose an MEC with the smallest compute delay (i.e., C_4 with 1 ms) and then find two shortest paths from source to R_4 and from R_4 to destination, respectively. This method finally finds the path source $-$ R_1 $-$ R_4 $-$ R_5 $-$ destination with an E2E delay of 27 ms. In contrast, a better solution, requiring only 19 ms, chooses C_2 to process data and selects path source $-$ R_1 $-$ R_2 $-$ R_5 $-$ destination for data delivery. This reveals that the fast intuitive methods may fail to find the optimal solution.

The root of this difficulty lies in the inherent coupling between routing and computing decisions. Selecting a specific compute node inevitably constrains the available routing space, while routing decisions similarly restrict where computation can occur. A straightforward way to address this coupling is to enumerate candidate routes and compute node assignments and evaluate the E2E delay for each combination [13,15,17]. For example, as illustrated by the example in Figure 1, there are three distinct transmission paths and each path includes two candidate compute nodes, resulting in a total of six feasible combinations. By exhaustively enumerating these six path-compute node combinations, we can evaluate the E2E delay for each and identify the one with the minimum total delay, thereby finding the optimal solution. However, as the network size grows, the number of such combinations increases exponentially, leading to significant computational overhead. This observation highlights the core insight of our work: an integrated approach that jointly optimizes routing and compute node selection can effectively reduce computational complexity while achieving efficient E2E performance.

In this paper, we propose a CAN-expanded graph (CANEG) model to optimally solve the considered joint optimization problem within polynomial time. The proposed CANEG model can accurately capture all feasible scheduling options—including all source-to-compute node subpaths, compute node selections, and compute-to-destination node subpaths—within a three-layered graph structure. Consequently, we transform what would otherwise be a

complex ILP problem into a shortest path problem in CANEG. As a result, CANEG achieves optimality while dramatically reducing computational complexity compared to prior methods. The specific contributions are as follows.

- We formulate the joint optimization problem as an ILP problem. This formulation decomposes the full path into two subpaths where the subpath selections are tightly coupled with the compute node selection. This structure allows the model to explore all feasible joint combinations efficiently, yielding optimal solutions even when loops are permitted in the final routing path.
- We propose the CANEG model, which models the CAN into a three-layered graph that accurately characterizes computing constraints, the dependency between computing and communication resources, and the delivery-compute-delivery process.
- We theoretically prove that the proposed CANEG provides a complete and lossless representation of all feasible DCD paths in CAN. Based on this property, we can optimally solve the ILP problem by using a Dijkstra algorithm to find the shortest path in CANEG. This transformation significantly reduces the computational complexity from exponential time (required by ILP solvers) to polynomial time, specifically $O(|E| + |V| \log |V|)$.
- We conduct extensive simulations on CANs of varying sizes and densities. The results show that the CANEG-based scheme achieves the same optimal routing solutions as ILP, while significantly reducing running time by up to 8.1× compared to ILP-based solutions and up to 3.8× compared to KSP-based heuristics.

2 Related work

To fulfill the transmission and in-network processing requirements of latency- and compute-intensive tasks, a CAN must jointly orchestrate routing and compute node selection, forming a DCD process. Existing literature on this problem can be broadly categorized into three groups: (1) single-resource optimization, (2) two-stage heuristic methods, and (3) integrated optimization based on mathematical programming.

(1) Single-resource optimization. Early studies focus either on routing path optimization [7–9] or on compute offloading [25–27], optimizing communication or computing in isolation. These approaches are unable to capture the interaction between transmission paths and compute site placement, often leading to suboptimal E2E delay.

(2) Two-stage heuristic methods. To reduce computational overhead, many recent studies decouple the joint routing and computing problem into two stages. Some first determine candidate paths and then assign compute nodes [13,14], while others select compute nodes prior to path planning [15–17]. Although these approaches simplify the problem, their lack of global coordination leads to suboptimal performance. For instance, the authors in [13] preselected K candidate paths using the K-shortest paths (KSP) algorithm. For each path, they evaluate candidate compute nodes based on their ability to execute the assigned service. A collaborative scheduling algorithm is then used to match computing tasks with candidate compute nodes, forming the final mapping. While this method considers compute-node suitability, it lacks global optimality guarantees, as preselected paths may exclude the true optimum. Similarly, the authors in [14] constructed a legal path-node pair set and prioritized selection based on dual-priority: one for minimizing delay and the other for balancing long-term load. Candidate nodes are mapped onto available paths, but due to the discrete nature of mapping, there is no guarantee that selected nodes fall on selected paths. The scheme improves average delay but cannot ensure feasibility across all combinations. In [15], the authors used a genetic algorithm to search for optimal path-node pairs by evaluating compute node candidates based on both communication and computation delay. They generate initial solutions by selecting nodes with the lowest compute delay and computing their respective source-to-node and node-to-destination paths. While this exhaustive approach helps avoid naive decoupling, its iterative nature makes it computationally expensive in large-scale networks. Research [16] exploited probabilistic modeling of the users' movement to pre-allocate the computing resources at base stations and to select suitable communication paths between the users and the base station with the pre-allocated computing resources. The authors in [17] ranked compute nodes by resource metrics (e.g., residual CPU, availability) and then selected a communication path with the best delay from a routing table. However, their routing decisions are not influenced by the compute node location, leading to potential mismatches between communication and computation and suboptimal E2E delay.

(3) Integrated optimization via ILP/MINLP. To achieve optimality, several studies formulate the joint optimization problem as an ILP [19] or MINLP [20–23]. Specifically, the authors in [21] addressed delay-sensitive offloading in MEC networks and formulated the problem as an MINLP. They applied a sequential fixing method to reduce the solution space. In [22], the authors proposed a database placement and routing optimization strategy for MEC, which is then reformulated as an MILP. The authors in [23] developed a two-timescale DRL scheme to approximate the MINLP solution, trading optimality for tractability. Although theoretically optimal, these

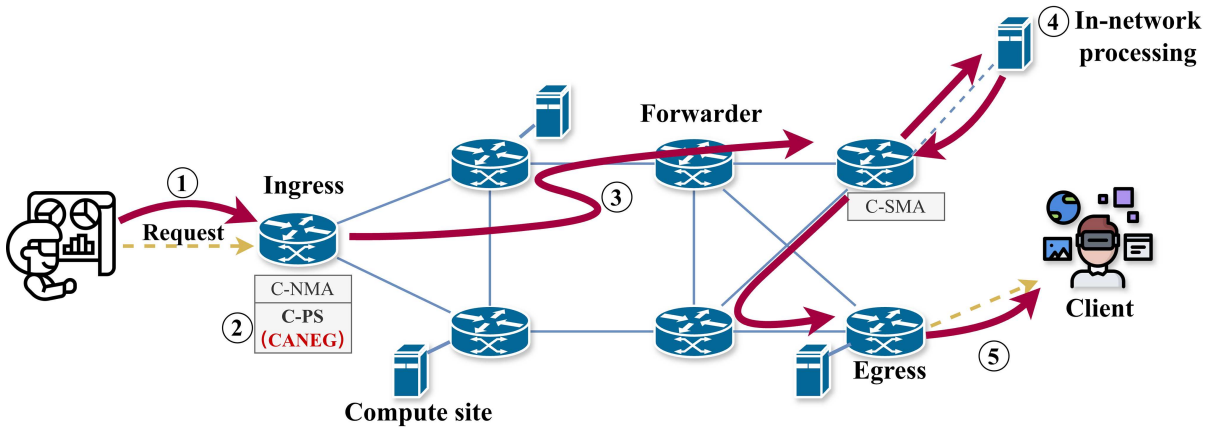


Figure 2 (Color online) The CAN framework.

formulations are computationally intractable in large-scale networks. Various techniques have been proposed to mitigate complexity, including sequential fixing [21], MILP approximations [22], or DRL-based relaxation [23], yet they remain unsuitable for flow-level, latency-sensitive decision-making. Others adopt convex approximation methods [24,28–30] that simplify the problem by relaxing non-convex constraints but compromise modeling accuracy. For example, Ref. [29] used successive convex approximation (SCA) to iteratively solve the joint resource scheduling problem, converting the non-convex formulation into a sequence of convex subproblems. Ref. [30] proposed a bi-convex decomposition-based method to jointly minimize average delay and energy. However, both approaches require convergence over multiple iterations and may get stuck in local minima.

(4) Graph-based modeling studies. Graph theory is widely used to support network modeling and path search [10–12, 31, 32]. The work in [31] transformed each compute node into an edge with a delay weight representing its processing time, thereby allowing routing algorithms to indirectly account for computation. However, this transformation fails to enforce the DCD structure, potentially selecting zero or multiple compute nodes, which contradicts the requirements of CAN. The authors in [32] constructed a compute-node-expanded graph by duplicating the entire network topology per compute node candidate. This enables resource-aware path selection, but results in a large and memory-intensive graph, limiting its scalability. For satellite networks, time-expanded graphs (TEG) have been proposed [10–12] to capture time-varying topology and link conditions. Although these models accommodate link dynamics, they do not incorporate computing resource awareness and are tailored for temporal routing scenarios, rather than in-network computation as required in CAN.

Overall, existing methods either rely on decoupled heuristics or incur high complexity via mathematical programming. Few of them model the DCD constraint explicitly, nor do they provide a scalable and globally optimal solution suitable for real-time CAN routing. In contrast, this work proposes the CANEG, the first graph model that explicitly encodes all feasible DCD paths in a single three-layer graph. This model enables the transformation of the original ILP problem into a shortest-path problem solvable in polynomial time, achieving global optimality with dramatically reduced complexity.

3 System model and problem formulation

3.1 System framework

Figure 2 illustrates the considered CAN framework, which is designed in alignment with the IETF computing-aware traffic steering (CATS) architecture [2]. It supports both centralized and distributed deployment modes and enables joint routing and compute node selection in CANs. The entire process consists of five key steps.

(1) **Request initiation.** A computing task is generated by a user-side application (e.g., VR headset or AI client) and forwarded to the CAN's ingress router. This task contains both data payload and service requirement metadata.

(2) **Ingress processing.** At the ingress, the C-PS (CATS path selector) module is responsible for making decisions. It aggregates network and compute status via C-NMA (CATS network metric agent) and C-SMA (CATS service metric agent), then runs the proposed CANEG algorithm to jointly determine the best routing path and compute site.

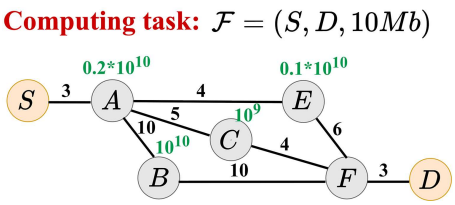


Figure 3 (Color online) An example of a CAN topology with one computing task.

(3) **Upstream transmission.** The task is delivered along the selected communication path to the chosen compute site, where in-network processing is performed (e.g., projection, extraction, inference).

(4) **Processing and downstream delivery.** After computation, the processed data is sent to the destination via the selected downstream path, completing the DCD cycle.

(5) **Response reception.** The final result is returned to the user, ensuring a seamless, low-latency experience.

3.2 System model

Building upon the CAN framework described above, we now define the network elements and computing tasks involved. We represent a computing task by $\mathcal{F}$. We further define $\mathcal{F}$ as a tuple (S, D, μ, γ), where S represents its source node, D represents the destination node and μ represents the initial amount of data associated with the task, measured in Mb. Note that the source node S and destination node D can be the same node since the processed data may be sent back to the same terminal. The initial data will be changed after being processed at the selected compute node. Therefore, we define this change as a scaling factor, denoted as γ, which represents the ratio of the output data amount to the input data amount at the selected compute node. γ is task-specific, determined by the processing operation such as compression, feature extraction. Here we omit the definition of $\mathcal{F}$'s guaranteed E2E delay since our goal is to find the minimum delay that $\mathcal{F}$ can experience in the CAN.

It is important to clarify that $\mathcal{F}$ denotes an atomic computing task in the CAN. Large real-world jobs can always be decomposed into multiple such atomic tasks, each carrying a non-divisible data unit and a service requirement. Therefore, in this work, we focus on optimizing the routing and computing decision for each atomic task individually, while the coordination across multiple tasks is left for future work.

We consider a general CAN comprised of numerous routers and compute nodes. The computing capacity can vary across those compute nodes and can be 0 for the other nodes incapable of processing the task related data. We formally denote the CAN by $G = (V, E)$. Specifically, we define $V = V_t \cup V_c$ as the set of nodes in CAN. We collect the nodes that can process data into the set V_c and collect the other nodes into the set V_t. $E = \{(u, v)|u, v \in V\}$ is the set of transmission links with (u, v) representing the communication link between u and v. Specifically, for each edge $(u, v) \in E$, let $c_{u,v}$ represent its transmission capacity, i.e., the currently available bandwidth on the link (u, v), measured in Gigabits per second (Gbps). Moreover, for each node $w \in V_c$, let c_w denote the residual computing capacity of w, representing the currently available CPU frequency for task $\mathcal{F}$, measured in CPU cycles per second.

Figure 3 shows an example of a CAN with one atomic computing task. Specifically, the orange nodes (i.e., S, D) represent the source node and destination node of the task. Notably, both the source node and destination node can also be computing nodes. The number labeled on the links denotes their transmission capacity. Nodes (i.e., A, B, C, E) with green values labeled are the compute nodes and the green values denote their computing capacity. Nodes (i.e., S, D, F) without green values labeled are regular nodes and cannot process task data.

3.3 Problem formulation

3.3.1 *Variables definition*

Given a CAN $G = (V, E)$ and a computing task $\mathcal{F} = (S, D, \mu, \gamma)$, recall that the CAN first delivers task data from source node S to a compute node w, and then transmits the processed data to the destination node D. Therefore, the overall path for task $\mathcal{F}$ can be decomposed into two sub-paths: (1) from the source node S to a compute node w; (2) from the compute node w to the destination node D. In this case, minimizing the E2E latency is equivalent to selecting an appropriate compute node and corresponding sub-paths such that the sum of the following three components is minimized:

- The communication delay from S to the compute node w;
- The computing delay at the compute node w;
- The communication delay from the compute node w to the destination node D.

To facilitate the formulation of the co-optimization problem, we begin by defining binary variables that characterize the allocation of computing and communication resources.

• y_w. Indicates whether node w is selected to process task $\mathcal{F}$ ($y_w = 1$) or not ($y_w = 0$). For completeness, we also define $y_u = 0$ for $u \in V_t$.

• $x_{u,v}$. Indicates whether link $(u, v) \in E$ is selected to forward task $\mathcal{F}$ ($x_{u,v} = 1$) from source S to compute node w or not ($x_{u,v} = 0$).

• $z_{u,v}$. Indicates whether link $(u, v) \in E$ is selected to forward the processed data of task $\mathcal{F}$($z_{u,v} = 1$) from compute node w to destination D or not ($z_{u,v} = 0$).

To ensure a valid E2E transmission path, we enforce path-formation constraints for both the source-to-compute and compute-to-destination sub-paths. For clarity, we use the terms upstream and downstream to denote these two logical segments. Specifically, the upstream path carries the raw task input data from the source node to the selected compute node, while the downstream path carries the processed data from the compute node to the destination node.

For one atomic task, we impose two conditions: the data must be processed at exactly one compute node and delivered along a single transmission path. Otherwise, the task would no longer be atomic, as data splitting across nodes or paths implicitly corresponds to decomposing it into multiple smaller tasks. The global coordination among multiple atomic tasks, including their parallelism, dependency handling, and job-level completion time, is beyond the scope of this work and will be addressed in our future research.

3.3.2 *Source-to-compute node path constraints*

(a) Upstream E2E transmission constraints. This set of constraints ensures that task $\mathcal{F}$ starts at its designated source node S and reaches exactly one compute node for processing.

Eq. (1) is designed to account for the special case where the source node S itself may serve as the compute node. Specifically, if the source node S is selected as the compute node (i.e., $y_S = 1$), then the computation is performed locally. In this case, there is no need to transmit data from S to any other node, and thus, no outgoing link is selected from S. This leads to (1) being reduced to $\sum_{u:(S,u)\in E} x_{S,u} = 0$, which is consistent with $1 - y_S = 0$. On the other hand, if S is not selected to perform the computation (i.e., $y_S = 0$), the task must be offloaded to a compute node in the network. To ensure this, exactly one outgoing link from S must be selected, resulting in $\sum_{u:(S,u)\in E} x_{S,u} = 1$, which corresponds to $1 - y_S = 1$.

$$\sum_{u:(S,u)\in E} x_{S,u} = 1 - y_S. \tag{1}$$

Eq. (2) captures the flow conservation condition at each node w except S, governed by whether it is selected as the compute node. If w is selected to process task $\mathcal{F}$ (i.e., $y_w = 1$), it becomes the terminal of the upstream path, and no further forwarding occurs; thus, the total incoming flow exceeds the outgoing flow by one. If w is not selected (i.e., $y_w = 0$), it acts as a transit node and must satisfy standard flow conservation, where incoming and outgoing flows are equal. Both scenarios are unified by expressing the net inflow (i.e., total incoming links minus outgoing links) as equal to y_w.

$$\sum_{u:(u,w)\in E} x_{u,w} - \sum_{u:(w,u)\in E} x_{w,u} = y_w, \quad w \in V\backslash\{S\}. \tag{2}$$

(b) Upstream delay constraints. The link delay described here, denoted as $d_{u,v}$, comprises three main components: transmission delay, propagation delay, and queuing delay. Among these, the processing delay, incurred by lightweight operations such as packet inspection, is usually very small and is ignored in this paper.

The transmission delay, denoted as $t_{u,v}$, is determined by the task data and the available bandwidth on the link. Since the task data remains unprocessed before reaching the compute node, its volume stays at μ. Therefore, the transmission delay is given by

$$t_{u,v} = \frac{\mu}{c_{u,v}}, \quad (u, v) \in E. \tag{3}$$

The propagation delay, denoted as $\xi_{u,v}$, is determined by the distance $\delta_{u,v}$ between nodes u and v, and the speed of light s. Thus, it can be computed as follows:

$$\xi_{u,v} = \frac{\delta_{u,v}}{s}, \quad (u, v) \in E. \tag{4}$$

The queuing delay, denoted as $\rho_{u,v}$, depends on the amount of queued data and the transmission rate. However, accurately estimating queuing delay is challenging due to its dynamic nature. To address this, we consider the worst-case queuing delay as an upper bound, ensuring that the actual performance can only be better. Thus, the queuing delay can be computed as follows:

$$\rho_{u,v} = \frac{q_u}{c_{u,v}}, \quad (u,v) \in E, \tag{5}$$

where q_u is the queue length of node u.

Therefore, the link delay can be computed as follows:

$$d_{u,v} = t_{u,v} + \xi_{u,v} + \rho_{u,v}, \quad (u,v) \in E. \tag{6}$$

3.3.3 *Compute-to-destination node path constraints*

(a) Downstream E2E transmission constraints. After computation, the processed data must be forwarded from the compute node w to the final destination D.

Eq. (7) is designed to account for the special case where destination node D itself may serve as the compute node. Specifically, if the destination node D is selected as the compute node (i.e., $y_D = 1$), then the computation is also performed locally. In this case, there is no need to find the compute-to-destination sub-path, and thus, no incoming link is selected to D. Conversely, if node D is not selected as the compute node (i.e., $y_D = 0$), the data must be processed at another compute node and subsequently transmitted to node D. Therefore, exactly one incoming link to D must be activated, and the constraint becomes $\sum_{u:(u,D)\in E} z_{u,D} = 1$, aligning with $1 - y_D = 1$.

$$\sum_{u:(u,D)\in E} z_{u,D} = 1 - y_D. \tag{7}$$

Eq. (8) enforces the correct flow behavior at each node w within the compute-to-destination sub-path. If a node w is selected to process task $\mathcal{F}$ (i.e., $y_w = 1$), it becomes the source node of the downstream path. In this case, the processed data departs from w, and no data arrives from the upstream, resulting in a net outflow of one, i.e., the number of outgoing links minus incoming links equals 1. Conversely, if w is not selected (i.e., $y_w = 0$), it functions as a regular intermediate node, and must satisfy flow conservation, i.e., its incoming and outgoing flows must balance. These two cases are unified in (7) by setting the net outflow equal to y_w.

$$\sum_{u:(w,u)\in E} z_{w,u} - \sum_{u:(u,w)\in E} z_{u,w} = y_w, \quad w \in V \backslash \{D\}. \tag{8}$$

(b) Downstream delay constraints. After processing, the task data volume changes to $\gamma\mu$. Consequently, the transmission delay needs to be recalculated as follows:

$$\tilde{t}_{u,v} = \frac{\gamma\mu}{c_{u,v}}, \quad (u,v) \in E, \tag{9}$$

where $\tilde{t}_{u,v}$ denotes the transmission delay on link (u,v) within the compute-to-destination sub-path.

Therefore, the delay experienced on the links forming the compute-to-destination sub-path for task $\mathcal{F}$ can be recomputed as follows:

$$\tilde{d}_{u,v} = \tilde{t}_{u,v} + \xi_{u,v} + \rho_{u,v}, \quad (u,v) \in E. \tag{10}$$

3.3.4 *Compute node constraints*

(a) One compute node selected constraint. Constraint (11) ensures that exactly one compute node is selected for the computing task.

$$\sum_{w\in V_c} y_w = 1. \tag{11}$$

(b) Computing delay. We define computing delay as the ratio of the CPU cycles required to process the task data to the residual CPU frequency of the compute node, specifically:

$$d_w = \frac{\omega \cdot \mu}{c_w}, \quad w \in V_c, \tag{12}$$

where ω is the number of CPU cycles required to process one bit of data, measured in cycles/bit.

Now, we can get the E2E latency as follows:

$$\mathcal{L} = \sum_{(u,v)\in E} x_{u,v} \cdot d_{u,v} + \sum_{w\in V_c} y_w \cdot d_w + \sum_{(u,v)\in E} z_{u,v} \cdot \tilde{d}_{u,v}. \tag{13}$$

3.3.5 *Problem formulation*

Finally, the routing problem of $\mathcal{F}$ in the CAN can be formulated as

$$\begin{aligned} &\mathbb{P}: \min\ \mathcal{L} \\ &\text{s.t. } (1)\text{–}(13), \\ &\text{var. } x_{u,v},\ y_w,\ z_{u,v} \in \{0,1\}. \end{aligned} \tag{14}$$

Since all constraints are linear and the variables consist solely of binary integer values, $\mathbb{P}$ is classified as an ILP problem. Although $\mathbb{P}$ can be solved optimally, the computation time is prohibitively long. For latency-sensitive applications, the strict E2E latency requirements impose additional constraints, as prolonged delays in determining optimal routing paths can significantly degrade user experience. Furthermore, considering the highly dynamic nature of computing and network resources in CANs, a faster solution approach is required to cope with the dynamic network conditions while still guaranteeing optimality.

4 CAN expanded graph model

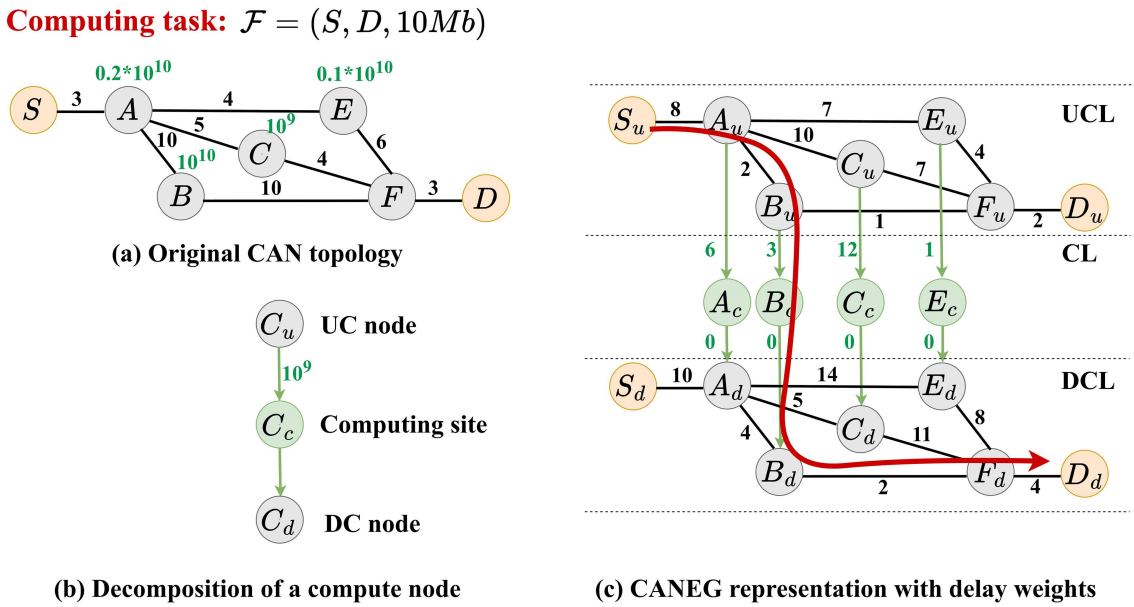
In this section, we introduce the CANEG model, which provides a complete and lossless representation of the original CAN with respect to the DCD process. Building on this model, the routing problem can be transformed from the original ILP formulation into a shortest-path (SP) problem, enabling the use of classical SP algorithms for efficient routing in CAN.

4.1 Construction of CANEG

The CANEG is structured into three layers: the upstream communication layer (UCL), the computing layer (CL), and the downstream communication layer (DCL), each representing a distinct phase in the task's processing pipeline. Figure 4(c) provides a concrete example of how the original CAN topology in Figure 3 is modeled into the CANEG structure. We formally define CANEG as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ represents the set of vertices and $\mathcal{E}$ represents the set of edges. Below, we summarize how to generate the CANEG.

Computing element graph modeling. As shown in Figure 4(b), we decompose each node with compute capability in CAN (i.e., C) into three interconnected components: the upstream communication node (i.e., C_u), the computing site (i.e., C_c), and the downstream communication node (i.e., C_d). This separation explicitly distinguishes their networking and computing functions. Since data follows a strict sequence-first transmitted via upstream links to the computing site, then processed at the computing site, and finally forwarded to the destination node by downstream links—we use directed edges between these three decomposed nodes to maintain the logical flow of this process. For nodes without compute capability, they either serve as upstream communication nodes, forwarding unprocessed data to compute nodes, or as downstream communication nodes, forwarding processed data to the destination node. Therefore, they are only decomposed into two communication nodes (i.e., F_u, F_d in Figure 4(c)). Then, we collect all upstream communication nodes into set $\mathcal{V}_u$, all computing sites into set $\mathcal{V}_c$, and all downstream communication nodes into set $\mathcal{V}_d$. Naturally, we have $\mathcal{V} = \mathcal{V}_u \cup \mathcal{V}_c \cup \mathcal{V}_d$. We also use a set $\mathcal{E}_c$ to collect all the directed edges, termed computing links.

CANEG modeling. We construct the upstream communication layer by connecting the nodes in $\mathcal{V}_u$ according to the topology of the original graph G and similarly form the downstream communication layer by connecting the nodes in $\mathcal{V}_d$ in the same manner. Specifically, for each link $(p, q) \in E$ of G, we create an edge (p_u, q_u) for UCL and create an edge (p_d, q_d) for DCL. Formally, we collect all edges in UCL into set $\mathcal{E}_u$ and all edges in DCL into set $\mathcal{E}_d$. The computing layer, consisting of nodes in $\mathcal{V}_c$, comprises mutually independent compute nodes that are structurally isolated, i.e., no edges exist between them within the layer. Naturally, $\mathcal{E} = \mathcal{E}_u \cup \mathcal{E}_c \cup \mathcal{E}_d$. The source node S in G is mapped to S_u in CANEG, representing the ingress point of the computing task data, while the destination node D is mapped to D_d, ensuring that the processed data is correctly delivered to its final destination. Directed edges from the set $\mathcal{E}_c$ connect these layers, enforcing that data must traverse a computing site from the source node to the destination node while ensuring that only one compute node is selected for processing.



(a) Original CAN topology

(b) Decomposition of a compute node

(c) CANEG representation with delay weights

Figure 4 (Color online) An example of CAN to CANEG modeling.

Label the edges. Since we aim to minimize the E2E latency of the computing task, we label each edge with the potential delay that the task data will experience when passing through it. Specifically, the upstream communication layer corresponds to the source-to-compute node path, so the delay on its edges can be labeled according to (6). Similarly, the downstream communication layer represents the compute-to-destination node path, and its edge delays can be labeled using (10). The computing edges that connect the upstream communication layer to the computing layer represent the processing phase at the corresponding computing site, and their delays should be labeled according to (12). To avoid double-counting the computation delay, the delay for edges transitioning from the computing layer to the downstream communication layer is set to 0.

4.2 Theoretical guarantee of CANEG representation

We now formally prove that the proposed CANEG provides a complete and lossless representation of all feasible DCD paths in the original CAN. This property ensures that solving the shortest path problem in CANEG is equivalent to solving the original ILP problem.

Lemma 1 (Path feasibility)**.** Every feasible path from S_u to D_d in the CANEG must traverse exactly one compute node $w_c \in \mathcal{V}_c$, explicitly corresponding to a valid DCD sequence in the original CAN topology, where the data is transmitted from the source S to a compute node w, processed at w, and subsequently transmitted to the destination D.

Proof. According to the construction of the CANEG, any feasible path from S_u to D_d must traverse through three distinct layers in strict order: first the UCL, then the CL, and finally the DCL. The layers UCL and DCL are structurally isolated from each other and connected only via CL. Thus, every feasible path necessarily passes through exactly one computing vertex w_c, satisfying that exactly one compute node is involved. Consequently, this guarantees a valid mapping of the DCD process from the original CAN to the CANEG.

Theorem 1 (Completeness of CANEG)**.** The CANEG representation is complete: for any feasible DCD path in the original CAN, there exists exactly one corresponding path from S_u to D_d in the CANEG, and vice versa.

Proof. We prove the two directions separately.

Necessity. Assume, for contradiction, that there exists a path $\mathcal{P}$ in CANEG from S_u to D_d that does not correspond to any valid DCD path in the original CAN. By Lemma 1, path $\mathcal{P}$ necessarily includes exactly one compute node w_c. Given the explicit mapping rules between CANEG and CAN, all edges and nodes in CANEG directly correspond to those in CAN, implying a valid DCD path. This directly contradicts our assumption that path $\mathcal{P}$ is invalid. Therefore, every feasible path in CANEG corresponds to a valid DCD path in CAN.

Sufficiency. For any feasible DCD path in CAN, its segments (source-to-compute and compute-to-destination) map directly and uniquely to the corresponding vertices and edges within CANEG by the defined construction. The

source S, compute node w, and destination D are explicitly represented as vertices S_u, w_c, and D_d, respectively. Thus, any feasible path in CAN uniquely corresponds to a valid path in CANEG.

Therefore, the CANEG representation is complete and lossless.

Remark 1. In CANEG, data traversing each path from a vertex in UCL to a vertex in DCL will satisfy the compute requirements as indicated by constraints (1), (2), (7), (8), and (11) since such a path will include exactly one node in $\mathcal{V}_c$ of CL. As a result, the joint compute node selection and routing problem is effectively transformed into a shortest path selection problem.

For completeness, we summarize such a CANEG-based graph scheme in Algorithm 1. Particularly, all edge weights of CANEG are non-negative, satisfying the conditions to use Dijkstra's algorithm to solve the shortest path problem. As such, we can input the generated CANEG and source-destination pair (S_u, D_d) into the Dijkstra algorithm, to obtain the path $\mathcal{P}$ and the E2E latency $\mathcal{L}$. Notably, the identified data delivery path for task data can be obtained by removing the subscript of vertices along the identified $\mathcal{P}$. The selected compute node can be identified by finding the vertex in CL that the $\mathcal{P}$ traverses and removing its subscript. Since Dijkstra's algorithm is already proven to be optimal and the CANEG is a lossless representation of CAN, the solution produced is necessarily optimal.

Algorithm 1 CANEG algorithm.

input: $G = (V, E)$, $\mathcal{F} = (S, D, \mu, \gamma)$;
output: The optimal path $\mathcal{P}$, minimum E2E latency $\mathcal{L}$;
Initialize $\mathcal{V}, \mathcal{E}, \mathcal{V}_u, \mathcal{E}_u, \mathcal{V}_d, \mathcal{E}_d, \mathcal{V}_c, \mathcal{E}_c$ as empty set;
while node $v \in V$ **do**
  Create v_u in $\mathcal{V}_u$ and v_d in $\mathcal{V}_d$;
end while
while edge (p, q) in E **do**
  Add (p_u, q_u) to $\mathcal{E}_u$ and (p_d, q_d) to $\mathcal{E}_d$;
  Update (p_u, q_u)'s weight according to (6);
  Update (p_d, q_d)'s weight according to (10);
end while
while each compute node w in V_c **do**
  Create w_c in $\mathcal{V}_c$, add (w_u, w_c) and (w_c, w_d) to $\mathcal{E}_c$;
  Update (w_u, w_c)'s weight according to (12);
  Update (w_c, w_d)'s weight to 0;
end while
Set $\mathcal{V} = \mathcal{V}_u \cup \mathcal{V}_c \cup \mathcal{V}_d$ and set $\mathcal{E} = \mathcal{E}_u \cup \mathcal{E}_c \cup \mathcal{E}_d$;
Obtain $\mathcal{G} = (\mathcal{V}, \mathcal{E})$;
$[\mathcal{P}, \mathcal{L}]$ = Dijkstra($\mathcal{G}$, S_u, D_d);
return $\mathcal{P}$, $\mathcal{L}$.

4.3 Algorithm complexity analysis

We analyze the computational complexity of the considered algorithms. Let $|E|$ denote the number of links and $|V|$ denote the number of nodes in a CAN. In the worst case, all nodes are computing nodes and the number of compute nodes is $|V|$.

ILP. For the formulated problem $\mathbb{P}$, the number of binary variables is $|V| + 2|E|$, where $|V|$ variables correspond to compute node selection and $2|E|$ variables correspond to upstream and downstream path selection. Therefore, the time complexity of ILP is $O(2^{|V|+2|E|})$.

KSP. The KSP baseline identifies the top-K shortest communication paths between the source and destination using Yen's algorithm. The complexity of finding the K shortest paths is $O(K(|E| + |V| \log |V|))$.

FCF. The fastest-computing-first (FCF) heuristic first selects the compute node with the smallest compute delay among all candidate compute nodes, which requires $O(|V|)$ operations. Then, the shortest communication paths from the source node to the selected compute node and from the compute node to the destination node are determined using Dijkstra's algorithm. Therefore, the overall complexity of FCF is $O(|E| + |V| \log |V|)$.

Proposed algorithm. The proposed method transforms the joint optimization problem into a shortest path problem on an expanded graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. The expanded graph is constructed by duplicating nodes and adding auxiliary edges according to the original topology. Since $|\mathcal{V}| = 3|V|$ and $|\mathcal{E}| = 2(|E| + |V|)$, the graph construction incurs a linear overhead of $O(|E| + |V|)$. For each new task, only the communication and compute weights on the corresponding edges need to be updated according to the current resource status. This process requires scanning the links and compute nodes once, resulting in a complexity of $O(|E| + |V|)$. After constructing the expanded graph, Dijkstra's algorithm is applied to find the shortest delay path. Therefore, the overall time complexity of Algorithm 1 is $O(|\mathcal{E}| + |\mathcal{V}| \log |\mathcal{V}|)$, which is $O(|E| + |V| \log |V|)$ under the assumption that $|E| \gg |V|$.

5 Simulation

5.1 Experimental setup

We conduct experiments on both realistic topologies for real-world applicability and random topologies for scalability. The evaluated topologies are summarized as follows.

• Fat-Tree ($k = 32$): 1280 nodes and 19968 links, introducing abundant equal-cost paths that require computing-aware routing. Each edge switch connects to $k/2$ servers. We collect all servers into compute node set and randomly select nodes in the set as the source and destination nodes of tasks.

• Google B4: 32 datacenter nodes and 610 edges, modeled with intra-region full-mesh and multi-path inter-region parallel links (100/400 Gbps). All nodes are compute nodes.

• GEANT-2012: 40 nodes and 118 edges, derived from the Internet Topology Zoo and updated with real GEANT link capacities from the latest public documentation. We randomly select the nodes as compute nodes, accounting for 40% of the total nodes.

• Random topologies: we use NetworkX to generate the random topologies. The random network topologies are generated as undirected graphs, where the number of nodes ranges from 1000 to 10000 and the number of edges from 50000 to 500000. Nodes are randomly designated as compute nodes according to a configurable ratio. Each compute node is assigned a computing capacity between 10^8 to 10^{10} cycles/s [33]. The bandwidth is uniformly sampled from 1 to 10 Gbps [34]. The queue length q_u is fixed at 10 Mb, resulting in queuing delays that typically range from 0.1 to 10 ms. The propagation delay ranges from 0.1 to 1 ms, implying the distance between two connected nodes ranges from 20 to 200 km. The data size of a computing task is 10 Mb [13]. Additionally, the scale factor γ is set to 5, and the compute resource factor ω is set to 10 [33].

We evaluate the algorithms using the following performance metrics.

• **E2E latency.** Total latency of the selected routing path, including both communication and computing delays.

• **Running time.** Time required by the algorithm to obtain a routing solution.

• **Peak memory usage.** Maximum resident set size (RSS) observed during execution. We monitor the process memory consumption (via `psutil.memory_info().rss`) from graph loading to termination and record the highest RSS as the peak memory usage.

• **Success rate.** Percentage of runs in which the algorithm attains the same optimal E2E latency as the ILP baseline. A run is deemed successful if its objective value matches the ILP optimum.

• **Energy consumption.** Total energy on the selected path, computed as the sum of (i) transmission energy on traversed links and (ii) computation energy at the selected compute node.

We consider three baseline schemes in the evaluation, including an optimal ILP-based method and two heuristic strategies corresponding to the intuitive approaches illustrated in Figure 1, namely the routing-first strategy and the computing-first strategy.

• ILP. The first one is our formulated problem $\mathbb{P}$, called ILP and solved by Gurobi Optimizer 12.0.3 (academic license).

• KSP. This method corresponds to the routing-first strategy illustrated in Figure 1 and was adopted in [13]. Specifically, for each source-destination pair, the algorithm first updates link weights according to (6). Then, the algorithm identifies the top-K loop-free shortest communication delay paths between the source and destination using Yen's algorithm. For each path, if the path contains no compute node, it is considered infeasible and excluded from further evaluation. If the path contains multiple compute nodes, each candidate compute node along the path is evaluated in sequence. For each candidate, the corresponding E2E latency is calculated based on (13). The compute node that yields the minimum E2E latency on the current path is selected. Finally, among all feasible paths, the one with the lowest overall E2E latency is chosen as the final routing path.

• FCF [1]. This heuristic corresponds to the computing-first strategy illustrated in Figure 1. The algorithm first selects the compute node with the smallest compute delay among all candidate compute nodes. Then, the shortest communication paths from the source node to the selected compute node and from the compute node to the destination node are determined using Dijkstra's algorithm. The sum of communication delay and compute delay is used as the final E2E latency.

We conduct all simulations on an Apple M3 processor (8 cores), 8 GB RAM, MacOS (Darwin Kernel 25.0.0, ARM64 architecture). For each experiment, we conduct 500 trials to calculate the average value.

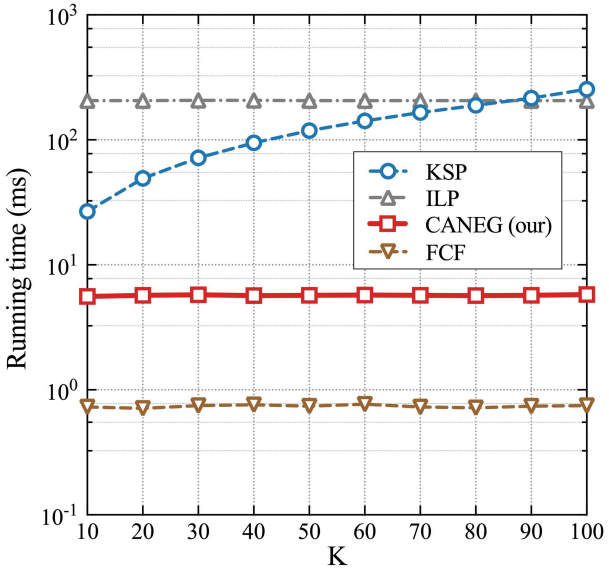


Figure 5 (Color online) Running time vs. varying K.

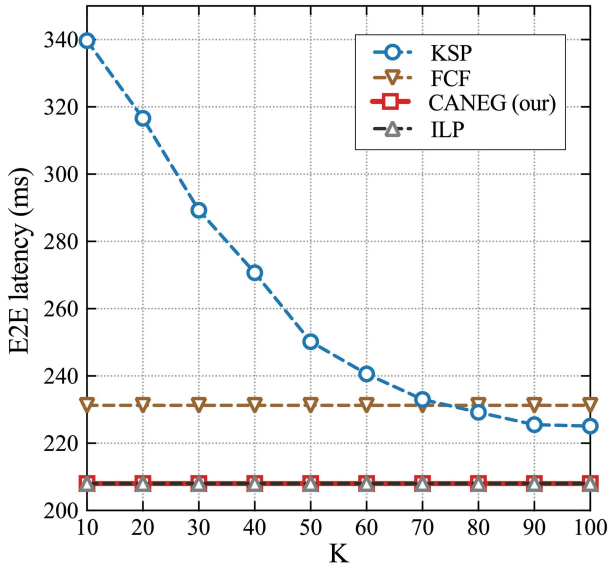


Figure 6 (Color online) E2E latency vs. varying K.

5.2 Results and analysis

5.2.1 *Impact of parameter K in KSP algorithm*

To determine a suitable value of K for the KSP-based baseline, we first conduct experiments by gradually increasing K within a fixed CAN environment, and observe its impact on both the E2E latency and running time performance. Specifically, the experiments are performed on a CAN with 1000 nodes, 10000 links, and a compute node ratio of 5%. The results are presented in Figures 5 and 6.

As shown in Figure 5, the running time of the KSP algorithm increases linearly with the value of K, since a larger K requires enumerating more candidate paths. In contrast, the running times of CANEG and ILP remain nearly constant because the network size is fixed and both methods directly solve the routing and computing decisions without expanding the search space with respect to K. The FCF baseline exhibits the shortest running time among all compared methods, as it simply selects the computing node with the smallest computing delay and then determines the routing path using a shortest-path strategy.

Figure 6 illustrates the impact of the parameter K on the E2E latency of the KSP baseline. As K increases, the latency of KSP gradually decreases. However, the improvement becomes progressively smaller as K continues to grow, indicating a clear diminishing return effect. By comparison, CANEG consistently achieves E2E latency identical to the optimal ILP solution across all tested values of K. The FCF baseline, despite its very low running time, results in significantly higher E2E latency because it adopts a computing-first heuristic without jointly optimizing routing and computation. When $K = 10$, KSP achieves a running time comparable to our CANEG-based approach but suffers from significantly higher latency, with an E2E delay approximately 63.3% larger than that of CANEG. Increasing K can reduce this performance gap; however, the running time increases substantially, ranging from 2.19 to 7.75 times that of CANEG. Considering this trade-off between latency performance and computational cost, we select $K = 10$ and $K = 40$ as representative configurations in the subsequent experiments. The former highlights the efficiency advantage of the KSP heuristic with a limited search space, while the latter represents a practical compromise where latency performance is significantly improved without incurring excessive computational overhead.

5.2.2 *Impact of computing node ratio*

Figures 7 and 8 illustrate the performance of the evaluated schemes under varying compute node ratios, defined as the proportion of compute nodes to the total number of nodes in the network. The experiments are conducted on a fixed network with 1000 nodes and 10000 links. Although the number of nodes and links is fixed, we vary the network topology across multiple trials for each compute node ratio by using different random seeds. This ensures statistical diversity in the experimental results. Additionally, when varying the compute node ratio, the selection of compute nodes is incremental, i.e., nodes selected at lower ratios are always included at higher ratios, rather than

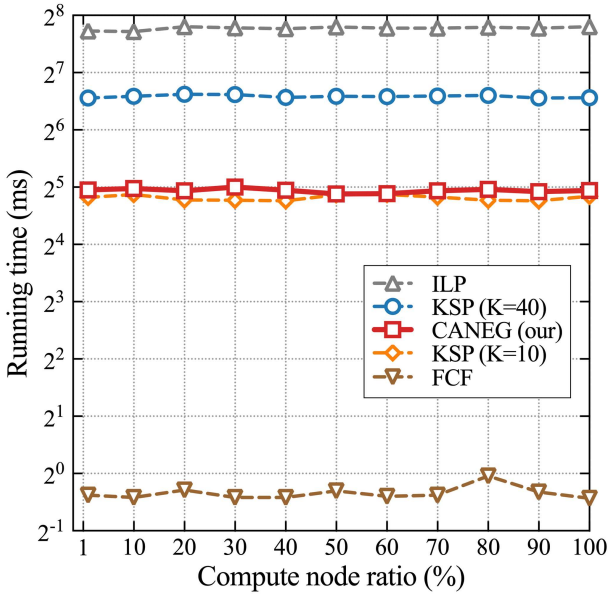


Figure 7 (Color online) Running time vs. varying compute node ratio.

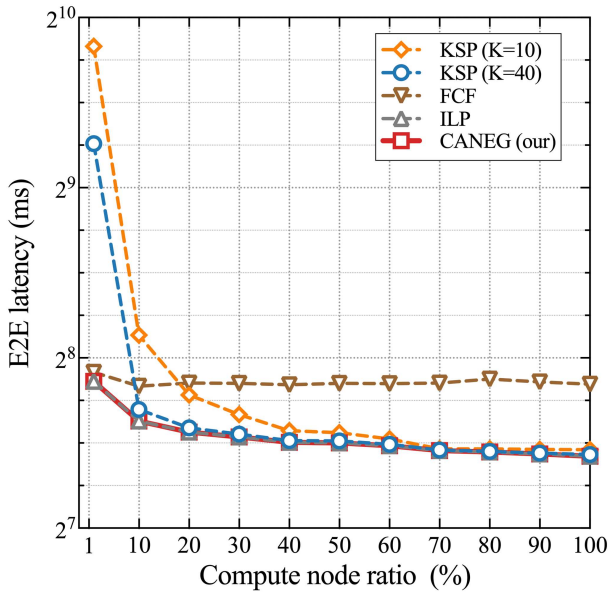


Figure 8 (Color online) E2E latency vs. varying compute node ratio.

being re-sampled.

As shown in Figure 7, the running time of CANEG remains stable across different compute node ratios, ranging from 29.50 to 32.06 ms. In contrast, the running time of the KSP baseline depends on K. When $K = 40$, KSP requires 94.08–98.56 ms, which is about 3× slower than CANEG, while reducing K to 10 lowers the running time to 27.10–29.28 ms, comparable to CANEG. The ILP solver exhibits the highest running time (210.27–223.09 ms) due to its exhaustive search for the optimal solution. In contrast, the FCF baseline has the lowest running time (below 1 ms) since it adopts a simple computing-first heuristic without jointly optimizing routing and computation.

Figure 8 shows the corresponding E2E latency performance. Both CANEG and ILP consistently achieve the lowest latency, confirming the optimality of CANEG. For both $K = 10$ and $K = 40$, the E2E latency of KSP is considerably worse at low compute node ratios. This is because if the optimal compute node resides on a communication path with relatively high delay, the KSP algorithm, which only explores the top-K shortest communication delay paths, may fail to include this path. As the compute node ratio increases, the number of candidate compute nodes along the selected paths also increases, making it more likely for KSP to find a near-optimal solution, thereby reducing the overall latency. In comparison, KSP ($K = 10$) performs the worst, with E2E delays up to 2.91 times higher than CANEG, especially at a low compute node ratio. Increasing K to 40 still results in large performance gaps (up to 163.3% worse than CANEG). The FCF baseline maintains relatively high latency and results in latency up to 34.6% higher than CANEG.

Notably, when the compute node ratio exceeds 40%, both the E2E latency and running time of all schemes become stable, and further increases in compute node density yield negligible improvements. Therefore, we select 40% as the default compute node ratio for the subsequent evaluations, as it provides a good balance between performance and computational resource provisioning.

5.2.3 *Performance comparison across large network conditions*

To evaluate the scalability of the proposed method, we examine both the running time and E2E latency under different network scales, where the number of nodes increases from 1000 to 10000, and the number of edges grows from 50000 to 500000. The results are illustrated in Figures 9 and 10.

As shown in Figure 9, the running time of CANEG increases gradually with the network size while remaining consistently low, demonstrating strong scalability. In contrast, both KSP ($K = 40$) and ILP incur significantly higher computational overhead, with ILP being particularly sensitive to the network scale due to its combinatorial complexity. For example, when the network contains 1000 nodes and 50000 edges, CANEG completes in only 147.64 ms, whereas KSP requires 369.26 ms and ILP takes more than 1126.52 ms, making them approximately 2.5× and 7.6× slower than CANEG, respectively.

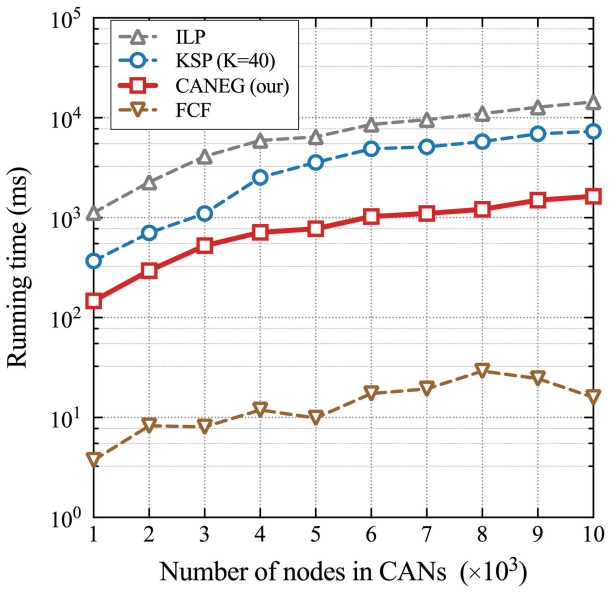


Figure 9 (Color online) Running time vs. network scale.

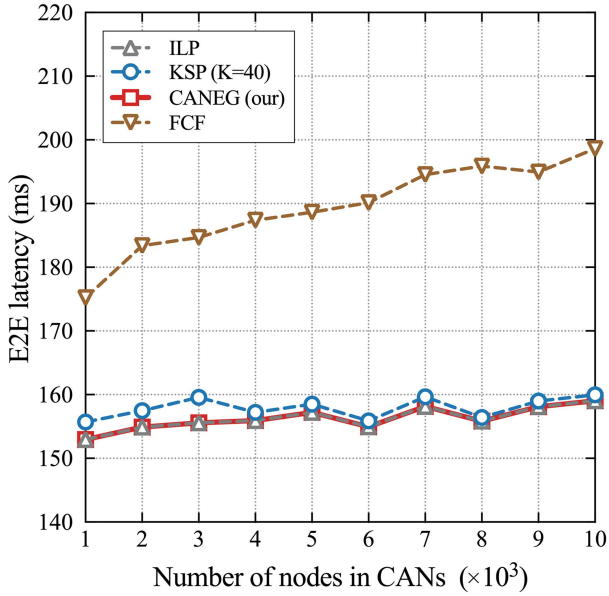


Figure 10 (Color online) E2E latency vs. network scale.

Figure 10 presents the corresponding E2E latency results. It can be observed that CANEG consistently achieves latency identical to that of the optimal ILP solution across all tested scales. The KSP ($K = 40$) baseline produces latency close to the optimal solution, typically within about 1%–3% of CANEG, but at the cost of significantly higher running time. In contrast, the FCF baseline achieves the lowest running time but results in substantially higher latency, around 13%–25% larger than CANEG, since it selects compute nodes based only on computing delay. Overall, these results demonstrate that CANEG achieves an effective balance between computational efficiency and latency optimality, providing optimal E2E performance while maintaining strong scalability in large-scale networks.

To illustrate the real-world applicability of the proposed algorithm, we then simulate the E2E latency, running time, peak memory usage and success rate metrics of the four algorithms (ILP, KSP ($K = 40$), CANEG, FCF) on three realistic network topologies (Fat-Tree, Google B4, GEANT-2012). As shown in Figure 11, CANEG still maintains the minimum running time while guaranteeing the same solution as the ILP's. Besides, CANEG consistently demonstrates comparable peak memory usage compared to ILP, KSP and FCF across all topologies. For example, on the Google B4 topology, the peak memory usage of CANEG is approximately 65.1 MB, while ILP's, KSP's and FCF's are 64.9, 65.1 and 65.1 MB, respectively. On Fat-Tree and GEANT, CANEG maintains a similar memory usage, confirming its applicability. While for a large-scale CAN with 10000 nodes and 500000 edges, the peak memory usage of CANEG is 2264.08 MB, which is much lower than ILP's (3216.87 MB).

As shown in Figure 11, CANEG also achieves a 100% success rate in finding the optimal solution (matching ILP) on all topologies among 500 trials. In contrast, KSP's success rate drops significantly on complex topologies (e.g., 70% on Fat-Tree). The FCF baseline performs even worse in terms of success rate; for example, it achieves only about 1% on the Fat-Tree topology, indicating that it rarely finds the optimal solution. These results demonstrate that CANEG not only matches the optimality of ILP but also offers superior memory efficiency and robustness across diverse, large-scale real-world networks.

Additionally, we further evaluated the energy consumption of the paths obtained by the four algorithms on three realistic topologies. Before presenting these results, we first describe how the energy consumption along a selected path is computed. Assuming identical device models, where power characteristics are consistent across all compute and communication nodes, we treat the power of each compute node and communication link as constant during task execution or transmission.

• KSP. We first update the edge weights in the original CAN as $(\alpha \cdot d_{u,v} + (1-\alpha) \cdot P_{u,v} \cdot t_{u,v})$. Then, we extract the top-K shortest paths on the original CAN topology using the updated weights. We select paths that contain at least one compute node as candidate paths. For each candidate path, all compute nodes located on the path are enumerated. For each such node, the actual E2E cost is recomputed by following the DCD execution order-upstream before the compute node, computation at the node, and downstream afterwards. The compute node that yields the smallest path cost is associated with the path, and the path with minimum cost among all candidate paths is selected as the final KSP solution.

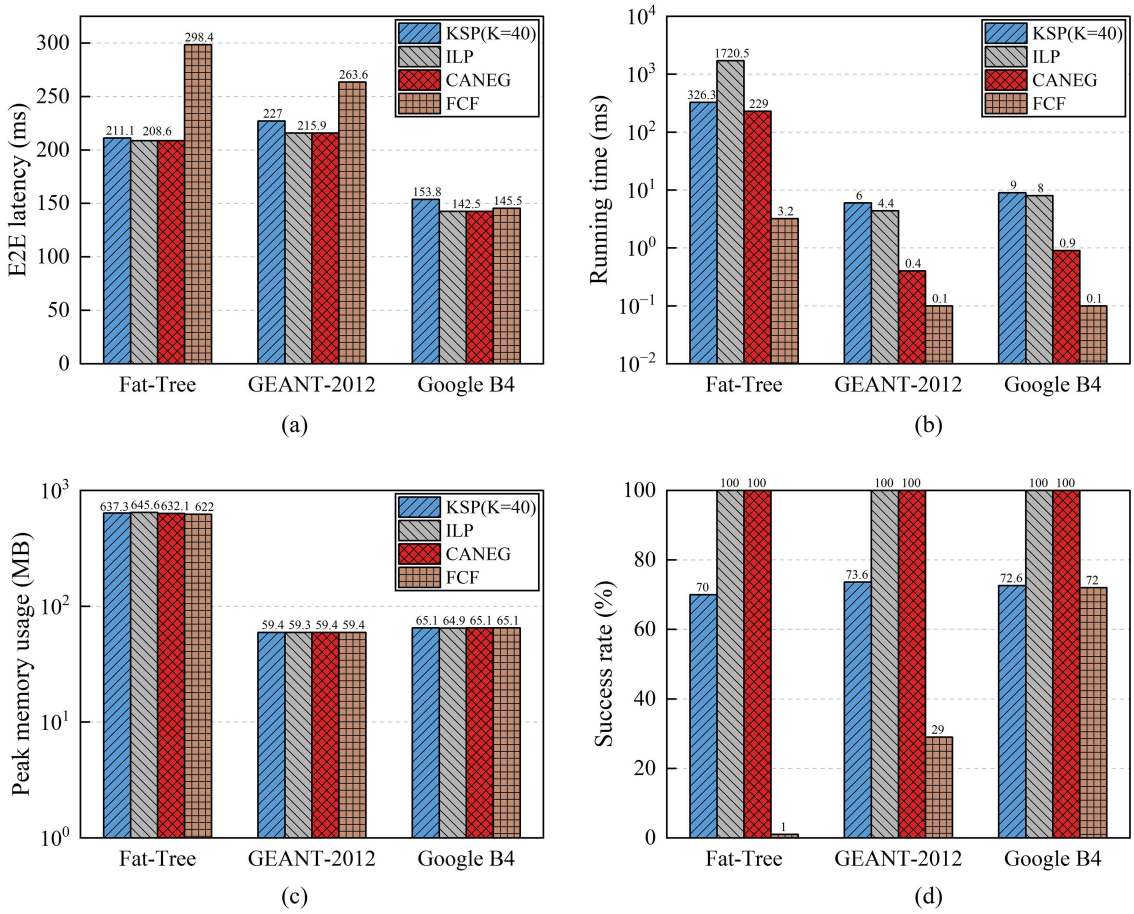


Figure 11 (Color online) Performance comparison of algorithms across three realistic network topologies. (a) E2E latency; (b) running time; (c) peak memory usage; (d) success rate.

• ILP. We reformulated the single objective ILP to minimize a weighted sum of total energy consumption and E2E latency. Specifically, the new ILP problem is modeled as $\mathbb{P}1$.

$$\begin{aligned}
&\mathbb{P}1: \min\ \ \alpha \cdot \mathcal{L} + (1-\alpha) \cdot \Psi \\
&\text{s.t.} \\
&\Psi = \sum_{(u,v)\in E} x_{u,v} \cdot P_{u,v} \cdot t_{u,v} + \sum_{w \in V_c} y_w \cdot P_w \cdot d_w + \sum_{(u,v)\in E} z_{u,v} \cdot P_{u,v} \cdot \tilde{t}_{u,v}, \\
&(1)\text{–}(13), \\
&\text{var. } x_{u,v},\ y_w,\ z_{u,v} \in \{0,1\}.
\end{aligned} \tag{15}$$

Here, $\alpha \in [0,1]$ is a tunable weight parameter that balances the importance of energy consumption and latency. $P_{u,v}$ and P_w are the power consumption of link (u,v) and compute node w, respectively. By adjusting α, we can explore different trade-offs between the two objectives. For example, if we want to find a minimum energy consumption DCD path, we can set α to 0.

• CANEG. As the CANEG already separates upstream, compute, and downstream into three layers, we update the weights of upstream by $\alpha \cdot d_{u,v} + (1-\alpha) \cdot P_{u,v} \cdot t_{u,v}$ and update the weights of downstream by $\alpha \cdot \tilde{d}_{u,v} + (1-\alpha) \cdot P_{u,v} \cdot \tilde{t}_{u,v}$. We update the weight of edges from UCL to CL by $\alpha \cdot d_w + (1-\alpha) \cdot P_w \cdot d_w$. Then we use the Dijkstra algorithm to find the shortest cost path within CANEG.

• FCF. The weights of all compute nodes are first updated as $\alpha d_w + (1-\alpha) P_w d_w$, and the node with the minimum weight is selected as the compute node. The shortest paths from the source to the selected compute node and from the compute node to the destination are then determined by updating the link weights as $\alpha d_{u,v} + (1-\alpha) P_{u,v} t_{u,v}$ and $\alpha \tilde{d}_{u,v} + (1-\alpha) P_{u,v} \tilde{t}_{u,v}$, respectively. The sum of these three costs is taken as the final cost of FCF.

Figure 12 shows the average total energy consumption of ILP, CANEG, KSP ($K = 40$) and FCF over 500 simulation runs on the Fat-Tree, GEANT-2012, and Google B4 topologies. Specifically, ILP and CANEG exhibit

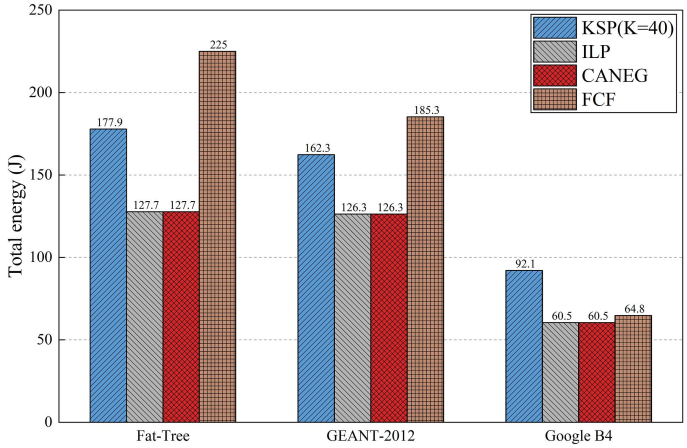


Figure 12 (Color online) Energy consumption of three algorithms within three realistic topology.

identical energy consumption in all cases. In contrast, KSP consumes 40%–50% more energy in Fat-Tree and Google B4, and about 30% more in GEANT due to suboptimal path and computing-node choices. FCF consumes 50%–76.2% more energy in GEANT and Fat-Tree, and about 10% in Google B4. Overall, CANEG matches ILP's optimal energy performance while significantly outperforming KSP and FCF.

6 Conclusion

In this paper, we studied the joint routing and compute-node selection problem in CANs. We first formulated the problem as an ILP problem and then proposed the CANEG model to overcome the computational intractability of solving the ILP directly. By constructing a complete and lossless three-layer expanded graph, CANEG enables using the Dijkstra algorithm to jointly determine routing paths and compute nodes while preserving ILP-level optimality. Numerical experiments demonstrate that CANEG can always obtain the optimal solution while significantly reducing execution time and energy consumption compared with existing baselines. Although CANEG guarantees polynomial-time optimality for each atomic computing task, the global coordination among multiple atomic tasks, including their parallelism, dependency handling, and job-level completion time, is still a great challenge. We will address the above challenge in our future research.

Acknowledgements This work was supported by National Natural Science Foundation of China (Grant No. 61931017).

References

1 IETF Computing-Aware Traffic Steering Working Group. Computing-aware traffic steering (CATS) problem statement, use cases, and requirements. https://datatracker.ietf.org/doc/draft-ietf-cats-usecases-requirements/06/
2 IETF Computing-Aware Traffic Steering Working Group. A framework for computing-aware traffic steering (CATS). https://datatracker.ietf.org/doc/draft-ietf-cats-framework/06/
3 Yukun S, Bo L, Junlin L, et al. Computing power network: a survey. China Commun, 2024, 21: 109–145
4 Gallo G, Pallottino S. Shortest path algorithms. Ann Oper Res, 1988, 13: 1–79
5 Bastug E, Bennis M, Medard M, et al. Toward interconnected virtual reality: opportunities, challenges, and enablers. IEEE Commun Mag, 2017, 55: 110–117
6 Ren Y L, Lyu X C, Tao X F, et al. Robust joint optimization framework for highly reliable low-latency communication services under traffic uncertainties. Sci China Inf Sci, 2026, 69: 112305
7 Ahn J, Kin Y Y, Kim R Y. Delay oriented VR mode WLAN for efficient Wireless multi-user Virtual Reality device. In: Proceedings IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, 2017. 122–123
8 Garg S, Ihler A, Bentley E S, et al. A cross-layer, mobility, and congestion-aware routing protocol for UAV networks. IEEE Trans Aerosp Electron Syst, 2023, 59: 3778–3796
9 Liu K, Quan W, Cheng N, et al. Deadline-constrained multi-agent collaborative transmission for delay-sensitive applications. IEEE Trans Cogn Commun Netw, 2023, 9: 1370–1384
10 Wang P, Li H, Chen B, et al. Enhancing earth observation throughput using inter-satellite communication. IEEE Trans Wireless Commun, 2022, 21: 7990–8006
11 Wang P, Sourav S, Li H Y, et al. One pass is sufficient: a solver for minimizing data delivery time over time-varying networks. In: Proceedings of IEEE Conference on Computer Communications, New York, 2023. 1–10
12 Shi K, Wang J, Li H, et al. Enhancing resource utilization of non-terrestrial networks using temporal graph-based deterministic routing. IEEE Trans Veh Technol, 2024, 73: 9211–9216
13 Li F, Xie R C, Tang Q Q, et al. CaRCS: joint optimization of computing-aware routing and collaborative scheduling in computing power networks. IEEE Netw, 2024, 73: 2739–2752
14 Xie R, Feng L, Tang Q, et al. Delay-prioritized and reliable task scheduling with long-term load balancing in computing power networks. IEEE Trans Serv Comput, 2024, 17: 3359–3372
15 Cao J, Zhang S, Chen Q, et al. Computing-aware routing for LEO satellite networks: a transmission and computation integration approach. IEEE Trans Veh Technol, 2023, 72: 16607–16623

16 Plachy J, Becvar Z, Strinati E C, et al. Dynamic allocation of computing and communication resources in multi-access edge computing for mobile users. IEEE Trans Netw Serv Manage, 2021, 18: 2089–2106
17 Zhao Y H, Chong Z, Han X Y, et al. Simulation study of routing mechanism in the computing-aware network. In: Proceedings of International Conference on Network, Communication and Computing, New York, 2021. 126–134
18 Liu G Y, Xi R Y, Jiang T, et al. Feasibility study of cooperative sensing: radar cross section, synchronization, cooperative cluster, performance and prototype. Sci China Inf Sci, 2025, 68: 150302
19 Poularakis K, Llorca J, Tulino A M, et al. Service placement and request routing in MEC networks with storage, computation, and communication constraints. IEEE ACM Trans Netwing, 2020, 28: 1047–1060
20 Li X L, Xie R C, Huang T, et al. Joint resource allocation for software-defined serverless service-centric networking. In: Proceedings of IEEE Global Communications Conference, Rio de Janeiro, 2022. 861–866
21 Huynh D V, Khosravirad S R, Cotton S L, et al. Joint sensing, communications, and computing design for 6G URLLC service-oriented MEC networks. IEEE Int Things J, 2024, 11: 32429–32439
22 Cai Y, Llorca J, Tulino A M, et al. Joint compute-caching-communication control for online data-intensive service delivery. IEEE Trans Mobile Comput, 2024, 23: 4617–4633
23 Liu Q, Zhang H, Zhang X, et al. Joint service caching, communication and computing resource allocation in collaborative MEC systems: a DRL-based two-timescale approach. IEEE Trans Wireless Commun, 2024, 23: 15493–15506
24 Jia M, Zhang L, Wu J, et al. Joint computing and communication resource allocation for edge computing towards Huge LEO networks. China Commun, 2022, 19: 73–84
25 Younis A, Maheshwari S, Pompili D. Energy-latency computation offloading and approximate computing in mobile-edge computing networks. IEEE Trans Netw Serv Manage, 2024, 21: 3401–3415
26 Kim M, Jang J, Choi Y, et al. Distributed task offloading and resource allocation for latency minimization in mobile edge computing networks. IEEE Trans Mobile Comput, 2024, 23: 15149–15166
27 Liu H, Long X, Li Z, et al. Joint optimization of request assignment and computing resource allocation in multi-access edge computing. IEEE Trans Serv Comput, 2022, 16: 1254–1267
28 Li Z, Zhang H, Li X, et al. Distributed task scheduling for MEC-assisted virtual reality: a fully-cooperative multiagent perspective. IEEE Trans Veh Technol, 2024, 73: 10572–10586
29 Liu Y, Zhou J, Tian D, et al. Joint communication and computation resource scheduling of a UAV-assisted mobile Edge computing system for platooning vehicles. IEEE Trans Intell Transp Syst, 2022, 23: 8435–8450
30 Chen L J, Chen Z Y, Xia B, et al. Joint optimization of communications and computing resources allocation for deterministic transmission in wireless edge networks. China Commun, 2022, 19: 1–11
31 Simmons J M. Optical Network Design and Planning. Berlin: Springer, 2014
32 Liu F, Li H Y, Wang P, et al. Graph based Joint Computing and Communication Scheduling for Virtual Reality Applications. In: Proceedings of IEEE Wireless Communications and Networking Conference (WCNC), Glasgow, 2023. 1–6
33 Sun Y, Chen Z, Tao M, et al. Communications, caching, and computing for mobile virtual reality: modeling and tradeoff. IEEE Trans Commun, 2019, 67: 7573–7586
34 Wang X, Ning Z, Guo L, et al. Online learning for distributed computation offloading in wireless powered mobile edge computing networks. IEEE Trans Parallel Distrib Syst, 2021, 33: 1841–1855