Delay-Optimal Service Chain Forwarding and Offloading in Collaborative Edge Computing

Jinkun Zhang and Edmund Yeh
Department of Electrical and Computer Engineering
Northeastern University, US

***Abstract*—Collaborative edge computing (CEC) is an emerging paradigm for heterogeneous devices to collaborate on edge computation jobs. For congestible links and computing units, delay-optimal forwarding and offloading for service chain tasks (e.g., DNN with vertical split) in CEC remains an open problem. In this paper, we formulate the service chain forwarding and offloading problem in CEC with arbitrary topology and heterogeneous transmission/computation capability, and aim to minimize the aggregated network cost. We consider congestion-aware non-linear cost functions that cover various performance metrics and constraints, such as average queueing delay with limited processor capacity. We solve the non-convex optimization problem globally by analyzing the KKT condition and proposing a sufficient condition for optimality. We then propose a distributed algorithm that converges to the global optimum. The algorithm adapts to changes in input rates and network topology, and can be implemented as an online algorithm. Numerical evaluation shows that our method significantly outperforms baselines in multiple network instances, especially in congested scenarios.**

I. Introduction

Recent years have seen an explosion in the number of mobile and IoT devices. Many of the emerging mobile applications, such as VR/AR, autonomous driving, are computation-intensive and time-critical. Directing all computation requests and their data to the central cloud is becoming impractical due to limited backhaul bandwidth and high associated latency. Edge computing has been proposed as a promising solution to provide computation resources and cloud-like services in close proximity to mobile devices. An extension of the idea of edge computing is the concept of collaborative edge computing (CEC). In addition to point-to-point offloading, CEC permits multiple stakeholders (mobile devices, IoT devices, edge servers, and cloud) to collaborate with each other by sharing data, communication resources, and computation resources to accomplish computation tasks [1]. Devices equipped with computation capabilities can collaborate with each other through D2D communication [2]. Edge servers can also collaborate with each other for load balancing or further with the central cloud to offload demands that they cannot accommodate [3]. CEC is also needed if no direct connection exists between devices and edge servers. That is, computation-intensive tasks of unmanned aerial vehicle (UAV) swarms or autonomous cars far from the wireless access point should be collaboratively computed or offloaded through multi-hop routing to the server with the help of other devices [2], [4].

The delay-optimal routing and offloading for independent one-step computation tasks in CEC has been proposed [5]. As a generalization of one-step computation, *service chaining* enables the ordering of computation tasks [6]. In the service chain model, the network provides services called *applications*, where one application consists of a chain of *tasks* that are performed sequentially, mapping input data to output results via (potentially multiple) intermediate stages. Service chaining has been widely studied in the network function virtualization (NFV) context, e.g., [7], [8]. An example of service chaining is shown in Fig 1.

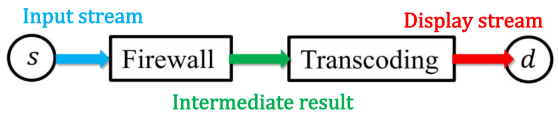

Fig. 1: Service chain example: LAN video stream client for from source s to display d. Input data stream undergoes sequential tasks.

Optimization for service chain computation allocation has been widely considered. Zhang et al. [7] proposed a throughput-optimal control policy for distributed computing networks with service chain applications and mixed-cast traffic flows, but did not consider the optimization of delay performance. Khoramnejad et al. [9] minimized the cost of service chain delay and energy consumption at the edge via deep learning, but within the restricted setting of bipartite network topologies. Sahni et al. [10] optimized for the earliest finish time (EFT) of service chains in mesh networks, albeit without considering link congestion. Recently, an efficient framework for dispersed computing of interdependent tasks was established by Ghosh et al. [11], but without a theoretical performance guarantee.

In CEC with arbitrary topology and heterogeneous nodes, the delay-optimal forwarding and offloading of service chain tasks remains an open problem. One major challenge is incorporating congestion-dependent non-linear costs (e.g., queueing delay at transmission links and computing units). Furthermore, the forwarding and offloading mechanism should preferably be distributed for network scaling.

To meet these challenges, this paper considers an arbitrary multi-hop network, where nodes collaboratively finish multiple service chain applications. Nodes have heterogeneous computation capabilities, while some are also data sources (e.g., sensors and mobile users). For each application, data enter the network at the source nodes. Intermediate results are produced and forwarded at collaborative nodes, and the final result is delivered to the destination node. To more accurately model the delay incurred on links and computing units, we consider general congestion-dependent non-linear costs. We propose a

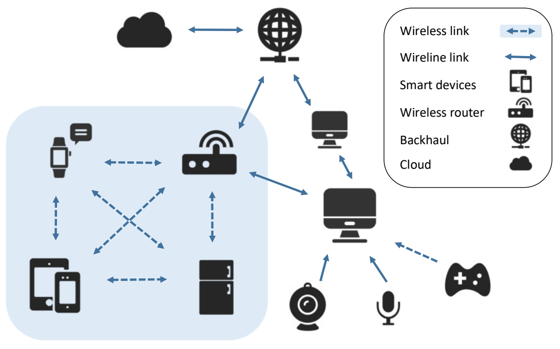

Fig. 2: Sample topology involving IoT network on the edge [5].

framework that unifies the mathematical representation of forwarding and computation offloading, and tackle the non-linear cost minimization problem with a node-based perspective first introduced by [12] and adopted in [5]. We first characterize the Karush–Kuhn–Tucker (KKT) necessary conditions for the proposed optimization problem, and demonstrate that the KKT condition can lead to arbitrarily suboptimal performance in certain degenerate cases. To overcome this issue, we propose a modification to the KKT condition, and prove that the new condition is a sufficient for global optimality. We also devise a distributed and adaptive algorithm that converges to an operating point satisfying the sufficiency condition.

We summarize our detailed contributions as follows:

- We investigate joint forwarding and offloading for service chain applications in arbitrary multi-hop network topologies with non-linear transmission and computation costs and formulate a non-convex cost minimization problem.
- We study the KKT necessary condition for optimality in the non-convex problem, and present a sufficient condition for optimality by modifying the KKT condition.
- We devise a distributed and adaptive algorithm that converges to the global optimum.
- Through numerical evaluation, we demonstrate the advantages of our algorithm relative to baselines in different network instances, especially in congested scenarios.

In Section II we present the network model and the optimization problem. In Section III we present the sufficient optimality conditions. In Section IV we develop a distributed algorithm, and in Section V we present our numerical results.

II. Problem formulation

We model the CEC network with a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ and $\mathcal{E}$ are the sets of nodes and links, respectively. Fig. 2 gives an example CEC network. A node $i \in \mathcal{V}$ can represent a user device, an edge server, or the cloud. A link $(i, j) \in \mathcal{E}$ represents an available (wireline or wireless) channel from i to j. We assume the nodes in $\mathcal{V}$ and links in $\mathcal{E}$ have heterogeneous capabilities for computation and transmission. The computation and transmission in $\mathcal{G}$ are driven by *applications*, where we let $\mathcal{A}$ denote the set of the applications. Each application is a service chain, i.e., an application consists of a finite number of *tasks* performed

$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	Network graph $\mathcal{G}$, set of nodes $\mathcal{V}$ and links $\mathcal{E}$
$\mathcal{A}$; d_a	Set of applications; result destination of application a
$\mathcal{T}_a$	Service chain tasks of application $a \in \mathcal{A}$
$\mathcal{S}$	Set of all stages (a, k) where $a \in \mathcal{A}$, $k = 0, \cdots, \lvert\mathcal{T}_a\rvert$
$L_{(a,k)}$	Packet size of stage $(a, k) \in \mathcal{S}$
$r_i(a)$	Exogenous input data rate for application a at node i
$t_i(a, k)$	Traffic of stage (a, k) at node i
$\phi_{ij}(a, k)$	Fraction of $t_i(a, k)$ forwarded to node j (if $j \neq 0$)
$\phi_{i0}(a, k)$	Fraction of $t_i(a, k)$ assigned to CPU at i
$f_{ij}(a, k)$	Rate (packet/sec) of stage (a, k) on link (i, j)
$g_i(a, k)$	Rate (packet/sec) of stage (a, k) assigned to CPU at i
$D_{ij}(F_{ij})$	Transmission cost (e.g. queueing delay) on (i, j)
$C_i(G_i)$	Computation cost (e.g. CPU runtime) at node i

TABLE I: Major notations

sequentially with a pre-determined order. For $a \in \mathcal{A}$, we let $\mathcal{T}_a$ be the (ordered) set of a's tasks.

We assume that any node in $\mathcal{V}$ can act as a data source or computation site for any application. The data input rate for application a at node i is $r_i(a)$ (packet/sec)[1], where each data packet has size $L_{(a,0)}$ (bit). Each application has one pre-specified destination $d_a \in \mathcal{V}$, to which the final results of the service chain are delivered. For application a, the data flows are injected into the network from nodes i with $r_i(a) > 0$, forwarded in a hop-by-hop manner to nodes that decide to perform the first task of a. After the first task, data flows are converted into flows of intermediate results, and are further forwarded for the next task in the service chain. Eventually, final results are delivered to the destination d_a. We assume the flows of application a are categorized into $|\mathcal{T}_a| + 1$ *stages*, where stage (a, k) with $k = 0, 1, \cdots, |\mathcal{T}_a|$ represents the (intermediate) results that have finished the k-th computation task of a. Particularly, we say the data flows are of stage $(a, 0)$, and the final results are of stage $(a, |\mathcal{T}_a|)$. We assume packets of stage (a, k) are of size $L_{(a,k)}$ (bit), and denote the set of all stages by $\mathcal{S} = \big\{(a, k)\big|a \in \mathcal{A}, k = 0, 1 \cdots, |\mathcal{T}_a|\big\}$.

To model the forwarding and computation offloading behavior in $\mathcal{G}$, we adopt a node-based perspective first introduced in [12] and followed by [5]. We let $t_i(a, k)$ denote the *traffic* of stage $(a, k) \in \mathcal{S}$ at node i. Specifically, $t_i(a, k)$ includes both the packet rate of stage (a, k) that is generated at i (this can be the input data rate $r_i(a)$ if $k = 0$, or the newly generated intermediate results at i's computation unit if $k \neq 0$), and the packet rate that is forwarded from other nodes to i. We let $\phi_{ij}(a, k) \in [0, 1]$ be the *forwarding/offloading* variable for $i, j \in \mathcal{V}$ and $(a, k) \in \mathcal{S}$. Specifically, of the traffic $t_i(a, k)$ that arrive at i, node i forwards a fraction of $\phi_{ij}(a, k)$ to node k. Moreover, if $k \neq |\mathcal{T}_a|$, node i forwards a fraction $\phi_{i0}(a, k)$ of $t_i(a, k)$ to its local CPU to perform computation of the $k+1$-th task.[2] We assume for every data or intermediate packet consumed at CPU for computation, one and only one next-stage packet is generated accordingly. Therefore, for $k = 0$,

$$t_i(a, 0) = \sum\nolimits_{j \in \mathcal{V}} t_j(a, 0)\phi_{ji}(a, 0) + r_i(a),$$

[1]We allow applications to have multiple data sources.

[2]For coherence, we let $\phi_{ij}(a, k) \equiv 0$ if $(i, j) \notin \mathcal{E}$, and $\phi_{i0}(a, |\mathcal{T}_a|) \equiv 0$. Such fractional forwarding can be achieved via various methods, e.g., random packet dispatching with probability $\phi_{ij}(a, k)$. More complex mechanisms also exist to stabilize the actual flow rates, e.g., [13].

and for $k \neq 0$,

$$t_i(a,k) = \sum\nolimits_{j\in\mathcal{V}} t_j(a,k)\phi_{ji}(a,k) + t_i(a,k-1)\phi_{i0}(a,k-1).$$

Let $\boldsymbol{\phi} = [\phi_{ij}(a,k)]_{(a,k)\in\mathcal{S},i\in\mathcal{V},j\in\{0\}\cup\mathcal{V}}$ be the *global forwarding/offloading strategy*, with the following constraint

$$\sum_{j\in\{0\}\cup\mathcal{V}} \phi_{ij}(a,k) = \begin{cases} 0, & \text{if } k = |\mathcal{T}_a| \text{ and } i = d_a, \\ 1, & \text{otherwise.} \end{cases} \quad (1)$$

Constraint (1) guarantees that all input data will eventually be processed to the final result, and exit the network at the destination. Let $f_{ij}(a,k)$ be the packet rate (packet/sec) for stage (a,k) packets transmitted on link (i,j), and let $g_i(a,k)$ be the packet rate (packet/sec) for stage (a,k) packets forwarded to i's CPU for computation. Then,

$$f_{ij}(a,k) = t_i(a,k)\phi_{ij}(a,k), \quad g_i(a,k) = t_i(a,k)\phi_{i0}(a,k).$$

The total flow rate (bit/sec) on link $(i,j) \in \mathcal{E}$ is given by

$$F_{ij} = \sum\nolimits_{(a,k)\in\mathcal{S}} L_{(a,k)} f_{ij}(a,k),$$

and the total computation workload at node $i \in \mathcal{V}$ is given by

$$G_i = \sum\nolimits_{(a,k)\in\mathcal{S}} w_i(a,k) g_i(a,k),$$

where $w_i(a,k)$ is the *computational weight*, i.e., the computation workload for node i to perform the $(k+1)$-th task of application a on a single input packet. Fig. 3 gives a detailed illustration of how the flows are directed within one node.

Non-linear costs are incurred on the links due to transmission, and on the nodes due to computation. We denote the transmission cost on link (i,j) by $D_{ij}(F_{ij})$, where function $D_{ij}(\cdot)$ is continuously differentiable, monotonically increasing and convex, with $D_{ij}(0) = 0$. Such $D_{ij}(\cdot)$ subsumes various existing cost functions, including linear cost (transmission delay), link capacity constraints. It also incorporates congestion-dependent performance metrics. For example, let μ_{ij} be the service rate of an M/M/1 queue, then $D_{ij}(F_{ij}) = F_{ij}/(\mu_{ij} - F_{ij})$ gives the average number of packets waiting in the queue or being served on (i,j) [14]. The computation cost at i is denoted by $C_i(G_i)$, where $C_i(\cdot)$ is also increasing, continuously differentiable and convex, with $C_i(0) = 0$. Function $C_i(G_i)$ can incorporate computation congestion (e.g., average number of packets waiting for available processor or being served at CPU). When both $D_{ij}(F_{ij})$ and $C_i(G_i)$ represent queue lengths, by Little's Law, the network aggregated cost is proportional to the expected packet system delay. Our major notation is summarized in Table I.

We aim to minimize the aggregated transmission and computation cost in the network, formally cast as[3]

$$\min_{\boldsymbol{\phi}} \quad D(\boldsymbol{\phi}) = \sum\nolimits_{(i,j)\in\mathcal{E}} D_{ij}(F_{ij}) + \sum\nolimits_{i\in\mathcal{V}} C_i(G_i) \quad (2)$$
$$\text{subject to} \quad \phi_{ij}(a,k) \in [0,1], \text{ and (1) holds.}$$

III. Optimality Conditions

To tackle the non-convex problem (2), we first present the KKT necessary condition, and demonstrate that the KKT condition can yield arbitrarily worse performance when compared to that of the global optimum. Then, we propose a sufficient condition for optimality by modifying the KKT condition.

We start by giving closed-form derivatives of $D(\boldsymbol{\phi})$. Our analysis generalizes [5] and makes non-trivial extensions incorporating service chain flows. For link (i,j) and stage (a,k), the marginal cost of increasing $\phi_{ij}(a,k)$ is equivalent to a sum of two parts, 1) the marginal communication cost on link (i,j) due to increasing of $f_{ij}(a,k)$, and 2) the marginal cost due to increasing of node j's traffic $t_j(a,k)$. Formally, for $j \neq 0$,

$$\frac{\partial D}{\partial \phi_{ij}(a,k)} = t_i(a,k)\left(L_{(a,k)} D'_{ij}(F_{ij}) + \frac{\partial D}{\partial t_j(a,k)}\right), \quad (3a)$$

For $j = 0$, the marginal cost of increasing $\phi_{i0}(a,k)$ consists of 1) marginal computation cost at node i, and 2) marginal cost for increasing next-stage traffic $t_i(a,k+1)$. Namely[4],

$$\frac{\partial D}{\partial \phi_{i0}(a,k)} = t_i(a,k)\left(w_i(a,k)C'_i(G_i) + \frac{\partial D}{\partial t_i(a,k+1)}\right). \quad (3b)$$

In (3), $\partial D/\partial t_i(a,k)$ is a weighted sum of marginal costs for out-going links and local CPU, i.e., if $k \neq |\mathcal{T}_a|$,

$$\frac{\partial D}{\partial t_i(a,k)} = \phi_{i0}(a,k)\left(w_i(a,k)C'_i(G_i) + \frac{\partial D}{\partial t_i(a,k+1)}\right) + \sum_{j\in\mathcal{V}} \phi_{ij}(a,k)\left(L_{(a,k)} D'_{ij}(F_{ij}) + \frac{\partial D}{\partial t_j(a,k)}\right), \quad (4a)$$

and for $k = |\mathcal{T}_a|$, recall $\phi_{i0}(a,|\mathcal{T}_a|) \equiv 0$, $\frac{\partial D}{\partial t_i(a,|\mathcal{T}_a|)}$ equals

$$\sum_{j\in\mathcal{V}} \phi_{ij}(a,|\mathcal{T}_a|)\left(L_{(a,|\mathcal{T}_a|)} D'_{ij}(F_{ij}) + \frac{\partial D}{\partial t_j(a,|\mathcal{T}_a|)}\right). \quad (4b)$$

Recall the fact that final results do not introduce any cost at destination, i.e., $\partial D/\partial t_{d_a}(a,|\mathcal{T}_a|) \equiv 0$, one could calculate $\partial T/\partial t_i(a,k)$ recursively by (4). With the absence of computation offloading, a rigorous proof of (3), (4) is provided in [12] Theorem 2. Then, Lemma 1 gives the KKT necessary optimality condition for problem (2).

Lemma 1 (KKT necessary condition)**.** *Let $\boldsymbol{\phi}$ be an optimal solution to* (2)*, then for all $i \in \mathcal{V}$, $j \in \{0\}\cup\mathcal{V}$, and $(a,k) \in \mathcal{S}$,*

$$\frac{\partial D}{\partial \phi_{ij}(a,k)} \begin{cases} = \min_{j'\in\{0\}\cup\mathcal{V}} \frac{\partial D}{\partial \phi_{ij'}(a,k)}, & \text{if } \phi_{ij}(a,k) > 0, \\ \geq \min_{j'\in\{0\}\cup\mathcal{V}} \frac{\partial D}{\partial \phi_{ij'}(a,k)}, & \text{if } \phi_{ij}(a,k) = 0. \end{cases} \quad (5)$$

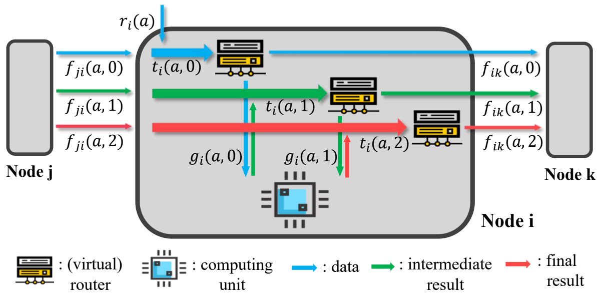

Fig. 3: Detailed behavior of flows on nodes j,i,k when $|\mathcal{T}_a| = 2$. Node i handles traffic $t_i(a,k)$ for different stage (a,k) via (virtual) routers. Each router is controlled by variable $[\phi_{ij}(a,k)]_{j\in\{0\}\cup\mathcal{V}}$.

[3]We do not explicitly impose any link or computation capacity constraints in (2), as they are already incorporated in the cost functions.

[4]For coherence, we let $\partial D/\partial\phi_{ij}(a,k) \equiv \infty$ for $(i,j) \notin \mathcal{E}$, and $\partial D/\partial\phi_{i0}(a,|\mathcal{T}_a|) \equiv \infty$ as the final results will not be further computed.

Proof of Lemma 1 is omitted due to space limit. Condition (5) only gives the necessary condition for optimality. In fact, variable $\boldsymbol{\phi}$ satisfying condition (5) may perform arbitrarily worse than the global optimal solution.

Proposition 1. *For any $0 < \rho < 1$, there exists a scenario (i.e., network $\mathcal{G}$, applications $\mathcal{A}$, cost functions $F_{ij}(\cdot)$, $G_i(\cdot)$, and input rates $r_i(a)$) such that $\frac{D(\boldsymbol{\phi}^*)}{D(\boldsymbol{\phi})} = \rho$, where $\boldsymbol{\phi}$ is feasible to* (2) *and satisfies* (5)*, and $\boldsymbol{\phi}^*$ is an optimal solution to* (2)*.*

A scenario with $\frac{D(\boldsymbol{\phi}^*)}{D(\boldsymbol{\phi})} = \rho$ is demonstrated in Fig. 4. The suboptimality of condition (5) is incurred by degenerate cases at i and (a,k) satisfying $t_i(a,k) = 0$, where (5) automatically holds regardless of the actual forwarding variables $\phi_{ij}(a,k)$.

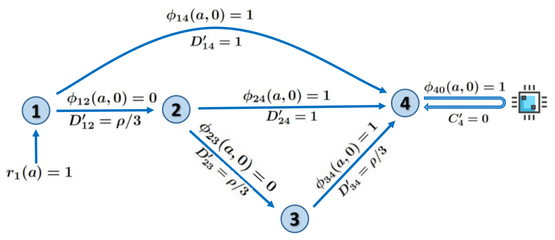

Fig. 4: The KKT condition (5) leads to arbitrarily suboptimal solution. Consider application a with $|\mathcal{T}_a| = 1, d_a = 4$. Data is input at 1 and CPU is only equipped at 4. All cost functions are linear. Variable $\boldsymbol{\phi}$ shown in the figure satisfies (5) with $D(\boldsymbol{\phi}) = 1$. However, the optimal strategy is to forward all data on path $1 \to 2 \to 3 \to 4$, with $D(\boldsymbol{\phi}^*) = \rho$. Thus Proposition 1 holds by letting ρ arbitrarily small.

We now address the non-sufficiency of condition (5). Inspired by [5], we introduce a modification to (5) that leads to a sufficient condition for optimality in (2). Observing that for any i and (a,k), the traffic term $t_i(a,k)$ repeatedly exists in RHS of (3) for all $j \in \{0\} \cup \mathcal{V}$. Thus when $t_i(a,k) = 0$, it always holds that $\phi_{ij}(a,k) = 0$. We thereby remove the traffic terms in (5), leading to condition (6).

Theorem 1 (Sufficiency condition)**.** *Let $\boldsymbol{\phi}$ be feasible to* (2)*. Then $\boldsymbol{\phi}$ is a global optimal solution to* (2) *if the following holds for all $i \in \mathcal{V}$, $j \in \{0\} \cup \mathcal{V}$ and $(a,k) \in \mathcal{S}$,*

$$\delta_{ij}(a,k) \begin{cases} = \min\limits_{j' \in \{0\} \cup \mathcal{V}} \delta_{ij'}(a,k), \text{ if } \phi_{ij}(a,k) > 0, \\ \geq \min\limits_{j' \in \{0\} \cup \mathcal{V}} \delta_{ij'}(a,k), \text{ if } \phi_{ij}(a,k) = 0, \end{cases} \quad (6)$$

where $\delta_{ij}(a,k)$ is the "modified marginal", given by

$$\delta_{ij}(a,k) = \begin{cases} L_{(a,k)} D'_{ij}(F_{ij}) + \frac{\partial D}{\partial t_j(a,k)}, & \text{if } j \neq 0, \\ w_i(a,k) C'_i(G_i) + \frac{\partial D}{\partial t_i(a,k+1)}, & \text{if } j = 0. \end{cases} \quad (7)$$

Proof of Theorem 1 is provided in our supplementary material [15]. To see the difference between the KKT necessary condition (5) and the sufficiency condition (6), consider again the example in Fig. 4. For any $\boldsymbol{\phi}$ satisfying (6), it must hold that $\phi_{12}(a,0) = 1$, $\phi_{23}(a,0) = 1$ and $\phi_{34}(a,0) = 1$, precisely indicating the shortest path $1 \to 2 \to 3 \to 4$ as expected.

IV. Distributed Algorithm

We propose a distributed algorithm that converges to the sufficiency condition (6). The algorithm generalizes the method in [12] to service chain computation placement. It is adaptive to changes in data input rates and network topology, and can be implemented as an online algorithm.

Existence of routing loops generates redundant flow circulation and causes potential instability. We say $\boldsymbol{\phi}$ has a *loop* of stage (a,k) if there exists i, $j \in \mathcal{V}$, such that i has a path[5] of stage (a,k) to j, and vice versa. We say $\boldsymbol{\phi}$ is *loop-free* if no loops are formed for any stage $(a,k) \in \mathcal{S}$. We assume the network starts with a feasible and loop-free strategy $\boldsymbol{\phi}^0$, and the initial cost $D^0 = D(\boldsymbol{\phi}^0)$ is finite. Time is partitioned into slots of duration T. At $(t+1)$-th slot $(t \geq 0)$, node i's forwarding/offloading strategy $\boldsymbol{\phi}_i$ is updated as follows,

$$\boldsymbol{\phi}_i^{t+1} = \boldsymbol{\phi}_i^t + \Delta\boldsymbol{\phi}_i^t, \quad (8)$$

where the update vector $\Delta\boldsymbol{\phi}_i^t$ is calculated by

$$\Delta\phi_{ij}^t(a,k) = \begin{cases} -\phi_{ij}^t(a,k), & \text{if } j \in \mathcal{B}_i^t(a,k) \\ S_i^t(a,k)/N_i^t(a,k), & \text{else if } e_{ij}^t(a,k) = 0 \\ -\min\left\{\phi_{ij}^t(a,k), \alpha e_{ij}^t(a,k)\right\}, & \text{else if } e_{ij}^t(a,k) > 0 \end{cases} \quad (9)$$

where $\mathcal{B}_i^t(a,k)$ is the set of *blocked nodes* to suppress routing loops, α is the stepsize, and

$$e_{ij}^t(a,k) = \delta_{ij}^t(a,k) - \min\nolimits_{j \notin \mathcal{B}_i^t(a,k)} \delta_{ij}^t(a,k), \; \forall j \notin \mathcal{B}_i^t(a,k),$$
$$N_i^t(a,k) = \left| \left\{ j \notin \mathcal{B}_i^t(a,k) \middle| e_{ij}^t(a,k) = 0 \right\} \right|, \quad (10)$$
$$S_i^t(a,k) = \sum\nolimits_{j: e_{ij}^t(a,k) > 0} \Delta\phi_{ij}^t(a,k).$$

Our method can be viewed as a variant of gradient projection. It transfers $\phi_{ij}(a,k)$ from non-minimum-marginal directions to the minimum-marginal ones (here "marginal" refers to $\delta_{ij}(a,k)$ defined in (7)), until (6) is satisfied. We next provide details regarding the blocked node set $\mathcal{B}_i^t(a,k)$, and introduce a distributed online mechanism to estimate $\delta_{ij}^t(a,k)$.

Blocked node set. To ensure feasibility and loop-free, we adopt the method of blocked node sets following [12]. Combining Theorem 1 with (4), if $\boldsymbol{\phi}$ is a global optimal solution satisfying (6), for any stage (a,k), the value of $\partial D / \partial t_i(a,k)$ should decrease monotonically along any path of stage (a,k). Thus node i should not forward flow of stage (a,k) to a neighbor j if either 1) $\partial D/\partial t_j(a,k) > \partial D/\partial t_i(a,k)$, or 2) j could form a path containing some link (p,q) such that $\partial D/\partial t_q(a,k) > \partial D/\partial t_p(a,k)$. The set containing nodes of these two categories, along with j with $(i,j) \notin \mathcal{E}$, is marked as the *blocked node set* $\mathcal{B}_i(a,k)^t$. Practically, the information needed to determine blocked node sets could be piggy-backed on the broadcast messages described shortly. If such a blocking mechanism is implemented in each iteration, the loop-free property is guaranteed to hold throughout the algorithm.

Marginal cost broadcast. Recall from (7) that in order to calculate $\boldsymbol{\delta}_i(a,k)$, node i needs the link marginal costs $D'_{ij}(F_{ij})$, computation marginal costs $C'_i(G_i)$, and the marginal costs due to traffic term, i.e., $\partial D/\partial t_j(a,k)$ and $\partial D/\partial t_i(a,k+1)$.

[5]A *path* from i to j is a sequence of nodes $n_1, \cdots, n_L$ with $(n_l, n_{l+1}) \in \mathcal{E}$ and $\phi_{n_l n_{l+1}}(a,k) > 0$ for $l = 1, \cdots, L-1$, and $n_1 = i$, $n_L = j$.

Suppose $D_{ij}(\cdot)$ and $C_i(\cdot)$ are known in closed-form, nodes can directly measure $D'_{ij}(F_{ij})$ and $C'_i(G_i)$ while transmitting on link (i,j) and performing computation (or equivalently, first measure flows F_{ij} and workloads G_i, then substitute into $D_{ij}(\cdot)$ and $C_i(\cdot)$). To collect $\partial D/\partial t_i(a,k)$, we use recursive calculate (4) starting with $i = d_a$ and $k = |\mathcal{T}_a|$ satisfying $\partial D/\partial t_{d_a}(a, |\mathcal{T}_a|) = 0$. To carry out this recursive calculation, we apply a multi-stage distributed broadcast protocol to every application $a \in \mathcal{A}$, described as follows:

1) Broadcast of $\partial D/\partial t_i(a, |\mathcal{T}_a|)$: Each node i first waits until it receives messages carrying $\partial D/\partial t_j(a, |\mathcal{T}_a|)$ from all its downstream neighbors $j \in \mathcal{V}$ with $\phi_{ij}(a, |\mathcal{T}_a|) > 0$. Then, node i calculates $\partial D/\partial t_i(a, |\mathcal{T}_a|)$ by (4b) with the measured $D'_{ij}(F_{ij})$ and received $\partial D/\partial t_j(a, |\mathcal{T}_a|)$. Next, node i broadcasts newly calculated $\partial D/\partial t_i(a, |\mathcal{T}_a|)$ to all its upstream neighbors $k \in \mathcal{V}$ with $\phi_{ki}(a, |\mathcal{T}_a|) > 0$. (This stage starts with the destination d_a, where d_a broadcasts $\partial D/\partial t_{d_a}(a, |\mathcal{T}_a|) = 0$ to its upstream neighbors.)
2) Broadcast of $\partial D/\partial t_i(a,k)$ for $k \neq |\mathcal{T}_a|$: (This stage starts with $k = |\mathcal{T}_a| - 1$ and is repeated with k being abstracted by 1 each time, until $k = 0$.) Suppose every node i has calculated $\partial D/\partial t_i(a,k')$ for all $k' \geq k+1$. Then, similar as stage 1), $\partial D/\partial t_i(a,k)$ can be calculated recursively by (4a) via broadcast. Besides $\partial D/\partial t_j(a,k)$ from all downstream neighbors j, node i must also obtain $\partial D/\partial t_i(a,k+1)$ and $C'_i(G_i)$ for (4a). For each k, this stage starts at nodes i where $\phi_{ij}(a,k) = 0$ for all $j \in \mathcal{V}$. i.e., the end-nodes of stage (a,k) paths.

With loop-free guaranteed, the broadcast procedure above is guaranteed to traverse throughout the network for all $(a,k) \in \mathcal{S}$ and terminate within a finite number of steps. We summarize our proposed method in Algorithm 1.

Algorithm 1: Gradient Projection (GP)

```
1 Start with $t = 0$ and a loop-free $\phi^0$ with $D^0 < \infty$.
2 do
3     Each node $i$ obtain $\partial D/\partial t_i(a,k)$ for all $(a,k) \in \mathcal{S}$
        via the marginal cost broadcast.
4     Node $i$ calculates $\delta_{ij}(a,k)^t$ by (7) for $j \in \{0\} \cup \mathcal{V}$.
5     Node $i$ calculates the blocked node sets $\mathcal{B}_{ij}(a,k)^t$.
6     Node $i$ obtain $\Delta\boldsymbol{\phi}_i^t$ by (9) and updates $\boldsymbol{\phi}_i^t$ by (8).
7 when  end of iteration $t$;
```

Convergence and complexity. The proposed algorithm can be implemented in an online fashion as it does not require prior knowledge of data input rates $r_i(a)$. Moreover, it is adaptive to changes in $r_i(a)$ since flow rates F_{ij} and workload G_i can be directly estimated by the packet number on links and CPU in previous time slots. It can also adapt to changes in the network topology: whenever a link (i,j) is removed from $\mathcal{E}$, node i only needs to add j to the blocked node set; when link (i,j) is added to $\mathcal{E}$, node i removes j from the blocked node set.[6]

[6]When a new node v is added to the network, it can randomly initiate $\boldsymbol{\phi}_v$ with (1). $t_v(a,k)$ will be automatically updated by the marginal cost broadcast in the next time slot, and the loop-free property will be guaranteed.

Theorem 2. *Assume the network starts at $\boldsymbol{\phi}^0$ with $D^0 < \infty$, and $\boldsymbol{\phi}^t$ is updated by Algorithm 1 with a sufficiently small stepsize α. Then, the sequence $\{\boldsymbol{\phi}^t\}_{t=0}^{\infty}$ converges to a limit point $\boldsymbol{\phi}^*$ that satisfies condition* (6).

The proof of Theorem 2 is a straightforward extension of [12] Theorem 5, and is omitted due to space limitation. The stepsize α that guarantees convergence can also be found in [12]. We remark that the convergence property of Algorithm 1 can be improved by adopting second-order quasi-Newton methods, e.g., the method in [5] speeds up convergence while guaranteeing convergence from any initial point.

Recall that $\boldsymbol{\phi}$ is updated every time slot of duration T, every broadcast message is sent once in every slot. Thus there are $|\mathcal{E}|$ broadcast message transmissions for each stage in one slot, and totally $|\mathcal{S}||\mathcal{E}|$ per slot, with on average $|\mathcal{S}|/T$ per link/second, and at most $|\mathcal{S}|\bar{d}$ for each node, where $\bar{d}$ is the largest out-degree. We assume the broadcast messages are sent in an out-of-band channel. Let t_c be the maximum transmission time for a broadcast message, and $\bar{h}$ be the maximum hop number for all paths. The broadcast completion time for application a is at most $(|\mathcal{T}_a| + 1)\bar{h}t_c$. Moreover, the proposed algorithm has space complexity $O(|\mathcal{S}|)$ at each node. The update may fail if broadcast completion time exceeds T or $|\mathcal{S}|/T$ exceeds the broadcast channel capacity. If so, we can use longer slots, or allow some nodes to perform updates every multiple slots. Meanwhile, if $|\mathcal{S}|$ is large, the algorithm overhead can be significantly reduced by applying our algorithm only to the top applications causing most network traffic.

V. Numerical evaluation

We evaluate the proposed algorithm **GP** via a flow-level simulator presented at [16]. We implement several baselines and compare against GP over different network scenarios summarized in Table II. Specifically, **Connected-ER** is a connectivity-guaranteed Erdős-Rényi graph. **Balanced-tree** is a complete binary tree. **Fog** is a sample topology for fog-computing [17]. **Abilene** is the predecessor of *Internet2 Network*. **GEANT** is a pan-European research and education data network. **SW** (small-world) is a ring-like graph with additional short-range and long-range edges. Table II also summarizes the number of nodes $|\mathcal{V}|$, edges $|\mathcal{E}|$, and applications $|\mathcal{A}|$. We assume each application has R random active data sources (i.e., the nodes i for which $r_i(a) > 0$), and $r_i(a,k)$ for each data source is chosen u.a.r. in $[0.5, 1.5]$. **Link** is the type of $D_{ij}(\cdot)$, where *Linear* denotes a linear cost $D_{ij}(F_{ij}) = d_{ij}F_{ij}$, and *Queue* denotes the non-linear cost $D_{ij}(F_{ij}) = \frac{F_{ij}}{d_{ij} - F_{ij}}$. **Comp** is the type of $C_i(G_i)$, where *Linear* denotes $C_i(G_i) = s_i \sum w_i(a,k) g_i(a,k)$, and *Queue* denotes $C_i(G_i) = \frac{G_i}{s_i - G_i}$.

We compare GP against: **SPOC** (Shortest Path Optimal Computation placement) fixes the forwarding variables to the shortest path (measured with marginal cost at $F_{ij} = 0$) and solves the optimal offloading along these paths. **LCOF** (Local Computation placement Optimal Forwarding) computes all exogenous input flows at their data sources, and optimally routes the result to destinations. **LPR-SC** (Linear Program

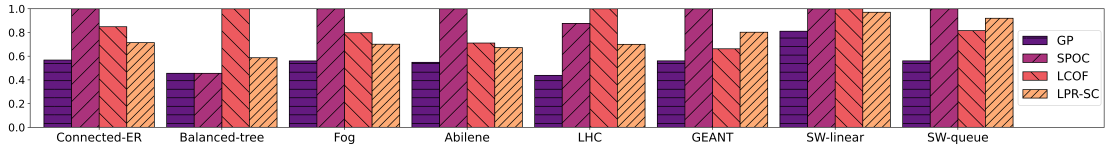

Fig. 5: Normalized total cost $D(\phi)$ for network scenarios in Table II

Network Topology	Parameters							
	$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	$\|\mathcal{A}\|$	R	**Link**	$\bar{d}_{ij}$	**Comp**	$\bar{s}_i$
Connected-ER	20	40	5	3	Queue	10	Queue	12
Balanced-tree	15	14	5	3	Queue	20	Queue	15
Fog	19	30	5	3	Queue	20	Queue	17
Abilene	11	14	3	3	Queue	15	Queue	10
LHC	16	31	8	3	Queue	15	Queue	15
GEANT	22	33	10	5	Queue	20	Queue	20
SW	100	320	30	8	(both)	20	(both)	20
Other Parameters	$\|\mathcal{T}_a\| = 2$, $r_i(a) \in [0.5, 1.5]$, $L_{(a,k)} = 10 - 5k$							

TABLE II: Simulated Network Scenarios

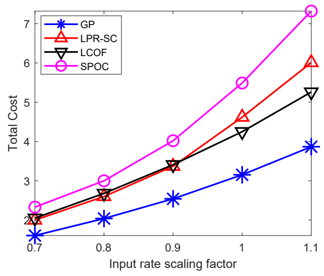

Fig. 6: $D(\phi)$ vs Data input rates

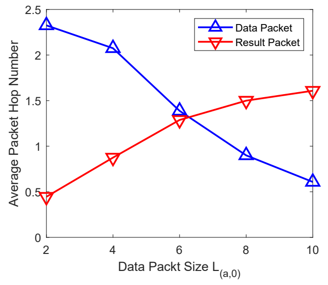

Fig. 7: Packet hop number vs $L_{(a,0)}$

Rounded for Service Chain) is the joint routing and offloading method by [18] (we extend this method heuristically to service chain applications), which does not consider link congestion.

Fig.5 compares the total cost (normalized according to the worst performing algorithm) across scenarios in Table II. We test both linear cost and convex queueing cost with other parameters fixed in `SW`, labeled as `SW-linear` and `SW-queue`. The proposed algorithm `GP` significantly outperforms other baselines in all simulated scenarios, with as much as 50% improvement over `LPR-SC` which also jointly optimizes routing and task offloading. Case `SW-linear` and `SW-queue` suggest that `GP` promises a more significant delay improvement for networks with queueing effect. Fig.6 shows total cost versus exogenous input rates $r_i(a)$ in `Abilene`. The performance advantage of `GP` quickly grows as the network becomes more congested. In Fig. 7, we compare average hop numbers travel by data and result packets for `GP` over the ratio of packet size. The average computation offloading distance grows when $L_{(a,0)}$ becomes relatively small, i.e., `GP` tends to offload tasks with larger data size nearer to requester and farther from the destination node.

VI. Conclusion

We propose a joint forwarding and offloading model for service chain applications in CEC. We formulate a non-convex total cost minimization problem and optimally solve it by providing sufficient optimality conditions. We devise a distributed and adaptive online algorithm that reaches the global optimal. Our method achieves delay-optimal forwarding and offloading for service chain computations, and can be applied to embed computation-intensive complex applications, e.g., DNN, into CEC networks. Our future work focuses on extending the proposed framework to incorporate general interdependency among tasks, e.g., directed acyclic graphs.

References

[1] Y. Sahni, J. Cao, L. Yang, and Y. Ji, "Multi-hop multi-task partial computation offloading in collaborative edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 5, pp. 1133–1145, 2020.

[2] Y. Sahni, J. Cao, S. Zhang, and L. Yang, "Edge mesh: A new paradigm to enable distributed intelligence in internet of things," *IEEE access*, vol. 5, pp. 16 441–16 458, 2017.

[3] K. Zhu, W. Zhi, X. Chen, and L. Zhang, "Socially motivated data caching in ultra-dense small cell networks," *IEEE Network*, vol. 31, no. 4, pp. 42–48, 2017.

[4] Z. Hong, W. Chen, H. Huang, S. Guo, and Z. Zheng, "Multi-hop cooperative computation offloading for industrial iot–edge–cloud computing environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 12, pp. 2759–2774, 2019.

[5] J. Zhang, Y. Liu, and E. Yeh, "Optimal congestion-aware routing and offloading in collaborative edge computing," in *2022 20th International Symposium on Modeling and Optimization in Mobile, Ad hoc, and Wireless Networks (WiOpt)*. IEEE, 2022, pp. 121–128.

[6] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE communications magazine*, vol. 53, no. 2, pp. 90–97, 2015.

[7] J. Zhang, A. Sinha, J. Llorca, A. M. Tulino, and E. Modiano, "Optimal control of distributed computing networks with mixed-cast traffic flows," *IEEE/ACM Transactions on Networking*, 2021.

[8] M. Wang, B. Cheng, B. Li, and J. Chen, "Service function chain composition and mapping in nfv-enabled networks," in *2019 IEEE World Congress on Services (SERVICES)*. IEEE, 2019.

[9] F. Khoramnejad, R. Joda, and M. Erol-Kantarci, "Distributed multi-agent learning for service function chain partial offloading at the edge," in *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE, 2021, pp. 1–6.

[10] Y. Sahni, J. Cao, and L. Yang, "Data-aware task allocation for achieving low latency in collaborative edge computing," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 3512–3524, 2018.

[11] P. Ghosh, Q. Nguyen, P. K. Sakulkar, J. A. Tran, A. Knezevic, J. Wang, Z. Lin, B. Krishnamachari, M. Annavaram, and S. Avestimehr, "Jupiter: a networked computing architecture," in *14th IEEE/ACM International Conference on Utility and Cloud Computing Companion*, 2021.

[12] R. Gallager, "A minimum delay routing algorithm using distributed computation," *IEEE transactions on communications*, vol. 25, 1977.

[13] J. Zhang and E. Yeh, "Congestion-aware routing and content placement in elastic cache networks," *arXiv preprint arXiv:2303.01648*, 2023.

[14] D. Bertsekas and R. Gallager, *Data networks*. Athena Scientific, 2021.

[15] J. Zhang and E. Yeh, "Delay-optimal service chain forwarding and offloading in collaborative edge computing," 2023. [Online]. Available: https://arxiv.org/abs/2310.06141

[16] J. Zhang, "Joint-Routing-and-Computation-2022." [Online]. Available: https://github.com/JinkunZhang/Joint-Routing-and-Computation-2022

[17] K. Kamran, E. Yeh, and Q. Ma, "Deco: Joint computation, caching and forwarding in data-centric computing networks," in *Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2019, pp. 111–120.

[18] B. Liu, Y. Cao, Y. Zhang, and T. Jiang, "A distributed framework for task offloading in edge computing networks of arbitrary topology," *IEEE Transactions on Wireless Communications*, vol. 19, no. 4, 2020.