DECO: Joint Computation Scheduling, Caching, and Communication in Data-Intensive Computing Networks

Khashayar Kamran, *Student Member, IEEE*, Edmund Yeh, *Senior Member, IEEE*, and Qian Ma, *Member, IEEE*

***Abstract*— Driven by technologies such as IoT-enabled health care, machine learning applications at the edge, and industrial automation, mobile edge and fog computing paradigms have reinforced a general trend toward decentralized computing, where any network node can route traffic, compute tasks, and store data, possibly at the same time. In many such computing environments, there is a need to cache significant amounts of data, which may include large data sets, machine learning models, or executable code. In this work, we propose a framework for joint computation scheduling, caching, and request forwarding within such decentralized computing environments. We first characterize the stability region of a "genie-aided" computing network where data required by computation are instantly accessible, and develop a throughput optimal control policy for this model. Based on this, we develop a practically implementable distributed and adaptive algorithm, and show that it exhibits superior performance in terms of average task completion time, when compared to several baseline policies.**

***Index Terms*— Edge computing, fog computing, computation scheduling, distributed computing, data-intensive computing, caching.**

I. Introduction

CENTRALIZED clouds have dominated IT service delivery over the past decade. Operation over the internet, and the clouds' low cost of operation [2] have made them the primary means of achieving energy efficiency and computation speed-up for resource-poor devices [3], [4]. Computation offloading to the cloud for mobile users has been studied extensively in the literature, and various software platforms [5]–[8] and analytical models [9], [10] for optimal offloading have been proposed. Recently, the cost efficiency and scalability of the centralized cloud have been challenged by the emergence of Internet of Things (IoT) devices and the predicted increase in services with ultra low latency requirements (one millisecond or less) [2], [11]. This has made paradigms such as fog computing [12] and mobile edge computing more appealing. In this paper, we refer to the family of such paradigms as *dispersed computing*.

In the fog computing paradigm, networking, computation, and storage resources are distributed at different hierarchical levels from the core of the network to the edge. In mobile edge computing, these resources are distributed throughout the mobile edge close to users. As the number of delay-sensitive applications increases, these platforms have the potential to outperform centralized cloud architectures in terms of request satisfaction delay [11]. The potential benefits of such paradigms are accompanied by challenges in distributed implementation and control. Another challenge is the increased popularity of media-rich and data-intensive applications where computations are often designed to be performed on large pieces of data stored in the network. Examples include IoT enabled health care and medical data analytics [13]–[15], machine learning at the edge [16]–[19], data intensive scientific computation [20], data processing for wearable devices [21], intelligent driving and transportation systems [22], in-network image/video processing [23]–[25].

A fundamental question in dispersed computing is how to optimally utilize the processing, storage and bandwidth resources in the network to accomplish data-intensive computation with high throughput and low latency. Specifically, how to forward computation requests, perform computations, and move and store data in the network? For instance, should the system bring data to the computation-requesting node for computation or take the computation to the data server? How can one provide a solution in a distributed and adaptive[1] manner with general network topology and request patterns?

While previous work has addressed aspects of the question raised above, the problem has not yet been solved in a coherent manner. To the best of our knowledge, this paper is the first to study joint computation scheduling, forwarding, and caching within an adaptive and distributed setting.

In this paper, we consider a data-intensive dispersed computing network with arbitrary topology and heterogeneous processing, communication, and storage resources available at

Manuscript received October 29, 2020; revised June 13, 2021 and October 13, 2021; accepted November 16, 2021; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor B. Ji. Date of publication December 28, 2021; date of current version June 16, 2022. This work was supported in part by DARPA under Grant HR0011-17-C-0050, in part by the National Science Foundation under Grant NeTS-1718355, and in part by the National Natural Science Foundation of China under Grant 62002399. A conference version of this paper is published in MobiHoc'19 [1] [DOI: 10.1145/3323679.3326509]. *(Corresponding author: Qian Ma.)*

Khashayar Kamran is with Pinterest, Inc., San Fransisco, CA 94107 USA (e-mail: khashayar.kamran@gmail.com).

Edmund Yeh is with the Department of Electrical and Computer Engineering, Northeastern University, Boston, MA 02115 USA (e-mail: eyeh@ece.neu.edu).

Qian Ma is with the School of Intelligent Systems Engineering, Sun Yat-sen University, Guangzhou 510275, China (e-mail: maqian25@mail.sysu.edu.cn).

This article has supplementary downloadable material available at https://doi.org/10.1109/TNET.2021.3136157, provided by the authors.

Digital Object Identifier 10.1109/TNET.2021.3136157

[1]By adaptive, we mean that control algorithms do not require prior knowledge of computation request rates.

each node. Users issue *computation requests* for performing a computation task on a required piece of data (e.g., inference using a trained model saved in the network). This computation request, along with input arguments (e.g., input data for inference), is forwarded through the network until a node decides to perform the task locally. This node can process the computation only when it has the required data object, either from its own cache or fetched from other nodes by issuing a *data request*. After the data request arrives at a data server or caching point, a copy of the data is sent back to the data requester on the reverse path. Each node on the path can optionally cache the data for future use. The data requester then processes the task, and sends the result back to the original computation requester on the reverse path (of the computation request). The question is how nodes should decide on computation request forwarding, data request forwarding, computation scheduling, and caching in an adaptive and distributed manner. High throughput and low latency are both important performance metrics to consider in designing computations, communication, and caching algorithms for data-intensive computing networks. In this paper, our focus is on designing algorithms to increase throughput, and we left the problem of minimizing the latency for future work. It is worth noting that low latency and high throughput are not mutually exclusive. As we show through extensive simulations, providing high throughput often leads to low latency.

We propose the DECO (Data-intEnsive COmputation) framework. DECO utilizes the queue sizes of computation interest packets and data interest packets to capture the measured demands for computations and data objects, respectively.[2] In this framework, we first consider a "genie-aided" version of the system where nodes have the ability to instantly cache the data and compute the results, without waiting for the data to come back. For the "genie-aided" system, we provide a throughput optimal algorithm and we derive the stability region, which is an outer bound for the stability region of the original system. We then propose an implementable version of the throughput optimal algorithm for handling computation requests, data requests, data objects and computation results. The superior performance of the DECO, compared to many baseline algorithms, is shown through extensive simulations.

Our key contributions are summarized as follows:

- To our best knowledge, this is the first work providing outer bounds for the stability region of a data-intensive computing network with an arbitrary topology. We present a throughput optimal algorithm for a "genie-aided" setting where nodes are allowed to cache data and perform computation, without waiting for the data to be fetched. The results are not straight forward extensions of previous works on stability analysis due to the internally generated traffic in the network.
- We provide implementable versions of the throughput optimal algorithm and present an adaptive and distributed framework called DECO for joint computation scheduling, caching, and request forwarding in a data-intensive dispersed computing network. Our solution handles arbitrary topology, data catalog, and task catalog consisting of single-stage computations, and takes into account the demand for both computation and data.

[2]Interest packets as a metric for measured demand was first introduced in [26] for caching networks.

II. Related Work

Computation scheduling and resource allocation in a heterogeneous computing platform have been studied in research and practice. Topcuoglu *et al.* [27] studied the problem of static task scheduling and proposed a centralized heuristic called HEFT for scheduling tasks represented by a DAG (Directed Acyclic Graph). Pegasus [28] and CIRCE [29] were proposed as frameworks for mapping tasks to computers in a dispersed computing system. Sakulkar *et al.* studied distributed assignment of tasks to computers [30]. More recently, Król *et al.* [31] proposed a framework for distributed computing which leverages knowledge about data location. These works present frameworks for the task assignment problem, but in contrast to our paper, they do not provide any optimality guarantee or bounds on the performance of the computing network.

Optimal in-network computation scheduling has been studied in different settings. Various works have studied the optimal placement of network virtual functions [32]–[36]. Closer to our setting, Feng *et al.* [37] studied a computation scheduling problem where computation services are modeled as a chain of consecutive tasks, and there is a linear cost associated with computation and communication resources in the network. They propose a throughput optimal policy for task scheduling and request forwarding based on Lyapunov drift minimization. In a similar setting, Zhang *et al.* [38] proposed a throughput optimal policy for computing networks with uni-cast and multi-cast flows based on a layered graph model. Yang *et al.* [39] generalized the chain-service model to DAG-based service model, and proposed a throughput optimal policy for it. However, these methods do not consider data placement and caching.

Several works have studied joint processor and storage allocation. Chen *et al.* [40] proposed a solution to joint caching and computation at the mobile edge where caching is used to store the final result of computations. Zeng *et al.* [41] studied the problem of task scheduling and image placement in order to minimize the request satisfaction delay. Ndikumana *et al.* [42] consider the problem of minimizing both bandwidth consumption and network latency in a mobile edge computing platform. In addition to differences in the problem objective, in contrast to our work, these works study the problem in specific one-hop or two-hop topologies, with request arrival rates known a prior. Several papers have formulated the problem of joint computation scheduling and service caching as an integer programming or mixed integer programming [43]–[46], and proposed approximate solutions. In contrast to this paper, these solutions are centralized, and only consider one-hop and two-hop network topologies.

III. Computation Network Model

Consider a network of computing nodes, each capable of performing computation tasks, caching data objects, and

TABLE I
NOTATION SUMMARY

$\mathcal{G}(\mathcal{V},\mathcal{E})$	Network graph with nodes $\mathcal{V}$ and edges $\mathcal{E}$
$\mathcal{F}$	Catalog of available tasks
$\mathcal{D}$	Catalog of available data objects
$\mathcal{R}$	Set of available computation requests ($\mathcal{R} \subseteq \mathcal{F} \times \mathcal{D}$)
$PCR(k)$	The queue of pending computation requests for data object k at each node
P_v	Processor capacity at node $v \in \mathcal{V}$
C_v	Cache capacity at node $v \in \mathcal{V}$
C_{ab}	Transmission capacity of link $(a,b) \in \mathcal{E}$
L_k	Size of data object $k \in \mathcal{D}$
$Z_{(m,k)}$	Size of the result for request $(m,k) \in \mathcal{R}$
$q_{(m,k)}$	Computation load of request $(m,k) \in \mathcal{R}$
$\lambda_n^{(m,k)}$	Average arrival rate of request $(m,k) \in \mathcal{R}$
$Y_n^{(m,k)}(t)$	Computation interest queue count for $(m,k) \in \mathcal{R}$ in node n at the beginning of time slot t
$V_n^k(t)$	Data interest queue count for $k \in \mathcal{D}$ in node n at the beginning of time slot t
$A_n^{(m,k)}(t)$	Number of exogenous arrivals at node n for computation request (m,k) during time slot t
$\mu_{ab}^{(m,k)}(t)$	Allocated transmission rate of computation interests for (m,k) on link (a,b) during time slot t
$\nu_{ab}^{(m,k)}(t)$	Allocated transmission rate of data interests for $k \in \mathcal{D}$ on link (a,b) during time slot t
$\mu_{n,proc}^{(m,k)}(t)$	Allocated processing rate of computation interests for (m,k) at node n during time slot t
$s_n^k(t)$	Caching state for object k at node n during slot t

communicating with other computing nodes. We model the network as a directed graph $\mathcal{G}(\mathcal{V},\mathcal{E})$ with $\mathcal{V}$ and $\mathcal{E}$ representing network nodes and links respectively. Assume that $(b,a) \in \mathcal{E}$ whenever $(a,b) \in \mathcal{E}$. Each node $v \in \mathcal{V}$ is equipped with a processor with capacity of P_v (in instructions per second) and a cache with capacity of C_v (in bits). We let the transmission capacity on link $(a,b) \in \mathcal{E}$ be C_{ab} (in bits per second).

There is a catalog $\mathcal{F}$ of computation tasks available in the network. These computation tasks operate on a set $\mathcal{D}$ of data objects. We specify a computation request by a pair $(m,k) \in \mathcal{R} \subseteq \mathcal{F} \times \mathcal{D}$, indicating a request for performing the m-th computation task on the k-th data object. Each request is associated with unique user-specified inputs with negligible size compared to the required data object. This makes each computation result inherently unique. Since the cardinality of the space of the user-specified inputs can be potentially huge, we do not consider caching of the final results of the computations here. We assume that the computation load and the size of the result are determined by the computation task and the data object. Let the size of the k-th data object be L_k (in bits) and let the size of the result of m-th task on k-th data object be $Z_{(m,k)}$ (in bits). The computation load of performing task m on data object k is denoted by $q_{(m,k)}$ (in number of instructions). We assume that for each data object $k \in \mathcal{D}$, there is a designated node denoted by $src(k) \in \mathcal{V}$ which serves as the permanent source of the data object.[3] Without loss of generality, we assume that each designated source stores the permanently stored data object(s) in excess memory outside its cache. In addition, designated sources are equipped with an extra processing capacity equal to the processing power needed for computing all tasks in $\mathcal{F}$ on the permanently stored data object(s); this is not included in the processor capacities at the designated sources ($P_{src(k)}$ for all $k \in \mathcal{D}$).

We assume that the routing information[4] to the source nodes, which is used to determine the output interfaces for forwarding the packets, is already populated at every node. Requests of type (m,k) arrive at node $n \in \mathcal{V}$ according to a stationary ergodic process $A_n^{(m,k)}(t)$ with the average arrival rate $\lambda_n^{(m,k)} \triangleq \lim_{t\to\infty} \frac{1}{t} \sum_{\tau=1}^{t} A_n^{(m,k)}(\tau)$. A node receiving a computation request generates a *computation interest packet* with negligible size (compared to the size of the data objects and the computation results) containing the task identification m, data identification k, as well as input arguments, to be forwarded through the network. Each node receiving a computation interest packet decides whether or not to *perform* the computation request locally. In this paper, we differentiate between *performing a computation request* and *processing a computation request*. Specifically, processing a computation request is a sub-step in performing the computation request. The difference is explained in the procedure below:

Input: A computation request $(m,k) \in \mathcal{R}$

1: **procedure** PERFORMING COMPUTATION (m,k) AT NODE n
2: **if** data object k is stored at node n **then** send computation request to the processor queue for *processing*.
3: **else**
4: put computation request in *pending computation requests for data object* k $\big(PCR(k)\big)$ queue.
5: issue a request for fetching data object k by creating a *data interest packet*.
6: **if** data object k arrives at node n **then** put the computation requests in the $PCR(k)$ queue into the processor queue in a first-come-first-served order.

If a node decides not to perform the computation locally, it can forward the computation interest packets to its neighbors. The receiving neighbor remembers the interface on which it receives the interest packet. The node which performs the computation puts the result into *result packets* and sends it back on the reverse path (of the computation interest packet) to the original requester.

A node issues a *data interest packet* whenever it decides to perform a task but does not have the required data object stored locally. As in the case for computation interest packets, nodes receiving data interest packets remember the incoming interface. When a node receives a data interest packet for an object which is in its cache, it creates a copy of that data object, puts it into *data packets*, and sends it back on the reverse path (of the data interest packet) to the requester. The satisfied data interest packet is then removed from the network. Nodes receiving data objects on the reverse path have the

[3]This setting can be extended to a scenario where there are multiple designated sources for each data object.

[4]We want to make an explicit distinction between the operations of routing and forwarding. Routing is a network-wide process which determines possible forwarding interfaces toward the data sources at each node; forwarding is the action of transferring packets to the appropriate output interface.

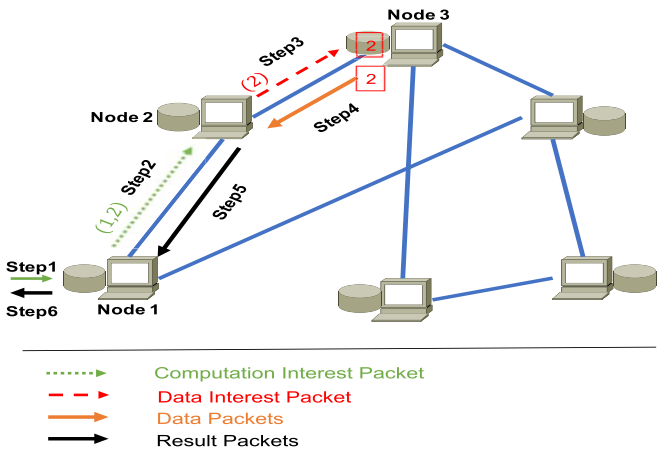


Fig. 1. A data-intensive computing network. **Step 1:** A computation request $(1, 2) \in \mathcal{R}$ arrives to Node 1. **Step 2:** Node 1 creates a computation interest packet (dotted arrow) and forwards it toward the source of data object 2 (Node 3). **Step 3:** Node 2 receives the computation interest packet and decides to perform it locally. Since it does not have data object 2 stored in the cache, it puts the computation request in the $PCR(2)$ queue and generates a data interest packet (dashed arrow) for data object 2 and forwards it toward the source (Node 3). **Step 4:** Node 3 receives a data interest packet for data object 2. It creates a copy of the data object 2 and forwards it on the reverse path toward the requester of data (Node 2). **Step 5:** Node 2 receives data object 2 and sends the pending computation request $(1, 2)$ to the processor. Once it is processed, Node 2 sends the result on the reverse path to the original requester (Node 1). **Step 6:** Node 1 delivers the result to the user.

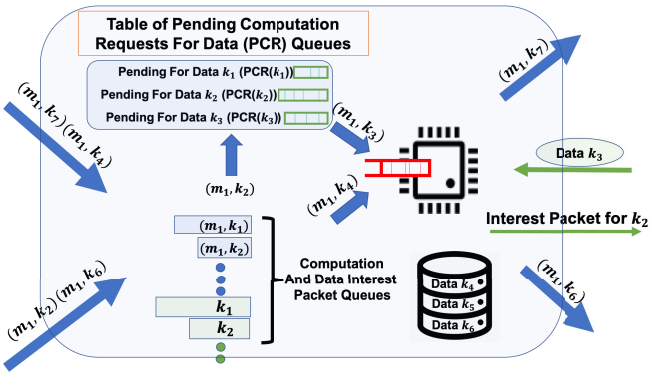


Fig. 2. Performing computation, caching, and forwarding. Computation interest packets for $(m_1, k_2), (m_1, k_4), (m_1, k_6), (m_1, k_7)$ are forwarded to the node. The node decides to perform (m_1, k_2) and (m_1, k_4), and forwards (m_1, k_7) and (m_1, k_6) to its neighbors. (m_1, k_4) is sent directly to the processor queue, since k_4 is already stored in the cache. Since k_2 is not available in the cache, request (m_1, k_2) is put in the $PCR(k_2)$ queue, and a data interest packet for k_2 is generated and forwarded. At this time, k_3 arrives to the node, and the node sends all pending computation requests for k_3 (e.g., (m_1, k_3)) to the processor.

option to cache them for future use. A graphical overview of the described network is shown in Figure 1. A graphical representation of the procedures discussed above is shown in Figure 2. For brevity, we do not discuss the implementation detail and the format of computation and data interest packets. For more information on how these packets are formatted, we refer the interested reader to the works on named-data networking and named-computations [47], [48].

Several challenges need to be addressed in this setting. These include how to forward the computation and data interest packets, how to decide on performing computations and caching, and how to make these decisions in a distributed and scalable manner. In the next section, we present the DECO framework which presents a comprehensive solution to this joint problem.

IV. DECO Framework

In this section we introduce the DECO framework as a solution for joint computation scheduling, caching, and request forwarding in the computing network discussed in Section III. We start by analyzing an idealized setting where nodes are allowed to compute the results and cache the data objects without waiting for the data objects to be fetched. We call this setting a "*genie-aided*" network. We discuss the dynamics through which interest packets are handled in genie-aided network. We then find an outer bound for the stability region of the data-intensive computing network by analyzing the stability region of the genie-aided network. We then propose a throughput optimal policy that can stabilize all the computation interest packet queues and data interest packet queues for any computation request arrival rates inside the stability region of the genie-aided network. Since the throughput optimal policy cannot be directly implemented in the data-intensive computing network, we propose implementable versions of this policy which lead to a superior performance in terms of request satisfaction delay compared to a number of baseline schemes as shown in Section V.

A. Dynamics of Interest Packets in the Genie-Aided Network

In the genie-aided network, nodes are allowed to compute the results and cache the data objects without waiting for the data objects to be fetched. Consider time slots of length 1 second (without loss of generality) indexed by $t = 1, 2, \ldots, \infty$, where time slot t refers to the interval $[t, t+1)$. Each node $n \in \mathcal{V}$ keeps a separate queue for computation interest packets corresponding to the request $(m, k) \in \mathcal{R}$. The count of this queue at the beginning of time slot t is denoted by $Y_n^{(m,k)}(t)$. Each node also keeps a separate queue for data interest packets corresponding to data object $k \in \mathcal{D}$, and its count is denoted by $V_n^k(t)$.[5] The packets in each queue are served on a first-come-first-served basis. Initially all queues are empty, i.e., $Y_n^{(m,k)}(1) = V_n^k(1) = 0$ for all n, m, k. For each computation request (m, k) entering the network, the count $Y_n^{(m,k)}$ is increased accordingly at the entry nodes. Nodes decrease their computation interest counts by performing the corresponding computations and decrease their data interest counts by caching the corresponding data object. Nodes can also decrease their computation and data interest counts by forwarding them to their neighbors. On the other hand, performing a computation request in a node may result in an interest for the required data object in that node. Let $A_n^{(m,k)}(t)$ be the number of exogenous computation request

[5]Note that in genie-aided networks, nodes are allowed to compute the results and cache the data objects without waiting for the data objects to be fetched. Here the purpose behind creating data interest packets and keeping track of their counts is to capture the measured demands for the corresponding data objects.

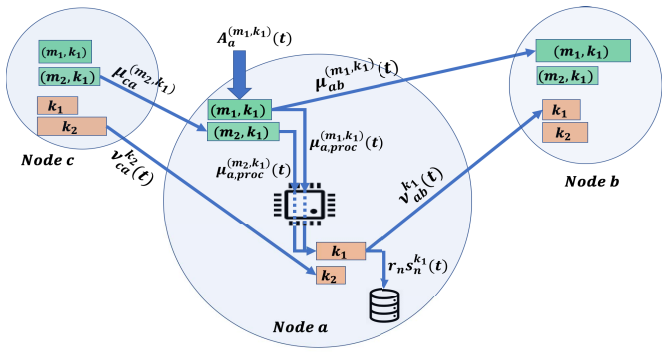


Fig. 3. Interest packet dynamics at node a. Computation interests queues and data interests queues evolve according to (1a)-(1b).

arrivals at node n for computation (m,k) during time slot t. For every computation request (m,k) arriving at node n, a corresponding computation interest packet for (m,k) is generated at n (i.e., $Y_n^{(m,k)}(t)$ incremented by 1). The long-term exogenous arrival rate at node n for computation (m,k) is $\lambda_n^{(m,k)} \triangleq \lim_{t\to\infty} \frac{1}{t}\sum_{\tau=1}^{t} A_n^{(m,k)}(\tau)$.

Let $\mu_{ab}^{(m,k)}(t)$ be the allocated transmission rate of computation interest packets (in number of interest packets per second) for (m,k) on link (a,b) during time slot t. Also, let $\nu_{ab}^{k}(t)$ be the allocated transmission rate of data interests (in number of interest packets per second) for data object k on link (a,b) during time slot t. We denote the allocated processing rate for computation (m,k) at node n during time slot t by $\mu_{n,proc}^{(m,k)}(t)$ (in number of computation interest per second). We assume that for each performed computation on data object k, a data interest packet for object k is generated. Let $s_n^k(t) \in \{0,1\}$ represent the caching state for object k at node n during slot t, where $s_n^k(t)=1$ if object k is cached at node n during slot t, and $s_n^k(t)=0$ otherwise. Note that even if $s_n^k(t)=1$, the cache at node n can satisfy only a limited number of interests during one time slot. This is because there is a maximum rate r_n (in data objects per second) at which node n can produce copies of cached object k. These dynamics can be written as

$$Y_n^{(m,k)}(t+1) \le \left(Y_n^{(m,k)}(t) - \mu_{n,proc}^{(m,k)}(t) - \sum_{b\in\mathcal{V}} \mu_{nb}^{(m,k)}(t)\right)^+ + \sum_{a\in\mathcal{V}} \mu_{an}^{(m,k)}(t) + A_n^{(m,k)}(t), \tag{1a}$$

$$V_n^{k}(t+1) \le \left(V_n^{k}(t) - r_n s_n^k(t) - \sum_{b\in\mathcal{V}} \nu_{nb}^{k}(t)\right)^+ + \sum_{m\in\mathcal{F}} \mu_{n,proc}^{(m,k)}(t) + \sum_{a\in\mathcal{V}} \nu_{an}^{k}(t), \tag{1b}$$

where $(x)^+ \triangleq max\{x,0\}$. Also, $V_{src(k)}^k(t)=0$ for all $t \ge 1$ and $Y_{src(k)}^{(m,k)}(t)=0$ for all $m\in\mathcal{F}$ and all $t\ge 1$. A graphical representation of these dynamics is shown in Figure 3.

From (1a) and (1b), we can see that the computation interests for (m,k) are processed at rate $\mu_{n,proc}^{(m,k)}(t)$, and data interests for data object k are decreased at rate r_n if node n decides to cache the data object k ($s_n^k(t)=1$). The remaining computation interests or data interests are transmitted to the neighbors with rate $\sum_b \mu_{nb}^{(m,k)}(t)$ and $\sum_b \nu_{nb}^{k}(t)$, respectively. The exogenous arrivals $A_n^{(m,k)}(t)$ and endogenous arrivals $\sum_a \mu_{an}^{(m,k)}(t)$ during time slot t are added to the computation interests queue at the end of time slot. The number of computation interests processed corresponding to data object k and the endogenous arrivals $\sum_a \nu_{an}^{k}(t)$ during time slot t are added to the data interest queue at the end of time slot. Note that (1a) is an inequality since the number of computation interests for (m,k) arriving to node n during slot t may be less than $\sum_a \mu_{an}^{(m,k)}(t)$, if the neighboring nodes have little or no computation interests to transmit. Also, (1b) is inequality because the number of data interests for object k arriving to node n during slot t may be less than $\sum_b \nu_{nb}^{k}(t)$, and the number of data interests created due to the processing of computation interests might be less than $\sum_m \mu_{n,proc}^{(m,k)}(t)$, if node n has little or no computation interests to process.

B. Stability Region for Genie-Aided System

In this section we discuss the constraints on processing rates, transmission rates, and caches, and provide the stability region for genie-aided system. We assume

- Exogenous computation request arrival processes $\{A_n^{(m,k)}(t); t=1,2,\dots\}$ are mutually independent with respect to $n,(m,k)$.
- $\{A_n^{(m,k)}(t); t=1,2,\dots\}$ are i.i.d with respect to t, for all $n\in\mathcal{V},(m,k)\in\mathcal{R}$.
- For all $n\in\mathcal{V},(m,k)\in\mathcal{R}$, $A_n^{(m,k)}(t) \le A_{n,max}^{(m,k)}$ for all t, where $A_{n,max}^{(m,k)} \in \mathbb{R}_+$.

During each time slot, a node cannot store more than its cache capacity, and cannot process computation requests more than processor capacity, i.e.,

$$\sum_{k\in\mathcal{D}} L_k s_n^k(t) \le C_n, \quad \forall n\in\mathcal{V}, \tag{2}$$

$$\sum_{(m,k)\in\mathcal{R}} q_{(m,k)} \mu_{n,proc}^{(m,k)}(t) \le P_n, \quad \forall n\in\mathcal{V}, \tag{3}$$

For each computation interest packet sent on a link, a result comes back on the reverse link eventually, and for each data interest packet sent on a link, a data object traverses back on the reverse link. Since the size of interest packets are negligible compared to results and data objects, when sending interest packets on a link (a,b) we need to take into account the reverse link capacity. Hence,

$$\sum_{k\in\mathcal{D}} L_k \nu_{ab}^{k}(t) + \sum_{(m,k)} Z_{(m,k)} \mu_{ab}^{(m,k)}(t) \le C_{ba}, \quad \forall (a,b)\in\mathcal{E}. \tag{4}$$

Before introducing the stability region, we first need to define the stability of the queues.

Definition 1: Stability for computation interest and data interest queues at node n is defined as

$$\limsup_{t\to\infty} \frac{1}{t}\sum_{\tau=1}^{t} 1_{[Y_n^{(m,k)}(\tau)>\xi]} \to 0, \ as\ \xi\to\infty, \tag{5}$$

$$\limsup_{t\to\infty} \frac{1}{t}\sum_{\tau=1}^{t} 1_{[V_n^{k}(\tau)>\xi]} \to 0, \ as\ \xi\to\infty, \tag{6}$$

where $1_{\{.\}}$ is the indicator function.

The stability region Λ is the closure of the set of all computation arrival rates, defined as $\lambda_n^{(m,k)} \triangleq \lim_{t\to\infty} \frac{1}{t}\sum_{\tau=1}^{t} A_n^{(m,k)}(\tau)$, for which there exists some feasible computation scheduling, caching, and forwarding policy which can stabilize all computation interest and data interest queues. By feasible, we mean that at any time t the caching vectors $\left(s_n^k(t)\right)_{k\in\mathcal{D},n\in\mathcal{V}}$ satisfy (2), the processing rate vector $\left(\mu_{n,proc}^{(m,k)}\right)_{(m,k)\in\mathcal{R},n\in\mathcal{V}}$ satisfy (3), and the forwarding rate vector $\left(\mu_{ab}^{(m,k)}\right)_{(m,k)\in\mathcal{R},(a,b)\in\mathcal{E}}$ satisfy (4). The following theorem characterizes the stability region of genie-aided.

Theorem 1: The stability region for computation and data interests of the genie-aided network $\mathcal{G}(\mathcal{V},\mathcal{E})$ with caching, computation, and transmission capacity constraints given by (2)-(3)-(4) and queue dynamics (1a)-(1b) is the set Λ consisting of all computation request arrival rates $\lambda_n^{(m,k)}$, such that there exists computation flow variables $(f_{ab}^{(m,k)})_{(m,k)\in\mathcal{R},(a,b)\in\mathcal{L}}$, data flow variables $(d_{ab}^k)_{k\in\mathcal{D},(a,b)\in\mathcal{L}}$, processing flow variables $(f_{n,proc}^{(m,k)})_{(m,k)\in\mathcal{R},n\in\mathcal{N}}$, and caching variables $(\beta_{n,i})_{n\in\mathcal{N};i\in\Psi_n}$ satisfying

$$f_{ab}^{(m,k)} \geq 0,\ f_{nn}^{(m,k)} = 0,\ f_{src(k)n}^{(m,k)} = 0, \quad \forall a,b,n \in \mathcal{N}, (m,k)\in\mathcal{R}, \tag{7a}$$

$$f_{ab}^{(m,k)} = 0, \quad \forall a,b \in \mathcal{N}, (m,k)\in\mathcal{R}, (a,b)\notin\mathcal{L}, \tag{7b}$$

$$d_{ab}^k \geq 0,\ d_{nn}^k = 0,\ d_{src(k)n}^k, \quad \forall a,b,n\in\mathcal{N}, k\in\mathcal{D}, \tag{7c}$$

$$d_{ab}^k = 0, \quad \forall a,b\in\mathcal{N}, k\in\mathcal{D}, (a,b)\notin\mathcal{L}, \tag{7d}$$

$$0 \leq \beta_{n,i} \leq 1, \quad i\in\Psi_n, \tag{7e}$$

$$f_{n,proc}^{(m,k)} \geq 0, \quad \forall n\in\mathcal{N}, (m,k)\in\mathcal{R}, \tag{7f}$$

$$\lambda_n^{(m,k)} \leq \sum_{b\in\mathcal{V}} f_{nb}^{(m,k)} - \sum_{a\in\mathcal{V}} f_{an}^{(m,k)} + f_{n,proc}^{(m,k)}, \quad \forall n\in\mathcal{N}, (m,k)\in\mathcal{R}, \tag{7g}$$

$$\sum_{a\in\mathcal{V}} d_{an}^k + \sum_m f_{n,proc}^{(m,k)} \leq \sum_{b\in\mathcal{V}} d_{nb}^k + r_n \sum_{i\in\Psi_n} \beta_{n,i}\mathbf{1}[k\in\mathcal{B}_{n,i}], \quad \forall n\in\mathcal{N}, (m,k)\in\mathcal{R}, \tag{7h}$$

$$\sum_{(m,k)\in\mathcal{R}} Z_{(m,k)} f_{ab}^{(m,k)} + \sum_{k\in\mathcal{D}} L_k d_{ab}^k \leq C_{ba}, \quad \forall (a,b)\in\mathcal{L}, \tag{7i}$$

$$\sum_{i\in\Psi_n} \beta_{n,i} = 1, \quad \forall n\in\mathcal{N}, \tag{7j}$$

$$\sum_{(m,k)\in\mathcal{R}} q_{(m,k)} f_{n,proc}^{(m,k)} \leq P_n, \quad \forall n\in\mathcal{N}. \tag{7k}$$

Here, Ψ_n is the set of feasible cache combination for node n. Theorem 1 is proved in Appendix A. We show that if the network is stabilized for arrival rates $\boldsymbol{\lambda}$ by a feasible computation scheduling, transmission, and caching policy, then the variables defined in Theorem 1 exists. Conversely, if such variables exist, then there is a randomized computation, transmission, and caching policy that stabilizes the network. To our knowledge, Theorem 1 is the first characterization of the stability region of a data-intensive computing network which incorporates the effect of computation, transmission, and caching jointly. In the genie-aided system we assume that the nodes have instant access to data. Hence the stability region defined in (7) is an outer bound for the stability region of the original data-intensive computing network.

C. Throughput Optimal Policy in Genie-Aided System

In this section we introduce a distributed policy based on Lyapunov drift minimization for throughput optimal decision making in genie-aided system. The drift minimization problem results in two different linear programming (LP) problems for allocating processing and transmission rates. It also involves solving a knapsack problem for caching. This is NP-hard in general, but can be solved using approximation techniques or dynamic programming at each node. In what follows we introduce the joint computation scheduling, transmission, and caching policy:

Algorithm 1: At the beginning of each time slot t, observe the counts for computation interests $(Y_n^{(m,k)}(t))_{n\in\mathcal{V},(m,k)\in\mathcal{R}}$ and data interests $(V_n^k(t))_{n\in\mathcal{V},k\in\mathcal{D}}$ and decide on processing, transmission, and caching in the sequence described below.

Computation Scheduling: *At each node n, choose processing rates of computation interests by solving the following LP*:

$$\text{maximize} \sum_{(m,k)\in\mathcal{R}} \mu_{n,proc}^{(m,k)}(t)\left(Y_n^{(m,k)}(t) - V_n^k(t)\right) \tag{8a}$$

$$\text{subject to} \sum_{(m,k)\in\mathcal{R}} q_{(m,k)}\mu_{n,proc}^{(m,k)}(t) \leq P_n. \tag{8b}$$

Transmission: *At each node n, choose transmission rate of computation interests and data interests on each outgoing link $(n,b)\in\mathcal{E}$ by solving the following LP*:

$$\text{maximize} \sum_{(m,k)\in\mathcal{R}} \mu_{nb}^{(m,k)}(t)\left(Y_n^{(m,k)}(t) - Y_b^{(m,k)}(t)\right) + \sum_{k\in\mathcal{D}} \nu_{nb}^k(t)\left(V_n^k(t) - V_b^k(t)\right) \tag{9a}$$

$$\text{subject to} \sum_{k\in\mathcal{D}} L_k\nu_{nb}^k(t) + \sum_{(m,k)\in\mathcal{R}} Z_{(m,k)}\mu_{nb}^{(m,k)}(t) \leq C_{bn}. \tag{9b}$$

Caching: *At each node n, choose caching variables by solving the following knapsack problem*:

$$\text{maximize} \sum_{k\in\mathcal{D}} V_n^k(t)s_n^k(t) \tag{10a}$$

$$\text{subject to} \sum_{k\in\mathcal{D}} L_k s_n^k(t) \leq C_n. \tag{10b}$$

Being distributed is an important characteristic of Algorithm 1. Computation scheduling and caching decisions are made at each node separately, and each node needs to exchange the size of computation interest and data interest queues only with its own neighbors. In the following theorem, we show that Algorithm 1 stabilizes all computation interest queues and data interest queues in the network for any $\boldsymbol{\lambda} \in int(\Lambda)$, without knowledge of $\boldsymbol{\lambda}$. As a result, Algorithm 1 is throughput optimal in the sense of adaptively maximizing the throughput of computation interests. The proof of Theorem 2 is provided in Appendix B.

Theorem 2 (Throughput Optimality): If there exists $\boldsymbol{\epsilon} = (\epsilon_n^{(m,k)})_{n\in\mathcal{V},(m,k)\in\mathcal{R}} \succ \boldsymbol{0}$ *such that* $\boldsymbol{\lambda}+\boldsymbol{\epsilon}\in\Lambda$, *then there exists* $(\epsilon_n^{(m,k)'})_{n\in\mathcal{V},(m,k)\in\mathcal{R}}, (\epsilon_n^{k'})_{n\in\mathcal{V},k\in\mathcal{D}} \succ \boldsymbol{0}$ *such that the network of interest queues under Algorithm 1 satisfies*:

$$\limsup_{t\to\infty}\frac{1}{t}\sum_{\tau=1}^{t}\Big(\sum_{n,(m,k)}E[Y_n^{(m,k)}(\tau)]+\sum_{n,k}E[V_n^k(\tau)]\Big)\leq\frac{NB}{\epsilon'}, \tag{11}$$

where $B \triangleq \frac{1}{2N}\sum_{n\in\mathcal{V}}(\mu_{n,out}^{max}+\mu_{n,proc}^{max}+r_n^{max})^2+(\mu_{n,in}^{max}+\mu_{n,proc}^{max}+A_n^{max})^2$, $\epsilon' \triangleq min\{(\epsilon_n^{(m,k)'})_{n\in\mathcal{V},(m,k)\in\mathcal{R}}, (\epsilon_n^{k'})_{n\in\mathcal{V},k\in\mathcal{D}}\}$ $\mu_{n,proc}^{max} \triangleq \frac{P_n}{min\{q_{(m,k)}\}}$, $\mu_{n,out}^{max} \triangleq \frac{\sum_b C_{bn}}{min\{Z_{(m,k)},L_k\}}$, $\mu_{n,in}^{max} \triangleq \frac{\sum_a C_{na}}{min\{Z_{(m,k)},L_k\}}$, $r_n^{max} \triangleq r_n|\mathcal{D}|$, $A_n^{max} \triangleq \sum_{(m,k)} A_{n,max}^{(m,k)}$, *and* $(\epsilon_n^{(m,k)'})_{n\in\mathcal{V},(m,k)\in\mathcal{R}}, (\epsilon_n^{k'})_{n\in\mathcal{V},k\in\mathcal{D}}$ *are variables depending on network topology and parameters.*

It is worth noting that in a network with equal-sized data, the knapsack caching problem (10) reduces to a max-weight problem which is solvable in linear time at each node. In a scenario where computation result sizes are also equal, the LP for transmission rates (9) turns into a backpressure algorithm on each link. Finally in a scenario where computation loads are equal, the LP problem for processing rates (8) turns into a backpressure-like algorithm between computation interest and data interest queues at each node. Consider a network where all data sizes are equal ($L_k = L, \quad \forall k\in\mathcal{D}$), all result sizes are equal ($Z_{(m,k)} = Z, \quad \forall (m,k)\in\mathcal{R}$) and all computation loads are equal ($q_{(m,k)} = q, \quad \forall (m,k)\in\mathcal{R}$). In this situation, Algorithm 1 reduces to the following simple backpressure and sorting algorithm which we call Algorithm 2.

Algorithm 2: In a network with $L_k = L$, $Z_{(m,k)} = Z$, $q_{(m,k)} = q$, *at the beginning of each time slot* t, *observe the counts for computation interests* $(Y_n^{(m,k)}(t))_{n\in\mathcal{V},(m,k)\in\mathcal{R}}$ *and data interests* $(V_n^k(t))_{n\in\mathcal{V},k\in\mathcal{D}}$ *and decide on computation scheduling, transmission, and caching in the sequence described below.*

Computation Scheduling: *At each node* n, *for each* $(m,k)\in\mathcal{R}$, *choose*:

$$\mu_{n,proc}^{(m,k)}(t)=\begin{cases}\frac{P_n}{q} & W_{n,proc}^*(t)>0, \quad (m,k)=(m,k)^*\\ 0 & \textit{otherwise.}\end{cases} \tag{12}$$

where

$$W_{n,proc}^{(m,k)}(t) \triangleq Y_n^{(m,k)}(t)-V_n^k(t) \tag{13a}$$

$$(m,k)^* \triangleq \underset{(m,k)}{argmax} \; W_{n,proc}^{(m,k)}(t) \tag{13b}$$

$$W_{n,proc}^*(t) \triangleq (W_{n,proc}^{(m,k)^*}(t))^+ \tag{13c}$$

Transmission: *At each node* n, *for each* $(m,k)\in\mathcal{R}$ *and each* $k\in\mathcal{D}$, *choose*:

$$\mu_{nb}^{(m,k)}(t)=\begin{cases}\frac{c_{bn}}{Z} & \frac{W_{nb}^*(t)}{Z}\geq\frac{G_{nb}^*(t)}{L}, W_{nb}^*(t)>0,\\ & m,k)=(m,k)^{**}\\ 0 & \textit{otherwise,}\end{cases} \tag{14a}$$

$$\nu_{nb}^k(t)=\begin{cases}\frac{c_{bn}}{L} & \frac{G_{nb}^*(t)}{L}>\frac{W_{nb}^*(t)}{Z}, G_{nb}^*(t)>0, k=k^*\\ 0 & \textit{otherwise,}\end{cases} \tag{14b}$$

where

$$W_{nb}^{(m,k)}(t) \triangleq Y_n^{(m,k)}(t)-Y_b^{(m,k)}(t), G_{nb}^k(t) \triangleq V_n^k(t)-V_b^k(t), \tag{15a}$$

$$(m,k)^{**} \triangleq \underset{(m,k)}{argmax} \; W_{nb}^{(m,k)}(t), \quad k^* \triangleq \underset{k}{argmax} \; G_{nb}^k(t), \tag{15b}$$

$$W_{nb}^*(t) \triangleq (W_{nb}^{(m,k)^*}(t))^+, \quad G_{nb}^*(t) \triangleq (G_{nb}^{k^*}(t))^+. \tag{15c}$$

Caching: *at each node* n, *let* $(d_1,d_2,\ldots,d_K)$ *be a permutation of* $\mathcal{D}$ *such that* $V_n^{d_1}\geq V_n^{d_2}\geq\cdots\geq V_n^{d_K}$. *Let* $i_n=\lfloor\frac{C_n}{L}\rfloor$, *then choose*

$$s_n^k(t)=\begin{cases}1 & k\in\{k_1,k_2,\ldots,k_{i_n}\}\\ 0 & \textit{otherwise.}\end{cases} \tag{16}$$

In Algorithm 2, each node n at each time slot t allocates the entire normalized processor capacity ($\frac{P_n}{q}$) to process the computation interests of $(m,k)^*$ which has the maximum difference between its computation interest count and data interest count for the required data object k as shown in (12)-(13). The intuition behind this is the following. The optimal policy allocates the processing capacity to the computation request for which there is relatively high local computation demand (i.e., large computation interest count) and low local demand for the required data object (i.e., low data interest count, often due to the data object being cached or permanently stored in close vicinity).

At each time t, each node n chooses for transmission on outgoing link $(n,b)\in\mathcal{E}$ the computation interest or data interest that has the maximum backlog difference on the link normalized by size (Z for computation interests and L for data interests), and allocates the entire normalized reverse link capacity (normalized by Z if the chosen count is a computation interest and by L if the chosen count is a data interest) to it, as shown in (14)-(15). For caching, each node n with capacity to cache $i_n=\lfloor\frac{C_n}{L}\rfloor$ data objects, chooses the i_n data objects with the highest data interest counts to be cached. We note that in Algorithm 2, at each node the computational complexity is $O(|\mathcal{F}|\times|\mathcal{D}|)$ for computation scheduling policy, $O(|\mathcal{V}|\times|\mathcal{F}|\times|\mathcal{D}|)$ for transmission policy, and $O(|\mathcal{D}|)$ for caching policy.

D. An Implementable Policy for Joint Computation, Caching, and Request Forwarding

Algorithm 1 cannot be implemented directly for the system introduced in Section III. As described in Section III and in Figure. 2, when a node decides to perform a computation request (m,k), it sends the computation to the processor if it is the source of data object k or has k stored in its cache. Otherwise it puts the computation request in the $PCR(k)$ queue and issues a data interest packet for k. When data object k returns to the node, that node sends all computation

requests in the $PCR(k)$ queue to the processor on a first-come-first-served basis. As for caching, nodes can only cache data objects when they are traversing back on the reverse path to the data requester. Our goal in this section is to design an implementable version of Algorithm 1. Although we do not prove theoretical optimality of the performance of algorithms discussed in this section and Section IV-E, they show excellent performance gains in the simulation experiments. At each time slot, each node decides on performing computation scheduling, request forwarding, and caching in the sequence described below.

1) Performing Computation Requests: At each time slot t, each node n performs computation requests of $(m,k) \in \mathcal{R}$ with rate $\mu_{n,proc}^{(m,k)^*}(t)$ where $\mu_{n,proc}^{(m,k)^*}(t)$ is the optimal processing rate at node n in time slot t obtained by solving (8a), (8b). That is, at each time slot t, each node n takes $\mu_{n,proc}^{(m,k)^*}(t)$ computation interest packets of type (m,k) out of its corresponding queue and sends them to the processor if $n = src(k)$ or has the data object k in its cache. Otherwise, it puts them in the $PCR(k)$ queue and generates $\mu_{n,proc}^{(m,k)^*}(t)$ data interest packets for data object k. When data object k reaches node n on the reverse path (of the data interest packet), node sends all the pending computation requests in the $PCR(k)$ queue to the processor on a first-come-first-served basis. The procedure is described as follows:

Input: $Y_n^{(m,k)}(t),\ V_n^k(t),\ q_{(m,k)},\ P_n, \forall (m,k) \in \mathcal{R},\ k \in \mathcal{D}$.

procedure PERFORMING COMPUTATION IN NODE n AT TIME t
  calculate $\mu_{n,proc}^{(m,k)^*}(t)$ by solving (8a), (8b).
  if data object k is stored at node n **then** send $\mu_{n,proc}^{(m,k)^*}(t)$ computation requests of type (m,k) from the corresponding queue to the processor queue in a first-come-first-served basis.
  else
    put $\mu_{n,proc}^{(m,k)^*}(t)$ computation requests of type (m,k) in $PCR(k)$ queue.
    issue $\mu_{n,proc}^{(m,k)^*}(t)$ data interest packets for fetching data object k.
  if data object k arrives at node n **then** put the computation requests in the $PCR(k)$ queue into the processor queue in a first-come-first-served order.

2) Transmission of Computation and Data Interest Packets: At each time slot t, each node n transmits $\mu_{nb}^{(m,k)^*}(t)$ computation interest packets of request (m,k), and transmits $\nu_{nb}^{k^*}(t)$ data interest packets of data object k on each outgoing link $(n,b) \in \mathcal{E}$, where $\mu_{nb}^{(m,k)^*}(t)$ and $\nu_{nb}^{k^*}(t)$ are optimal transmission rates at node n in time slot t obtained by solving (9a), (9b). The procedure is described below.

3) Caching Data Objects: A node can only cache data objects when the data objects are traversing back on the reverse path to the requester. We propose a caching policy based on the policy described in Algorithm 1 for the genie-aided system. That is, when new data objects arrive at the node, the combination of newly arrived and existing data objects that solves (10) is cached. The procedure is described as follows:

Input: $Y_n^{(m,k)}(t),\ V_n^k(t),\ L_k,\ Z_{(m,k)},\ C_{bn}, \forall (m,k) \in \mathcal{R},\ k \in \mathcal{D},\ (n,b) \in \mathcal{E}$.

procedure TRANSMISSION OF COMPUTATION AND DATA INTEREST PACKETS IN NODE n AT TIME t
  calculate transmission rate vectors $\boldsymbol{\mu}_{nb}^*(t)$ and $\boldsymbol{\nu}_{nb}^*(t)$ by solving (9a), (9b).
  transmit $\mu_{nb}^{(m,k)^*}(t)$ of computation interest packets of request (m,k) on each outgoing link $(n,b) \in \mathcal{E}$. If there is not enough computation interest packets in node n, transmit all of them.
  transmit $\nu_{nb}^{k^*}(t)$ of data interest packets of data object k on each outgoing link $(n,b) \in \mathcal{E}$. If there is not enough data interest packets in node n, it transmits all of them.

Input: $V_n^k(t),\ L_k,\ C_n$, for all $k \in \mathcal{D}$.

procedure CACHING DECISIONS IN NODE n AT TIME t
  when node n receives a data object k_{new}:
  if data object k is stored at node n **then** do nothing.
  else
    if cache of node n has sufficient space to store k_{new} **then** cache k_{new}.
    else
      let $\mathcal{K}_{n,old}$ be the set of objects that are currently cached at node n.
      cache a subset $\mathcal{K}_{n,new} \subseteq \mathcal{K}_{n,old} \cup k_{new}$ that solves (10).

Since the interest count for a data object is decremented by r_n immediately after node n caches the data object, we notice that this policy leads to oscillatory caching behaviour, whereby the cached data objects are shortly after removed from the cache again because other data objects' interest counts are now larger. In what follows we propose a method that has more stable caching behaviour.

E. Stable Caching Policy

In this section, we describe a policy that implements a stable solution in which the cache contents do not oscillate. The stable caching policy leads to significant performance gains in simulation experiments.

Within this policy, the performing of computation requests and transmission of packets coincide with the computation and forwarding scheme described in the implementable policy in Section IV-D. As for the caching policy, suppose that at time slot t, node n receives the data object k_{new} which is not currently cached at node n. If there is sufficient unused space in the cache of node n to accommodate k_{new}, then node n proceeds to cache the k_{new}. Otherwise, the node compares the cache scores (as calculated below) for k_{new} and the currently cached objects. For a given window size T, the cache score

TABLE II
EXPERIMENTAL PARAMETERS AND SETUP

	Abilene	Fog	GEANT	LHC
$\lvert\mathcal{F}\rvert$	100	200	100	100
$\lvert\mathcal{D}\rvert$	100	200	100	500
L_k	3GB	500MB	3GB	60GB
$Z_{(m,k)}$	300MB	50MB	1.5GB	6GB
$q_{(m,k)}$	5×10^4	5×10^4	5×10^4	10^5
Interest Packets' Size	60KB	10KB	60KB	60KB
Source Nodes	Seattle Sunnyvale Los Angles	S3	10, 11, ..., 21	MIT, WSC PRD, FNL VND, UFL NBR, UCSD
Requesting Nodes	Atlanta Washington New York	U1, U2, ..., U12	0, 1, ..., 9	MIT, WSC PRD, FNL VND, UFL NBR, UCSD

for object k at node n at time t is calculated as

$$CS_n^k(t) = \frac{1}{T} \sum_{\tau=t-T+1}^{t} \left[\sum_{(a,n)\in\mathcal{E}} \nu_{an}^k(\tau) + \sum_{m\in\mathcal{F}} \mu_{n,proc}^{(m,k)}(\tau) \right], \tag{17}$$

i.e., the average number of data interests for k received by node n over a window of size T prior to time slot t.

Let $\mathcal{K}_{n,old}$ be the set of objects that are currently cached at node n. Assuming all data objects are of equal size, let $k_{min} \in \mathcal{K}_{n,old}$ be the data object with the smallest cache score in $\mathcal{K}_{n,old}$. If k_{new} has a larger cache score than k_{min}, then object k_{min} is evicted and replaced with object k_{new}. Otherwise, the cache contents are unchanged. If objects have different sizes, the optimal set of objects is chosen to maximize the total cache score under the cache space constraint. This is a knapsack problem for which low complexity heuristics exist.

As a data object is cached at node n, the data interest count for that data object is decreased at the node. This attracts data interest packets for each object $k \in K_{n,new}$ to node n, where $K_{n,new}$ denotes the new set of cached objects. As a data object is evicted, data interest count for the evicted data object would begin to build up. As the count goes higher, the back-pressure forwarding policy would lead the data interests away from the node n to other parts of the network.

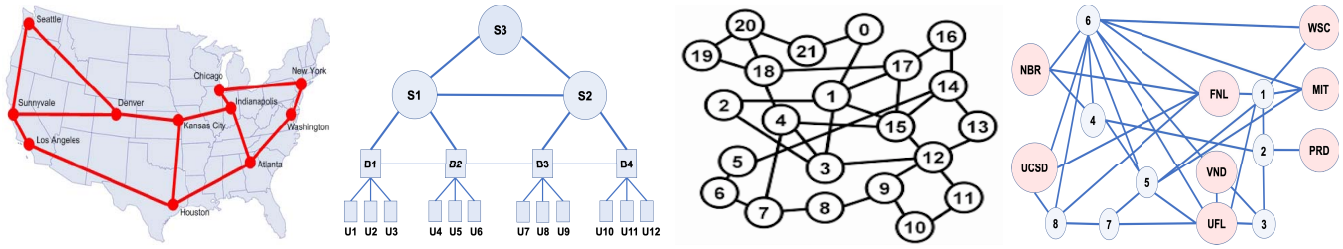
V. NUMERICAL EVALUATION

This section presents our experimental evaluation of the DECO framework. The simulations are performed on four different network topologies: Abilene topology shown in Figure 4(a), a fog computing topology shown in Figure 4(b), the GEANT topology shown in Figure 4(c), and the LHC topology, shown in Figure 4(d), for the Large Hadron Collider high energy physics network, one of the largest data-intensive computing networks in the world.

Experiment Setup. In Abilene, the cache capacity is 30GB and the processor capacity is 5×10^5 instructions/sec for all nodes. The link capacity (in both directions) is 240 Gbps for all links. In Fog, the cache capacity is 5GB for nodes $U1, U2, \ldots, U12$, 25GB for $B1$, $B2$, $B3$, $B4$, and 50GB for nodes $S1$, $S2$, $S3$. The processor capacity is 10^6 instructions/sec for $U1$, $U2$, ..., $U12$, 5×10^6 instructions/sec for $B1$, $B2$, $B3$, $B4$, and 10^7 instructions/sec for $S1$, $S2$, $S3$. The link capacity (in both directions) is 40 Gbps for the links between the bottom layer to the second layer $(U1, B1)$, $(U2, B1)$, ..., $(U11, B4)$, $(U12, B4)$, 200 Gbps for $(B1, B2)$, $(B2, B3)$, $(B3, B4)$, $(B1, S1)$, $(B2, S1)$, $(B3, S2)$, $(B4, S2)$, and 400 Gbps for $(S1, S2)$, $(S1, S3)$, $(S2, S3)$. In GEANT, the cache capacity is 30GB, and the processor capacity is 25×10^5 instructions/sec for all the nodes. The link capacity (in both directions) is 240 Gbps for all the links. In LHC, for "MIT", "WSC", "PRD", "FNL", "VND", "UFL", "NBR" and "UCSD", the cache capacity is 3TB and processing capacity is 3000, 5000, 5000, 2000, 1000, 1000, 3000, and 2000 instructions/sec, respectively. The cache and processor capacity are zero for all other nodes. The link capacity (in both directions) is 480 Mbps for all links. Other simulation parameters are shown in Table II for each topology. The designated source for each data object is chosen uniformly at random among the source nodes mentioned in Table II. At each requesting node, computation requests arrive according to a Poisson process with an overall rate λ (in request/node/sec). Each arriving request selects from the set of available tasks (independently) uniformly at random. In Abilene, Fog and GEANT topologies, we pair the ith computation task with ith data object to form a computation request. In LHC, the data objects are chosen according to a Zipf distribution with parameter 1, and pair the selected task and data to form a computation request.

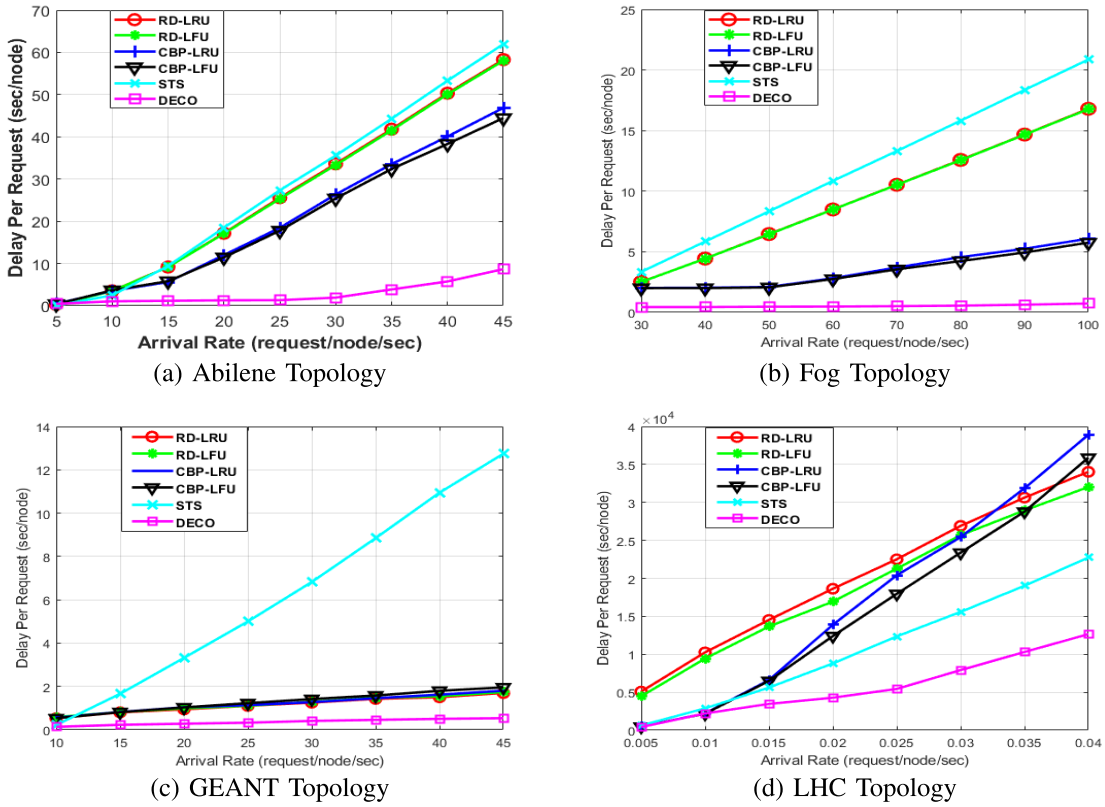
We calculate shortest paths from each node to the source for each data object, and populate the forwarding tables of the nodes with this information, beforehand. In all topologies, the buffers holding the computation interest packets, data interest packets, data packets, and result packets are assumed to have infinite size. Data packets and result packets share the same queue on the reverse paths, and are served on a first-come-first-served basis.

Policies and Measurements. We compare DECO with stable caching policy described in Section IV-E and five baseline policies in terms of computation request satisfaction delay. In the **RD-LRU** policy, *RD* stands for "Retrieve Data"; each computation request is performed at the entry node of the request, and if needed, a data interest packets is generated according to the procedure discussed already. All data interest packets in each node share one queue, and are forwarded to the source on a first-come-first-served basis. Each node caches the data objects when they travel back on the reverse path to the requesting node, if its cache is not full. If the cache is full, the nodes use *LRU* as the cache eviction policy. **RD-LFU** is similar to the RD-LRU policy but uses *LFU* as its cache eviction policy. In the **STS** policy, *STS* stands for "Send To Source"; each computation request (m, k) is forwarded to the source of the data object k. All computation requests share the same queue at each node and are forwarded on a first-come-first-served basis. When the computation requests reach the source, they are sent to the processor queue directly. Since



(a) Abilene Topology (b) Fog Topology (c) GEANT Topology (d) LHC Topology

Fig. 4. Network topologies.



(a) Abilene Topology (b) Fog Topology

(c) GEANT Topology (d) LHC Topology

Fig. 5. Computation request satisfaction delay.

the data objects do not move in this policy, there is no caching decision made by individual nodes. In the **CBP-LRU** policy, *CBP* stands for "Computation Backpressure". There is a separate queue for the computation interest packets of type (m, k) at each node. We use backpressure-based algorithms on the computation interest packets for performing computations and forwarding, similar to the approach introduced in [37]. Since all the result sizes and computation loads are equal, the policy performs the most backlogged computation request at each node. Also, the forwarding is done by simple backpressure on each outgoing link subject to the reverse link capacity normalized by the result size. The data interest packets all share the same queue and are forwarded on a first-come-first-served basis toward the sources of the data objects. Each node uses *LRU* as its cache eviction policy. **CBP-LFU** is similar to CBP-LRU, but uses *LFU*.

The simulator finest granularity *time step* is $2\mu sec$ for Abilene, Fog, and GEANT, and is $1msec$ for LHC. In DECO, decisions are carried out in the slots of length 10^4 time steps and the averaging window size is 10^6 time steps. In CBP-LRU and CBP-LFU, backpressure algorithms for performing computations and forwarding are carried out in the slots of length 10^4 time steps. The average window size in all policies that utilize LFU is 10^6 time steps. The simulator generates computation requests for 100 seconds in Abilene, Fog, and GEANT, and 50000 seconds in LHC. After generating requests, simulator waits for all computation requests to be satisfied. The delay of each computation request is calculated as the difference between the fulfillment time (i.e., time of arrival of the last result packet) and the creation time of the computation interest packet. We sum over all the delays, and normalize it by the total number of generated requests

and the number of requesting nodes. The computation request satisfaction delay (in second per request per node) is plotted for different arrival rates (in number of requests per node per second) for each topology in Figure 5.

We see that DECO outperforms all other schemes by a large margin. For instance, at arrival rate of $\lambda = 45$, DECO has around 80% delay improvement in Abilene and 90% delay improvement in GEANT compared to the policy with the closest performance.

VI. Conclusion

We address the problem of joint computation scheduling, caching, and request forwarding in a distributed data-intensive computing network where users issue requests for performing a computation task on a piece of data. With no prior knowledge on request rates, our framework utilizes computation interest counts and data interest counts to characterize the demand for computation and data. Although the policies discussed in this work are designed to increase the throughput, they lay the foundation for future research directions on minimizing the latency, cost, and prioritizing request deadlines. There are several methods proposed for minimizing the delay for back-pressure policies [49], [50]. Applying such methods to the policies discussed in this paper is a promising future research direction. In addition, cost and delay minimization in caching networks [51], [52] can be extended to data-intensive computation networks with arbitrary typologies using the computation and data model discussed in this paper.

Appendix A
Proof of Theorem 1

We need to show that $\boldsymbol{\lambda} \in \Lambda$ is necessary for stability and $\boldsymbol{\lambda} \in int(\Lambda)$ is a sufficient condition for stability. First we prove the necessary condition. Assume the network is stabilized for arrival rates $\boldsymbol{\lambda}$ by a feasible computation scheduling, transmission, and caching policy. Let $F_{ab}^{(m,k)}(t)$ be the amount of computation interests for (m,k) sent on link (a,b) over time slot t. Similarly, we define $D_{ab}^{k}(t)$ for data interests for object k sent on link (a,b) over time slot t. Also, we define $F_{n,proc}^{(m,k)}(t)$ to be the number of computation interests for (m,k) processed over time slot t. Hence,

$$F_{ab}^{(m,k)}(t) \geq 0,\ F_{nn}^{(m,k)}(t) = 0,\ F_{src(k)n}^{(m,k)}(t) = 0, \quad \forall a,b,n \in \mathcal{N}, (m,k) \in \mathcal{R}, \tag{18a}$$

$$F_{ab}^{(m,k)}(t) = 0, \quad \forall a,b \in \mathcal{N}, (m,k) \in \mathcal{R}, (a,b) \notin \mathcal{L}, \tag{18b}$$

$$D_{ab}^{k}(t) \geq 0,\ D_{nn}^{k}(t) = 0,\ D_{src(k)n}^{k}(t) = 0, \quad \forall a,b,n \in \mathcal{N}, k \in \mathcal{D}, \tag{18c}$$

$$D_{ab}^{k}(t) = 0, \quad \forall a,b \in \mathcal{N}, k \in \mathcal{D}, (a,b) \notin \mathcal{L}, \tag{18d}$$

$$F_{n,proc}^{(m,k)}(t) \geq 0, \quad \forall n \in \mathcal{N}, (m,k) \in \mathcal{R}, \tag{18e}$$

$$\sum_{(m,k)\in\mathcal{R}} Z_{(m,k)} F_{ab}^{(m,k)}(t) + \sum_{k\in\mathcal{D}} L_k D_{ab}^{k}(t) \leq C_{ba}, \quad \forall (a,b) \in \mathcal{L} \tag{18f}$$

$$\sum_{(m,k)\in\mathcal{R}} q_{(m,k)} F_{n,proc}^{(m,k)}(t) \leq P_n, \quad \forall n \in \mathcal{N}. \tag{18g}$$

By Lemma 6 of [53], if the network is stable, there exists a finite M such that $Y_n^{(m,k)}(t) \leq M$ and $V_n^k(t) \leq M$ for all n,m,k holds infinitely often. Given an arbitrarily small value $\epsilon > 0$, there exists a slot $\tilde{t}$ such that

$$Y_n^{(m,k)}(\tilde{t}) \leq M, V_n^k(\tilde{t}) \leq M, \frac{M}{\tilde{t}} \leq \epsilon, \tag{19}$$

$$\left|\frac{\sum_{\tau=1}^{\tilde{t}} A_n^k(\tau)}{\tilde{t}} - \lambda_n^{(m,k)}\right| \leq \epsilon. \tag{20}$$

Assuming all queues are initially empty, we can write the queue length as the difference between the interests that have arrived and departed (and drained):

$$Y_n^{(m,k)}(\tilde{t}) = \sum_{\tau=1}^{\tilde{t}} \sum_a F_{an}^{(m,k)}(\tau) + \sum_{\tau=1}^{\tilde{t}} A_n^{(m,k)}(\tau) \tag{21}$$

$$- \sum_{\tau=1}^{\tilde{t}} \sum_b F_{nb}^{(m,k)}(\tau) - \sum_{\tau=1}^{\tilde{t}} F_{n,proc}^{(m,k)}(\tau) \tag{22}$$

$$V_n^k(\tilde{t}) = \sum_{\tau=1}^{\tilde{t}} \sum_a D_{an}^k(\tau) + \sum_{\tau=1}^{\tilde{t}} \sum_m F_{n,proc}^{(m,k)}(\tau) \tag{23}$$

$$- \sum_{\tau=1}^{\tilde{t}} \sum_b D_{nb}^k(\tau) - r_n \sum_{\tau=1}^{\tilde{t}} s_n^k(\tau) \tag{24}$$

We know that $\frac{1}{\tilde{t}} \sum_{\tau=1}^{\tilde{t}} A_n^k(\tau) \geq \lambda_n^{(m,k)} - \epsilon$, $\frac{1}{\tilde{t}} Y_n^{(m,k)}(\tilde{t}) \leq \epsilon$, and $\frac{1}{\tilde{t}} V_n^k(\tilde{t}) \leq \epsilon$. Thus,

$$\epsilon \geq \frac{1}{\tilde{t}} \sum_{\tau=1}^{\tilde{t}} \sum_a F_{an}^{(m,k)}(\tau) + \lambda_n^{(m,k)} - \epsilon - \frac{1}{\tilde{t}} \sum_{\tau=1}^{\tilde{t}} \sum_b F_{nb}^{(m,k)}(\tau) - \frac{1}{\tilde{t}} \sum_{\tau=1}^{\tilde{t}} F_{n,proc}^{(m,k)}(\tau) \tag{25}$$

$$\epsilon \geq \frac{1}{\tilde{t}} \sum_{\tau=1}^{\tilde{t}} \sum_a D_{an}^k(\tau) + \frac{1}{\tilde{t}} \sum_{\tau=1}^{\tilde{t}} \sum_m F_{n,proc}^{(m,k)}(\tau) - \frac{1}{\tilde{t}} \sum_{\tau=1}^{\tilde{t}} \sum_b D_{nb}^k(\tau) - r_n \frac{1}{\tilde{t}} \sum_{\tau=1}^{\tilde{t}} s_n^k(\tau) \tag{26}$$

We define $f_{ab}^{(m,k)} \triangleq \frac{1}{\tilde{t}} \sum_{\tau=1}^{\tilde{t}} F_{ab}^{(m,k)}(\tau)$, $d_{ab}^k \triangleq \frac{1}{\tilde{t}} \sum_{\tau=1}^{\tilde{t}} D_{ab}^k(\tau)$, and $f_{n,proc}^{(m,k)} \triangleq \frac{1}{\tilde{t}} \sum_{\tau=1}^{\tilde{t}} F_{n,proc}^{(m,k)}(\tau)$. Then by (18a), (18b),(18c),(18d), (18e), (18f), and (18g) we can prove (7a), (7b),(7c),(7d),(7f), (7i), and (7k). Let $\mathcal{T}_{n,i} = \left\{ \tau \in \{1,\ldots,\tilde{t}\} : s_n^k(\tau) = 1\ \forall k \in \mathcal{B}_{n,i}, s_n^k(\tau) = 0\ \forall k \notin \mathcal{B}_{n,i} \right\}$, and $\beta_{n,i} = \frac{|\mathcal{T}_{n,i}|}{\tilde{t}}$. Thus, we can prove (7e) and (7j). Finally, from (25) and (26) we can write

$$\lambda_n^{(m,k)} \leq \sum_b f_{nb}^{(m,k)} - \sum_a f_{an}^{(m,k)} + f_{n,proc}^{(m,k)} + 2\epsilon, \tag{27}$$

$$\sum_a d_{an}^k + \sum_m f_{n,proc}^{(m,k)} \leq \sum_b d_{nb}^k + r_n \sum_{i\in\Psi_n} \beta_{n,i} \mathbf{1}[k \in \mathcal{B}_{n,i}] + \epsilon. \tag{28}$$

By letting $\epsilon \to 0$, we can prove (7g) and (7h).

Now we show that $\boldsymbol{\lambda} \in \text{int}(\Lambda)$ is a sufficient condition for stability. $\boldsymbol{\lambda} \in \text{int}(\Lambda)$ means that there exists

$\boldsymbol{\epsilon} = (\epsilon_n^{(m,k)})$, where $\epsilon_n^{(m,k)} > 0$ such that $\boldsymbol{\lambda} + \boldsymbol{\epsilon} \in \Lambda$. Let $(f_{ab}^{(m,k)}), (d_{ab}^k), (f_{n,proc}^{(m,k)})$, and $(\boldsymbol{\beta}_n)$ be the transmission, processing, and caching variables associated with $\boldsymbol{\lambda} + \boldsymbol{\epsilon}$. So all (7a), (7b),(7c),(7d),(7e),(7f), (7i),(7j), and (7k) are true for these values as well as

$$\lambda_n^{(m,k)} + \epsilon_n^{(m,k)} \leq \sum_b f_{nb}^{(m,k)} - \sum_a f_{an}^{(m,k)} + f_{n,proc}^{(m,k)}, \tag{29}$$

$$\sum_a d_{an}^k + \sum_m f_{n,proc}^{(m,k)} \leq \sum_b d_{nb}^k + r_n \sum_{i \in \Psi_n} \beta_{n,i} \mathbf{1}[k \in \mathcal{B}_{n,i}]. \tag{30}$$

First, we introduce new processing and transmission variables $f_{n,proc}^{(m,k)'}, f_{ab}^{(m,k)'}, d_{ab}^{k'}$ as defined in Appendix C. Then we propose a randomized policy for computation scheduling, transmission, and caching that makes service rates strictly larger than input rates in all computation and data interest queues, thus stabilizes all of them.

Algorithm 3: ***Computation Scheduling:*** *Choose a computation interest $(\tilde{m}, \tilde{k})$ which is chosen randomly to be (m,k) with probability $\frac{q_{(m,k)} f_{n,proc}^{(m,k)'}}{\sum_{m,k} q_{(m,k)} f_{n,proc}^{(m,k)'}}$. Then set $\tilde{\mu}_{n,proc}^{(m,k)} = \begin{cases} \frac{\sum_{m,k} q_{(m,k)} f_{n,proc}^{(m,k)'}}{q_{(m,k)}} & \text{if } (m,k) = (\tilde{m}, \tilde{k}) \\ 0 & \text{otherwise.} \end{cases}$*

Transmission: *For every link (a,b) such that $\sum_{m,k} f_{ab}^{(m,k)'} > 0$, pick a computation interest $(\tilde{m}, \tilde{k})$ which is chosen randomly to be (m,k) with probability $\frac{z_{(m,k)} f_{ab}^{(m,k)'}}{\sum_{m,k} z_{(m,k)} f_{ab}^{(m,k)'}}$. Then set $\tilde{\mu}_{ab}^{(m,k)} = \begin{cases} \frac{\sum_{m,k} z_{(m,k)} f_{ab}^{(m,k)'}}{z_{(m,k)}} & \text{if } (m,k) = (\tilde{m}, \tilde{k}) \\ 0 & \text{otherwise.} \end{cases}$ Also, For every link (a,b) such that $\sum_k d_{ab}^{k'} > 0$, pick a data interest $\tilde{k}$ which is chosen randomly to be k with probability $\frac{L_k d_{ab}^{k'}}{\sum_k L_k d_{ab}^{k'}}$. Then set $\tilde{\nu}_{ab}^k = \begin{cases} \frac{\sum_k L_k d_{ab}^{k'}}{L_k} & \text{if } k = \tilde{k} \\ 0 & \text{otherwise.} \end{cases}$*

Caching: *For every node n, cache the single combination $\tilde{\boldsymbol{\mathcal{B}}}_{\boldsymbol{n}}$ where $\tilde{\boldsymbol{\mathcal{B}}}_{\boldsymbol{n}}$ is chosen randomly to be $\mathcal{B}_{n,i}$ with probability $\beta_{n,i} / \sum_{i \in \Psi_n} \beta_{n,i}$. Then set $\tilde{s}_n^k(t) = \begin{cases} 1 \text{ if } k \in \tilde{\boldsymbol{\mathcal{B}}}_{\boldsymbol{n}} \\ 0 \text{ otherwise.} \end{cases}$*

Proposition 3: Algorithm 3 makes all service rates strictly larger than input rates in all computation and data interest queues, thus stabilizes all of them.

We prove Proposition 3 in Appendix D.

Appendix B
Proof of Theorem 2

We define the quadratic Lyapunov function as

$$\mathcal{L}\big(\boldsymbol{Y}(t), \boldsymbol{V}(t)\big) \triangleq \sum_{n,m,k} \big(Y_n^{(m,k)}(t)\big)^2 + \sum_{n,k} \big(V_n^k(t)\big)^2.$$

Hence, the Lyapunov drift at time slot t is defined by $\Delta(\boldsymbol{Y}(t), \boldsymbol{V}(t)) \triangleq \mathbb{E}\big[\mathcal{L}\big(\boldsymbol{Y}(t+1), \boldsymbol{V}(t+1)\big) - \mathcal{L}\big(\boldsymbol{Y}(t), \boldsymbol{V}(t)\big) | \boldsymbol{Y}(t), \boldsymbol{V}(t)\big]$. By taking square on both sides of (1a), (1b) we have

$$\begin{aligned}
&Y_n^{(m,k)}(t+1)^2 \\
&\leq \big(Y_n^{(m,k)}(t) - \sum_b \mu_{nb}^{(m,k)}(t) - \mu_{n,proc}^{(m,k)}(t)\big)^2 \\
&\quad + \big(\sum_a \mu_{an}^{(m,k)}(t) + A_n^{(m,k)}(t)\big)^2 + 2Y_n^{(m,k)}(t) \sum_a \mu_{an}^{(m,k)}(t) \\
&\quad + 2Y_n^{(m,k)}(t) A_n^{(m,k)}(t) \\
&\leq Y_n^{(m,k)}(t)^2 + \Big(\sum_b \mu_{nb}^{(m,k)}(t) + \mu_{n,proc}^{(m,k)}(t)\Big)^2 \\
&\quad - 2Y_n^{(m,k)}(t) \sum_b \mu_{nb}^{(m,k)}(t) - 2Y_n^{(m,k)}(t) \mu_{n,proc}^{(m,k)}(t) \\
&\quad + \big(\sum_a \mu_{an}^{(m,k)}(t) + A_n^{(m,k)}(t)\big)^2 + 2Y_n^{(m,k)}(t) \sum_a \mu_{an}^{(m,k)}(t) \\
&\quad + 2Y_n^{(m,k)}(t) A_n^{(m,k)}(t) \\
&V_n^k(t+1)^2 \\
&\leq \big(V_n^k(t) - \sum_b \nu_{nb}^k(t) - r_n s_n^k(t)\big)^2 \\
&\quad + \big(\sum_m \mu_{n,proc}^{(m,k)}(t) + \sum_a \nu_{an}^k(t)\big)^2 + 2V_n^k(t) \sum_m \mu_{n,proc}^{(m,k)}(t) \\
&\quad + 2V_n^k(t) \sum_a \nu_{an}^k(t) \\
&\leq V_n^k(t)^2 + \Big(\sum_b \nu_{nb}^k(t) + r_n s_n^k(t)\Big) - 2V_n^k(t) \sum_b \nu_{nb}^k(t) \\
&\quad - 2V_n^k(t) r_n s_n^k(t) + \Big(\sum_m \mu_{n,proc}^{(m,k)}(t) + \sum_a \nu_{an}^k(t)\Big)^2 \\
&\quad + 2V_n^k(t) \sum_m \mu_{n,proc}^{(m,k)}(t) + 2V_n^k(t) \sum_a \nu_{an}^k(t).
\end{aligned}$$

We remove the expectation and condition signs for simplicity, and we write Lyoponov drift as

$$\begin{aligned}
&\mathcal{L}(t+1) - \mathcal{L}(t) \\
&\leq \sum_{n,m,k} \big(\sum_b \mu_{nb}^{(m,k)}(t) + \mu_{n,proc}^{(m,k)}(t)\big)^2 \\
&\quad + \sum_{n,m,k} \Big(\sum_a \mu_{an}^{(m,k)}(t) + A_n^{(m,k)}(t)\Big)^2 \\
&\quad + \sum_{n,k} \Big(\sum_b \nu_{nb}^k(t) + r_n s_n^k(t)\Big)^2 \\
&\quad + \sum_{n,k} \Big(\sum_m \mu_{n,proc}^{(m,k)}(t) + \sum_a \nu_{an}^k(t)\Big)^2 \\
&\quad + \sum_{n,m,k} 2Y_n^{(m,k)}(t) A_n^{(m,k)}(t) \\
&\quad + 2 \sum_{n,m,k} \mu_{n,proc}^{(m,k)}(t) \Big(V_n^k(t) - Y_n^{(m,k)}(t)\Big) \\
&\quad - 2 \sum_{n,k} V_n^k(t) r_n s_n^k(t) \\
&\quad + 2 \sum_{(a,b)} \sum_{m,k} \mu_{ab}^{(m,k)}(t) \Big(Y_b^{(m,k)}(t) - Y_a^{(m,k)}(t)\Big) \\
&\quad + 2 \sum_{(a,b)} \sum_k \nu_{ab}^k(t) \Big(V_b^k(t) - V_a^k(t)\Big)
\end{aligned}$$

$$
\begin{aligned}
&\leq 2NB + \sum_{n,m,k} 2Y_n^{(m,k)}(t)A_n^{(m,k)}(t) \\
&\quad +2\sum_{n,m,k}\mu_{n,proc}^{(m,k)}(t)\Big(V_n^k(t) - Y_n^{(m,k)}(t)\Big) \\
&\quad -2\sum_{n,k} V_n^k(t) r_n s_n^k(t) \\
&\quad +2\sum_{(a,b)}\sum_{m,k}\mu_{ab}^{(m,k)}(t)\Big(Y_b^{(m,k)}(t) - Y_a^{(m,k)}(t)\Big) \\
&\quad +2\sum_{(a,b)}\sum_{k}\nu_{ab}^{k}(t)\Big(V_b^k(t) - V_a^k(t)\Big).
\end{aligned}
$$

Algorithm 1 minimizes the RHS subject to conditions in (2),(3),(4). So the policy that minimizes the drift is

$$
\begin{aligned}
&\text{maximize} \sum_{n,m,k}\mu_{n,proc}^{(m,k)}(t)\Big(Y_n^{(m,k)}(t) - V_n^k(t)\Big) \\
&\quad + \sum_{n,k} V_n^k(t) r_n s_n^k(t) \\
&\quad + \sum_{(a,b)}\sum_{m,k}\mu_{ab}^{(m,k)}(t)\Big(Y_a^{(m,k)}(t) - Y_b^{(m,k)}(t)\Big) \\
&\quad + \sum_{(a,b)}\sum_{k}\nu_{ab}^{k}(t)\Big(V_a^k(t) - V_b^k(t)\Big),
\end{aligned}
$$

subject to conditions in (2)-(3)-(4), which is exactly Algorithm 1. Thus,

$$
\begin{aligned}
&\Delta(\mathbf{Y}(t),\mathbf{V}(t)) \\
&\leq 2NB + \sum_{n,m,k} 2Y_n^{(m,k)}(t)\lambda_n^{(m,k)}(t) \\
&\quad -2E\Big\{\sum_{n,m,k}\mu_{n,proc}^{(m,k)}(t)\Big(Y_n^{(m,k)}(t) - V_n^k(t)\Big)|\mathbf{Y}(t),\mathbf{V}(t)\Big\} \\
&\quad -2E\Big\{\sum_{n,k} V_n^k(t) r_n s_n^k(t)|\mathbf{Y}(t),\mathbf{V}(t)\Big\} \\
&\quad -2E\Big\{\sum_{(a,b)}\sum_{m,k}\mu_{ab}^{(m,k)}(t) \times \\
&\qquad \Big(Y_a^{(m,k)}(t) - Y_b^{(m,k)}(t)\Big)|\mathbf{Y}(t),\mathbf{V}(t)\Big\} \\
&\quad -2E\Big\{\sum_{(a,b)}\sum_{k}\nu_{ab}^{k}(t)\Big(V_a^k(t) - V_b^k(t)\Big)|\mathbf{Y}(t),\mathbf{V}(t)\Big\}
\end{aligned}
$$

We know that Algorithm 1 has the smallest RHS among all policies including Algorithm 3 in Appendix A. As a result,

$$
\begin{aligned}
&\Delta(\mathbf{Y}(t),\mathbf{V}(t)) \\
&\leq 2NB + \sum_{n,m,k} 2Y_n^{(m,k)}(t)\lambda_n^{(m,k)}(t) \\
&\quad -2E\Big\{\sum_{n,m,k}\tilde{\mu}_{n,proc}^{(m,k)}(t)\Big(Y_n^{(m,k)}(t) - V_n^k(t)\Big)|\mathbf{Y}(t),\mathbf{V}(t)\Big\} \\
&\quad -2E\Big\{\sum_{n,k} V_n^k(t) r_n \tilde{s}_n^k(t)|\mathbf{Y}(t),\mathbf{V}(t)\Big\} \\
&\quad -2E\Big\{\sum_{(a,b)}\sum_{m,k}\tilde{\mu}_{ab}^{(m,k)}(t) \times \\
&\qquad \Big(Y_a^{(m,k)}(t) - Y_b^{(m,k)}(t)\Big)|\mathbf{Y}(t),\mathbf{V}(t)\Big\} \\
&\quad -2E\Big\{\sum_{(a,b)}\sum_{k}\tilde{\nu}_{ab}^{k}(t)\Big(V_a^k(t) - V_b^k(t)\Big)|\mathbf{Y}(t),\mathbf{V}(t)\Big\} \\
&\leq 2NB + \sum_{n,m,k} 2Y_n^{(m,k)}(t)\lambda_n^{(m,k)}(t) - 2\sum_{n,m,k} Y_n^{(m,k)}(t) \\
&\quad \times E\Big\{\sum_{b}\tilde{\mu}_{nb}^{(m,k)}(t) - \sum_{a}\tilde{\mu}_{an}^{(m,k)}(t) + \tilde{\mu}_{n,proc}^{(m,k)}(t)|\mathbf{Y}(t)\Big\} \\
&\quad -2\sum_{n,k} V_n^k(t) \times E\Big\{\sum_{b}\tilde{\nu}_{nb}^{k}(t) + r_n\tilde{s}_n^k(t) \\
&\quad -\sum_{a}\tilde{\nu}_{an}^{k}(t) - \sum_{m}\tilde{\mu}_{n,proc}^{(m,k)}(t)|\mathbf{V}(t)\Big\}
\end{aligned}
$$

Since in Algorithm 3 service rates are strictly larger than arrival rates, we have

$$
\begin{aligned}
&\leq 2NB + \sum_{n,m,k} 2Y_n^{(m,k)}(t)\lambda_n^{(m,k)}(t) \\
&\quad -2\sum_{n,m,k} Y_n^{(m,k)}(t)\Big\{\lambda_n^{(m,k)}(t) + \epsilon_n^{(m,k)'}\Big\} - 2\sum_{n} V_n^k(t)\epsilon_n^{k'}
\end{aligned}
$$

Therefore,

$$
\Delta(\mathbf{Y}(t),\mathbf{V}(t)) \leq 2NB - 2\epsilon'\sum_{n,m,k} Y_n^{(m,k)}(t) - 2\epsilon'\sum_{n,k} V_n^k(t),
$$

where $\epsilon' \triangleq min\{(\epsilon_n^{(m,k)'})_{n\in\mathcal{V},(m,k)\in\mathcal{R}}, (\epsilon_n^{k'})_{n\in\mathcal{V},k\in\mathcal{D}}\}$. By lemma 4.1 in [54], we conclude the proof.

REFERENCES

[1] K. Kamran, E. Yeh, and Q. Ma, "DECO: Joint computation, caching and forwarding in data-centric computing networks," in *Proc. 20th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, Jul. 2019, pp. 111–120, doi: 10.1145/3323679.3326509.

[2] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[3] M. V. Barbera, S. Kosta, A. Mei, and J. Stefa, "To offload or not to offload? The bandwidth and energy costs of mobile cloud computing," in *Proc. IEEE INFOCOM*, Apr. 2013, pp. 1285–1293.

[4] J. Y. Chung, C. Joe-Wong, S. Ha, J. W.-K. Hong, and M. Chiang, "CYRUS: Towards client-defined cloud storage," in *Proc. 10th Eur. Conf. Comput. Syst.*, Apr. 2015, Art. no. 17.

[5] E. Cuervo *et al.*, "MAUI: Making smartphones last longer with code offload," in *Proc. 8th Int. Conf. Mobile Syst., Appl., Services (MobiSys)*, 2010, pp. 49–62.

[6] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 945–953.

[7] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "CloneCloud: Elastic execution between mobile device and cloud," in *Proc. 6th Conf. Comput. Syst.*, 2011, pp. 301–314.

[8] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, "Cuckoo: A computation offloading framework for smartphones," in *Proc. Int. Conf. Mobile Comput., Appl., Services*. Berlin, Germany: Springer, 2010, pp. 59–79.

[9] Y. Wen, W. Zhang, and H. Luo, "Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones," in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 2716–2720.

[10] L. Yang, J. Cao, S. Tang, T. Li, and A. T. S. Chan, "A framework for partitioning and execution of data stream applications in mobile cloud computing," in *Proc. IEEE 5th Int. Conf. Cloud Comput.*, Jun. 2012, pp. 794–802.

[11] S. Sarkar, S. Chatterjee, and S. Misra, "Assessment of the suitability of fog computing in the context of Internet of Things," *IEEE Trans. Cloud Comput.*, vol. 6, no. 1, pp. 46–59, Jan./Mar. 2018.

[12] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. 1st Workshop Mobile Cloud Comput. (MCC)*, 2012, pp. 13–16.

[13] M. Chen, Y. Qian, Y. Hao, Y. Li, and J. Song, "Data-driven computing and caching in 5G networks: Architecture and delay analysis," *IEEE Wireless Commun.*, vol. 25, no. 1, pp. 70–75, Feb. 2018.

[14] A. A. Abdellatif, A. Mohamed, C. F. Chiasserini, M. Tlili, and A. Erbad, "Edge computing for smart health: Context-aware approaches, opportunities, and challenges," *IEEE Netw.*, vol. 33, no. 3, pp. 196–203, May/Jun. 2019.

[15] A. A. Mutlag, M. K. A. Ghani, N. Arunkumar, M. A. Mohammed, and O. Mohd, "Enabling technologies for fog computing in healthcare IoT systems," *Future Gener. Comput. Syst.*, vol. 90, pp. 62–78, Jan. 2019.

[16] H. Li, K. Ota, and M. Dong, "Learning IoT in edge: Deep learning for the Internet of Things with edge computing," *IEEE Netw.*, vol. 32, no. 1, pp. 96–101, Jan./Feb. 2018.

[17] X. Chen, Q. Shi, L. Yang, and J. Xu, "Thriftyedge: Resource-efficient edge computing for intelligent IoT applications," *IEEE Netw.*, vol. 32, no. 1, pp. 61–65, Jan./Feb. 2018.

[18] G. Zhu, D. Liu, Y. Du, C. You, J. Zhang, and K. Huang, "Toward an intelligent edge: Wireless communication meets machine learning," *IEEE Commun. Mag.*, vol. 58, no. 1, pp. 19–25, Jan. 2020.

[19] P. O'Donovan, C. Gallagher, K. Bruton, and D. T. J. O'Sullivan, "A fog computing industrial cyber-physical system for embedded low-latency machine learning Industry 4.0 applications," *Manuf. Lett.*, vol. 15, pp. 139–142, Jan. 2018.

[20] S. A. Marklouf and B. Yagoubi, "Data-aware scheduling strategy for scientific workflow applications in IaaS cloud computing," *Int. J. Interact. Multimedia Artif. Intell.*, vol. 5, no. 4, pp. 75–85, 2019.

[21] H. Dubey, J. Yang, N. Constant, A. M. Amiri, Q. Yang, and K. Makodiya, "Fog data: Enhancing telehealth big data through fog computing," in *Proc. ASE BigData*, 2015, Art. no. 14.

[22] J. Zhang, F.-Y. Wang, K. Wang, W.-H. Lin, X. Xu, and C. Chen, "Data-driven intelligent transportation systems: A survey," *IEEE Trans. Intell. Transp. Syst.*, vol. 12, no. 4, pp. 1624–1639, Dec. 2011.

[23] C. Long *et al.*, "Edge computing framework for cooperative video processing in multimedia IoT systems," *IEEE Trans. Multimedia*, vol. 20, no. 5, pp. 1126–1139, May 2018.

[24] A. Karimian, Z. Yang, and R. Tron, "Rotational outlier identification in pose graphs using dual decomposition," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2020, pp. 391–407.

[25] A. Karimian and R. Tron, "Bearing-only consensus and formation control under directed topologies," in *Proc. Amer. Control Conf. (ACC)*, Jul. 2020, pp. 3503–3510.

[26] E. Yeh, T. Ho, Y. Cui, M. Burd, R. Liu, and D. Leong, "VIP: A framework for joint dynamic forwarding and caching in named data networks," in *Proc. 1st Int. Conf. Inf.-Centric Netw. (INC)*, 2014, pp. 117–126.

[27] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.

[28] E. Deelman *et al.*, "Pegasus: Mapping scientific workflows onto the grid," in *Grid Computing*, M. D. Dikaiakos, Ed. Berlin, Germany: Springer, 2004, pp. 11–20.

[29] A. Knezevic *et al.*, "CIRCE—A runtime scheduler for DAG-based dispersed computing: Demo," in *Proc. 2nd ACM/IEEE Symp. Edge Comput.*, Oct. 2017, Art. no. 31.

[30] P. Sakulkar *et al.*, "Wave: A distributed scheduling framework for dispersed computing," USC Auton. Netw. Res. Group, Univ. Southern California, Los Angeles, CA, USA, Tech. Rep. ANRG 2018-01, 2018.

[31] M. Król, S. Mastorakis, D. Oran, and D. Kutscher, "Compute first networking: Distributed computing meets ICN," in *Proc. 6th ACM Conf. Inf.-Centric Netw.*, Sep. 2019, pp. 67–77.

[32] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2015, pp. 1346–1354.

[33] L. Wang, Z. Lu, X. Wen, R. Knopp, and R. Gupta, "Joint optimization of service function chaining and resource allocation in network function virtualization," *IEEE Access*, vol. 4, pp. 8084–8094, 2016.

[34] L. Qu, C. Assi, and K. Shaban, "Delay-aware scheduling and resource optimization with network function virtualization," *IEEE Trans. Commun.*, vol. 64, no. 9, pp. 3746–3758, Sep. 2016.

[35] D. Bhamare, M. Samaka, A. Erbad, R. Jain, L. Gupta, and H. A. Chan, "Optimal virtual network function placement in multi-cloud service function chaining architecture," *Comput. Commun.*, vol. 102, pp. 1–16, Apr. 2017.

[36] Q. Zhang, Y. Xiao, F. Liu, J. C. S. Lui, J. Guo, and T. Wang, "Joint optimization of chain placement and request scheduling for network function virtualization," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2017, pp. 731–741.

[37] H. Feng, J. Llorca, A. M. Tulino, and A. F. Molisch, "Optimal dynamic cloud network control," *IEEE/ACM Trans. Netw.*, vol. 26, no. 5, pp. 2118–2131, Oct. 2018.

[38] J. Zhang, A. Sinha, J. Llorca, A. Tulino, and E. Modiano, "Optimal control of distributed computing networks with mixed-cast traffic flows," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, Apr. 2018, pp. 1880–1888.

[39] C.-S. Yang, R. Pedarsani, and A. S. Avestimehr, "Communication-aware scheduling of serial tasks for dispersed computing," *IEEE/ACM Trans. Netw.*, vol. 27, no. 4, pp. 1330–1343, Aug. 2019.

[40] M. Chen, Y. Hao, L. Hu, M. S. Hossain, and A. Ghoneim, "Edge-CoCaCo: Toward joint optimization of computation, caching, and communication on edge cloud," *IEEE Wireless Commun.*, vol. 25, no. 3, pp. 21–27, Jun. 2018.

[41] D. Zeng, L. Gu, S. Guo, Z. Cheng, and S. Yu, "Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system," *IEEE Trans. Comput.*, vol. 65, no. 12, pp. 3702–3712, Dec. 2016.

[42] A. Ndikumana *et al.*, "Joint communication, computation, caching, and control in big data multi-access edge computing," *IEEE Trans. Mobile Comput.*, vol. 19, no. 6, pp. 1359–1374, Jun. 2020.

[43] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Joint service placement and request routing in multi-cell mobile edge computing networks," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, Apr. 2019, pp. 10–18.

[44] G. Zhao, H. Xu, Y. Zhao, C. Qiao, and L. Huang, "Offloading dependent tasks in mobile edge computing with service caching," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, Jul. 2020, pp. 1997–2006.

[45] P. Basu *et al.*, "Decentralized placement of data and analytics in wireless networks for energy-efficient execution," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, Jul. 2020, pp. 486–495.

[46] X. Ma, A. Zhou, S. Zhang, and S. Wang, "Cooperative service caching and workload scheduling in mobile edge computing," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, Jul. 2020, pp. 2076–2085.

[47] L. Zhang *et al.*, "Named data networking (NDN) project," *Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC*, vol. 157, p. 158, Oct. 2010.

[48] M. Król and I. Psaras, "NFaaS: Named function as a service," in *Proc. 4th ACM Conf. Inf.-Centric Netw.*, Sep. 2017, pp. 134–144.

[49] L. Huang, S. Moeller, M. J. Neely, and B. Krishnamachari, "LIFO-backpressure achieves near-optimal utility-delay tradeoff," *IEEE/ACM Trans. Netw.*, vol. 21, no. 3, pp. 831–844, Jun. 2013.

[50] Y. Cui, F. Lai, E. Yeh, and R. Liu, "Enhanced VIP algorithms for forwarding, caching, and congestion control in named data networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2016, pp. 1–7.

[51] S. Ioannidis and E. Yeh, "Adaptive caching networks with optimality guarantees," *IEEE/ACM Trans. Netw.*, vol. 26, no. 2, pp. 737–750, Apr. 2018.

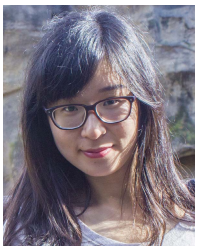
[52] M. Mahdian, A. Moharrer, S. Ioannidis, and E. Yeh, "Kelly cache networks," *IEEE/ACM Trans. Netw.*, vol. 28, no. 3, pp. 1130–1143, Jun. 2020.

[53] M. J. Neely, E. Modiano, and C. E. Rohrs, "Dynamic power allocation and routing for time-varying wireless networks," *IEEE J. Sel. Areas Commun.*, vol. 23, no. 1, pp. 89–103, Jan. 2005.

[54] L. Georgiadis, M. J. Neely, and L. Tassiulas, *Resource Allocation and Cross-Layer Control in Wireless Networks*. Boston, MA, USA: Now, 2006.

Khashayar Kamran (Student Member, IEEE) received the B.S. degree in electrical engineering from the University of Tehran, Tehran, Iran, in 2015, and the M.S. and Ph.D. degrees in electrical and computer engineering from Northeastern University, Boston, USA, in 2018 and 2020, respectively. He is currently a Software Engineer at Pinterest Inc., where he develops tools for monitoring, optimizing, and reducing mean time to resolution for Pinterest micro-service architecture. His main research interests are distributed computing systems, optimization, and network science.

Edmund Yeh (Senior Member, IEEE) received the B.S. degree (Hons.) in electrical engineering and Phi Beta Kappa from Stanford University in 1994, the M.Phil. degree in engineering from Cambridge University on the Winston Churchill Scholarship in 1995, and the Ph.D. degree in electrical engineering and computer science from MIT under Prof. R. Gallager in 2001. He was previously an Assistant Professor and an Associate Professor of electrical engineering, computer science, and statistics at Yale University. He is currently a Professor of electrical and computer engineering at Northeastern University, Boston, MA, USA. He was a recipient of the Alexander von Humboldt Research Fellowship, the Army Research Office Young Investigator Award, and the Best Paper Award at the 2017 ACM Conference on Information-Centric Networking (ICN), the 2015 IEEE International Conference on Communications (ICC) Communication Theory Symposium, and Conference on Ubiquitous and Future Networks, Phuket, Thailand, in 2012.



Qian Ma (Member, IEEE) received the B.S. degree from the Beijing University of Posts and Telecommunications, China, in 2012, and the Ph.D. degree from the Department of Information Engineering, The Chinese University of Hong Kong, in 2017. She worked as a Post-Doctoral Research Associate at Northeastern University from 2018 to 2019. She is currently an Associate Professor with the School of Intelligent Systems Engineering, Sun Yat-sen University. Her research interests lie in the field of network optimization and economics. She was a recipient of the Best Paper Award from the IEEE International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt) in 2021 and the Best Student Paper Award from the IEEE International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt) in 2015.