Joint Compute-Caching-Communication Control for Online Data-Intensive Service Delivery

Yang Cai, *Graduate Student Member, IEEE*, Jaime Llorca, *Member, IEEE*, Antonia M. Tulino, *Fellow, IEEE*, and Andreas F. Molisch, *Fellow, IEEE*

Abstract—Data-intensive augmented information (AgI) services (e.g., metaverse applications such as virtual/augmented reality), designed to deliver highly interactive experiences resulting from the real-time combination of live data-streams and pre-stored digital content, are accelerating the need for distributed compute platforms with unprecedented storage, computation, and communication requirements. To this end, the integrated evolution of next-generation networks (5G/6G) and distributed cloud technologies (mobile/edge/cloud computing) have emerged as a promising paradigm to address the interaction- and resource-intensive nature of data-intensive AgI services. In this paper, we focus on the design of control policies for the joint orchestration of compute, caching, and communication (3C) resources in next-generation 3C networks for the delivery of data-intensive AgI services. We design the first throughput-optimal control policy that coordinates joint decisions around (i) routing paths and processing locations for live data streams, with (ii) cache selection and distribution paths for associated data objects. We then extend the proposed solution to include a max-throughput data placement policy and two efficient replacement policies. Numerical results demonstrate the superior performance obtained via the novel multi-pipeline flow control and 3C resource orchestration mechanisms of the proposed policy, compared with state-of-the-art algorithms that lack full 3C integrated control.

Index Terms—Data-intensive services, virtual reality, augmented reality, metaverse, distributed cloud, mobile edge computing, fog computing, caching, network control, stream processing.

1 Introduction

The class of augmented information (AgI) services refers to a wide range of services and applications designed to deliver information of real-time relevance that results from the online aggregation, processing, and distribution of multiple data streams [2]. AgI services such as system automation (e.g., smart homes/factories/cities, self-driving cars) and metaverse experiences (e.g., multiplayer gaming, immersive video, virtual/augmented reality) are driving unprecedented requirements for communication, computation, and storage resources [3]. To address this need, distributed cloud network architectures such as multi-access edge computing (MEC) are becoming a promising paradigm, providing end users with efficient access to nearby computation resources. Together with continued advances in network virtualization and programmability [4], distributed cloud networks allow flexible and elastic deployment of disaggregated services composed of multiple software functions that can be dynamically instantiated at distributed network locations.

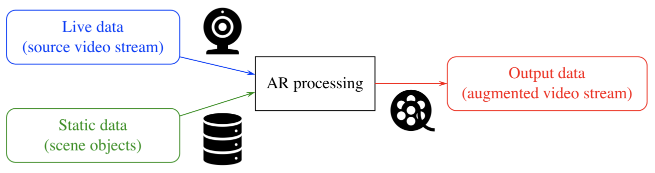


Fig. 1. Example AR application with one processing function that takes two inputs (live and static data) to create augmented data.

In addition to the *interaction-* and *compute-intensive* nature, an increasingly relevant feature of next-generation AgI services, such as metaverse applications, is their *intensive data requirements*. In these applications, the user experience results from the combination of live media streams and pre-stored digital content. Augmented reality (AR), as illustrated in Fig. 1, is a clear example, which enriches source video streams with scene objects to generate enhanced experiences that can be consumed by end users [5]. Face/object recognition is another example, where the access to a training dictionary is required to identify/classify the images recorded by end users [6].

Indeed, the efficient delivery of *data-intensive AgI services* requires the end-to-end optimization of communication, computation, and caching (3C) decisions and the joint orchestration of associated 3C resources. From a network control perspective, data packet decisions include: (i) ***packet processing:*** where to execute each service function in order to process associated data packets, (ii) ***packet caching:*** where to place possibly multiple copies of pre-produced content

- *Part of this work was presented at IEEE GlobeCom 2022 [1].*

- *Y. Cai and A. F. Molisch are with the Department of Electrical and Computer Engineering, University of Southern California, Los Angeles, CA 90089, USA.*
  E-mail: yangcai@usc.edu; molisch@usc.edu
- *J. Llorca is with the Electrical and Computer Engineering Department, New York University, Brooklyn, NY 11201 USA.*
  E-mail: jllorca@nyu.edu
- *A. M. Tulino is with the Electrical and Computer Engineering Department, New York University, Brooklyn, NY 11201 USA, and also with the Department of Electrical Engineering, Universityà degli Studi di Napoli Federico II, Naples 80138, Italy.*
  E-mail: atulino@nyu.edu; antoniamaria.tulino@unina.it

- *This material is based upon work supported by the National Science Foundation under CNS-1816699 and CNS-2148315 and is supported in part by funds from federal agency and industry partners as specified in the Resilient & Intelligent NextG Systems (RINGS) program.*

items and how to select appropriate copies for corresponding service functions, and (iii) ***packet routing:*** how to route data packets to their corresponding processing locations. In addition, the joint 3C decision making problem must be addressed in an online manner, in response to stochastic network conditions and service demands.

1.1 Related Work

1.1.1 Computation and Communication

The distributed cloud network control problem has received significant attention in recent literature, especially for applications that can be modeled as service function chains (SFCs), which include the single task offloading problem [7] as a special case [8].

One main line of work studies this problem in a *static* setting, where the goal is to allocate communication and computation resources for function placement and flow routing in order to optimize a network-wide objective, e.g., maximizing accepted service requests [9], [10], [11] or minimizing overall operational cost [12], [13], [14]. While useful for long timescale end-to-end service optimization, these solutions exhibit two main limitations: first, the problem is formulated as a *static* optimization problem without considering the dynamic nature of service demands, a critical aspect in next-generation AgI services; second, due to the combinatorial nature of the problem, the corresponding formulations typically take the form of (NP-hard) mixed integer programs (MIP), and either heuristic solutions or approximation algorithms are developed, compromising the quality of resulting solutions.

To address the SFC optimization problem in a dynamic scenario, one needs to make online packet processing, routing, and scheduling decisions, in response to stochastic system states (e.g., service demands and resource capacities). Among existing techniques, Lyapunov drift control, firstly applied to pure communication networks [15], [16], [17], has proven to be a powerful tool for the design of throughput-optimal cloud network communication and computation control policies, such as DCNC [18] and UCNC [19], by *dynamically* exploring processing and routing diversity. In general, centralized routing policies, e.g., UCNC, which exploit global knowledge (at the expense of additional communication overhead) to guarantee that packets follow *acyclic* paths, can attain better delay performance than distributed counterparts, e.g., DCNC [20].

1.1.2 Caching and Communication

Over the past decade, the dramatic growth of user demands for multimedia content has fueled rapid advances in caching techniques, especially at the wireless edge. By storing copies of popular content close to end users, the network traffic and latency for content retrieval and distribution can be significantly reduced [21], [22], [23].

Caching and delivery policies are two key elements in content distribution network design, dealing with (i) content placement in the network, and (ii) content delivery to users, respectively. Various caching policies have been designed aiming to optimize different performance metrics, e.g., throughput [21], delay [22], and energy efficiency [24]. In addition, the overall content distribution performance can benefit from the joint optimization of the caching policy and the employed communication technique, e.g., non-orthogonal multiple access (NOMA) [25], multiple-input and multiple-output (MIMO) [26], and coded-multicast [27].

In multi-hop networks, flow routing plays an important role in the delivery policy design, i.e., selecting the caching location to provision, and the path to deliver, the required content. Similarly, overall network performance can benefit from the joint optimization of caching and routing [28]. Some existing studies propose formulations targeting either throughput maximization [29] or service cost minimization [30], and approximation algorithms are developed to address the resulting MIP problems.

1.1.3 Joint 3C Optimization

While there is a large body of works on the integration of computation-communication and caching-communication technologies into network design, 3C integration is a less explored topic with fewer known results.

Two combinations, computing-assisted information centric networking (ICN) and cache-enabled MEC, are studied in [31], as promising directions for 3C integration. In cache-enabled MEC, a key aspect is service caching, dealing with service functions (software) with non-trivial storage requirements [9]; another aspect is data caching, i.e., caching frequently used data [32], such as processed results (from previous tasks) that might be repeatedly requested [33], [34], to save extra computation resources and latency for content generation.

In this paper, we focus on integrating data caching into the delivery of data-intensive AgI services (such as metaverse applications), assuming that service functions process a combination of cached digital objects (static data) and user-specific streams (live data) to generate highly personalized experiences for end users. Under such assumption, [13] developed approximation algorithms for the data-intensive service chain embedding problem in a static setting (i.e., with known average demands). A dynamic (but simplified) setting is investigated in [35], where the proposed DECO policy focuses on static object distribution and processing, but without considering the live service chain routing and processing pipeline.

1.2 Problem Statement and Challenges

In this paper, we investigate the problem of *joint 3C control for data-intensive AgI service delivery*.

As illustrated in Fig. 2, a data-intensive AgI service may be composed of multiple functions, and each function may require input streams of different nature (e.g., F2 in blue): live data (generated by device sensors), static data (pre-stored in network), or processed data (generated by previous processing functions).

Compared with existing service models, such as SFC, MEC, and DECO (also illustrated in Fig. 2), key new challenges arise in the delivery of data-intensive AgI services.

On the end-to-end flow control dimension, two new aspects arise: (i) processing location decisions impact not only the resulting computation load, but also the communication load of *all* associated live and static input streams, (ii) static data inputs can be created (via replication) at *any* caching

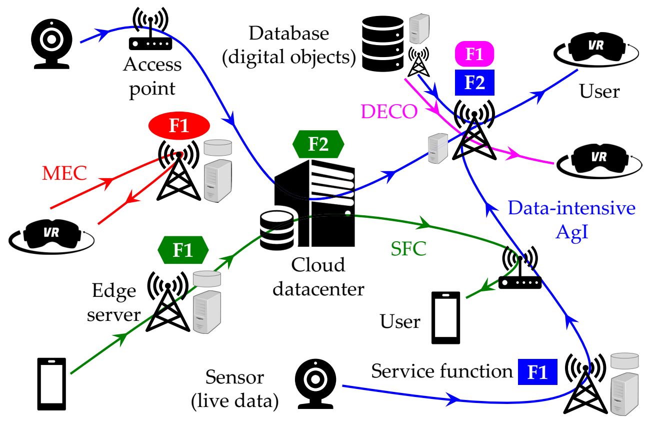


Fig. 2. Network and service models studied in this paper and in related works. Data-intensive AgI [13] (this paper): distributed cloud network + service DAG (see Section 2.2) with both live and static data. DECO [35]: distributed cloud network + one-step processing with static data. MEC [7]: single server network + one-step processing with live data. SFC [18], [19]: distributed cloud network + SFC with live data.

location that stores a copy of the required content in an *on-demand* manner (i.e., per service function's request), which is fundamentally different from live data inputs, associated with *fixed* source functions and *live streaming* rates. Existing cloud network control policies [7], [18], [19], [35], designed for simpler service models, cannot efficiently handle these challenges, let alone their inter-coupling, i.e., the need for joint selection of processing and caching locations, as well as live and static data routing paths. We term this challenge *multi-pipeline flow control*.

On the data placement dimension, a key element impacting the performance of data-intensive AgI service delivery is the caching policy design, including "which content databases to cache" and "where to place the databases". As mentioned in existing works on caching-communication integration, content placement shall be jointly optimized with flow routing decisions, but going beyond flow routing to also include flow processing, especially in heterogeneous networks with highly-distributed 3C resources, is particularly challenging. Furthermore, when the *service request distribution* is time-varying, the service delivery performance shall benefit from the dynamic adjustment of the caching policy. We collectively refer to these challenges as *processing-aware database placement*.

1.3 Contributions

This paper addresses the above problems, and our contributions are summarized as follows:

1) We characterize the stability region of distributed cloud networks supporting data-intensive AgI service delivery, in the settings of fixed and dynamic database placement.
2) We design the first throughput-optimal control policy for online data-intensive service delivery, termed DI-DCNC, which coordinates joint decisions around (i) routing paths and processing locations for live data streams, and (ii) cache selection and distribution paths for associated static data objects, under a given database placement.

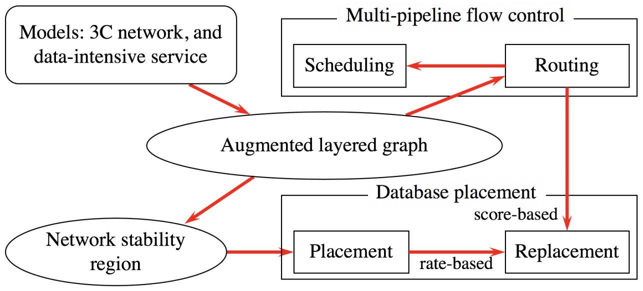


Fig. 3. Main components of overall methodology and their relationship.

3) We propose a database placement policy targeting throughput maximization, and derive an equivalent mixed integer linear programming (MILP) problem to implement the design.
4) We develop two database replacement policies able to adapt to time-varying service demand statistics, based on online estimations of service request distribution and database score, respectively.

We emphasize that the methodology proposed in this paper targets the general class of data-intensive AgI services characterized by the multi-pipeline service model introduced in Section 2.2, rather than any specific application, although many metaverse applications such as virtual/augmented reality are relevant special cases.

1.4 Paper Organization and Main Components

The paper is organized as follows. In Section 2, we introduce the system model, including 3C network and data-intensive AgI service models. In Section 3, we describe the augmented layered graph (ALG) used to model the multi-pipeline flow control as a routing problem, and characterize the network stability region. Section 4 presents the proposed DI-DCNC algorithm for multi-pipeline flow control, consisting of 1) a global live and static data routing policy and 2) a local scheduling policy, shown to be throughput-optimal under any given database placement. Section 5 describes a throughput-optimal database placement policy leveraging an equivalent characterization of the network stability region, and Section 6 two replacement policies, one guided by the placement policy (*rate-based*) and the other by the routing policy (*score-based*). Section 7 presents the numerical results, while conclusions are drawn in Section 8.

The main components of the overall approach presented in this paper and their relationship are depicted in Fig. 3, while frequently used notation is summarized in Table 1.

2 System Model

Fig. 2 illustrates the *cache-enabled MEC network* as the supporting infrastructure for the delivery of *data-intensive AgI services*, described as follows.

2.1 3C Network Model

Consider a distributed cloud network, modeled by a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with $\mathcal{V}$ and $\mathcal{E}$ denoting the node and edge sets, respectively. Each vertex $i \in \mathcal{V}$ represents

TABLE 1
Table of Notations

Symbol	Description
$\mathcal{V}, \mathcal{E}$	Node and edge sets of the actual network.
C_i, C_{ij}, S_i	3C resource capacities.
ϕ	Data-intensive AgI service.
$\mathcal{K}, F_k$	Set of databases, the size of database k.
(ξ, r, k, ζ)	Scaling factor, workload, object name, merging ratio.
m	Processing stage (also used as function index).
$s, \mathcal{V}(k), d$	Live source, set of static sources, destination.
$a^{(c)}(t), \lambda^{(c)}$	Number of arrivals of client c, arrival rate.
$\mathcal{V}^{(\phi)}, \mathcal{E}^{(\phi)}$	Node and edge sets of ALG for service ϕ.
o'_m	Super static source (of stage m static packets).
$\sigma, \mathcal{F}_c(x)$	Efficient route and the set of them.
$a^{(c,\sigma)}(t)$	Number of service requests selecting ER σ.
$\theta^{(m)}$	Processing location for function m.
$w_{ij}^{(c)}, \rho_{ij}^{(c,\sigma)}$	Resource load on an ALG edge/actual link.
$\Lambda(x), \Lambda$	Stability region under fixed/dynamic placements.
$\tilde{Q}(t), Q(t)$	Virtual queue and normalized virtual queue.
$x_{i,k}$	Caching variable (if database k is cached at node i).
$f_{ij}^{(c)}, f_{ij}'^{(k)}$	Live/static flows in ALG/actual network.
$p^{(c)}$	Service request distribution.
$U_{i,k}$	Score (of database k at node i).

a node equipped with computation resources (e.g., edge server) for service function processing. Each edge $(i, j) \in \mathcal{E}$ represents a point-to-point communication link, which can support data transmission from node i to j. Let $\delta^-(i)$ and $\delta^+(i)$ denote the incoming and outgoing neighbor sets of node i, respectively.

Time is slotted, and the network processing and transmission resources are quantified as follows:

- Processing capacity C_i: the maximum number of processing instructions (e.g., floating point operations) that can be executed in one time slot at node i.
- Transmission capacity C_{ij}: the maximum number of data units (e.g., packets) that can be transmitted in one time slot over link (i, j).

The network nodes are also equipped with storage resources[1] to cache databases composed of digital objects whose access may be required for service function processing. Let $\mathcal{K}$ denote the set of databases. Define the **caching vector** as

$$x = \{x_{i,k} \in \{0, 1\} : i \in \mathcal{V}, k \in \mathcal{K}\} \tag{1}$$

where $x_{i,k}$ is a binary variable indicating if database $k \in \mathcal{K}$ is cached at node i ($x_{i,k} = 1$) or not ($x_{i,k} = 0$). Let $\mathcal{V}(k) = \{i \in \mathcal{V} : x_{i,k} = 1\} \subset \mathcal{V}$ denote the *static sources* of database k, i.e., the set of nodes that cache database k. A caching vector must satisfy the following storage resource constraint: for $\forall\, i \in \mathcal{V}$,

$$\sum_{k\in\mathcal{K}} F_k\, x_{i,k} \le S_i \tag{2}$$

where F_k denotes the size of database $k \in \mathcal{K}$, and S_i the storage capacity of node $i \in \mathcal{V}$, i.e., the maximum number

1. In this paper, "storage resource" refers to memory or disk used for database caching. Data packets emanating from live or static data that travel through the network are collected in separate buffers, referred to as "actual queues" (see Section 2.4).

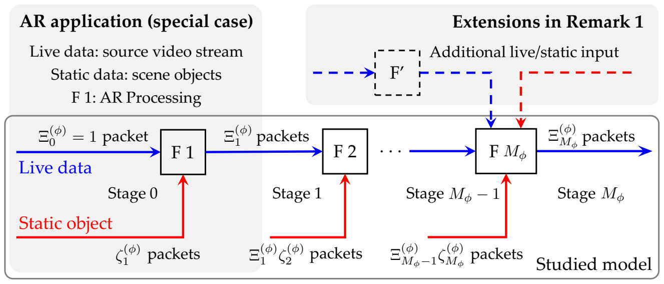

AR application (special case)
Live data: source video stream
Static data: scene objects
F 1: AR Processing
Extensions in Remark 1
F′
Additional live/static input
$\Xi_0^{(\phi)} = 1$ packet
Live data
F 1
Stage 0
Static object
$\zeta_1^{(\phi)}$ packets
$\Xi_1^{(\phi)}$ packets
F 2
Stage 1
$\Xi_1^{(\phi)}\zeta_2^{(\phi)}$ packets
F M_ϕ
Stage $M_\phi - 1$
$\Xi_{M_\phi-1}^{(\phi)}\zeta_{M_\phi}^{(\phi)}$ packets
$\Xi_{M_\phi}^{(\phi)}$ packets
Stage M_ϕ
Studied model

Fig. 4. The studied data-intensive service model composed of multiple functions (denoted by F), with each function requiring one live data input and one static data input. We depict an AR application with a single processing step (F1) as a special case, as well as extensions to multiple live and static inputs (using F M_ϕ as an example).

of static data units (e.g., databases) that can be cached at node $i \in \mathcal{V}$, respectively. Let $\mathcal{X}$ denote the caching vectors $\boldsymbol{x}$ satisfying (2).

We assume that there exists a cloud datacenter in the network, serving as an *external trusted source* with all databases stored, from which the edge servers can download databases for caching. We assume that such downloads happen at a longer timescale and neglect their impact on the network communication resources.[2]

2.2 Data-Intensive AgI Service Model

We assume a data-intensive service ϕ is composed of a sequence of M_ϕ functions, through which the user-specific data, referred to as *live data*, must be processed to produce consumable streams, resulting in the end-to-end data stream divided into $M_\phi + 1$ stages. We refer to the output stream of function $m \in \{1, \cdots, M_\phi\}$ as *stage m live packets*, with source packets denoted as stage 0 packets. In order to process each live packet, the associated service function requires access to a pre-stored digital object, referred to as *static data* [13], [35]. We then use *stage m static packets* to denote the static object input stream to function $m + 1$. Each processing step can take place at different network locations hosting the required service functions: for example, in Fig. 2, the two service functions F1 and F2 (in blue) are executed at different edge servers.

Each service function, say the mth function of service ϕ, is specified by 4 parameters $(\xi_m^{(\phi)}, r_m^{(\phi)}, k_m^{(\phi)}, \zeta_m^{(\phi)})$, defined as follows:

- **Object name** $k_m^{(\phi)}$: the name (or index) of the database to which the static object belongs.
- **Merging ratio** $\zeta_m^{(\phi)}$: the number of static packets per input live packet.
- **Workload** $r_m^{(\phi)}$: the amount of computation resource (e.g., instructions per time slot) to process one input live packet.
- **Scaling factor** $\xi_m^{(\phi)}$: the number of output packets per input live packet.

2. Database replacement can be supported by the backhaul connections between the cloud datacenter and edge servers, subject to *restricted* communication rates.

We also define the **cumulative scaling factor** $\Xi_m^{(\phi)}$ as the number of stage m live packets per stage 0 live packet (i.e., the live packet prior to any processing operation), given by

$$\Xi_m^{(\phi)} = \begin{cases} 1 & m = 0 \\ \xi_m^{(\phi)} \Xi_{m-1}^{(\phi)} & m = 1, \cdots, M_\phi \end{cases}. \tag{3}$$

Remark 1 (Extended Models). We note that in general, a data-intensive service may be described by a directed acyclic graph (DAG) with multiple live and static data inputs per service function. While for ease of exposition, we illustrate the proposed design in the context of one live and one static input per function, the generalization to multiple inputs is straightforward: (i) For functions with multiple static inputs, we can extend $k_m^{(\phi)}$ and $\zeta_m^{(\phi)}$ (from scalars) to sets. (ii) For functions with multiple live inputs, we can create a tree node for each service function and describe its inputs as child-nodes until reaching the source data (i.e., leaf nodes).

2.3 Client Model

We define each client c by a 3-tuple (s, d, ϕ), denoting the source node s (where the live packets arrive to the network), the destination node d (where the final packets are requested for consumption), and the requested service ϕ (which defines the sequence of service functions and the static packets that are required to process the live packets and create the final packets), respectively.[3]

2.3.1 Live Packet Arrival

Let $a^{(c)}(t)$ be the number of live packets of client c arriving to the network at time t. For each client c, we assume the arrival process $\{a^{(c)}(t) : t \geq 0\}$ is i.i.d. over time, with mean arrival rate of $\lambda^{(c)}$, and bounded maximum arrival number. Each live packet is immediately admitted to the network upon arrival.

Remark 2. In Section 5 (and subsequent sections), we assume that there exists a service request distribution (32) $\{p^{(c)} : \forall c\}$ governing the arrival rates of all clients, i.e., $\lambda^{(c)} \propto p^{(c)}$, when designing database placement policies targeting throughput maximization.

Remark 3. We assume i.i.d. arrivals for ease of exposition. The analytical results: Theorem 1, 2, and Proposition 2, are valid under the general assumption of Markov-modulated arrivals, i.e., the arrival rate is time-varying and follows a Markov process (see [16, Section 4.9]).

2.3.2 Static Packet Provisioning

Upon a live packet arrival, for each required static packet, one static source is selected to generate the required copy, which is loaded into the network immediately.

Definition 1 (Packet-level request). We refer to a live packet and all static packets required for its (multi-stage) processing as belonging to the same packet-level request.

In the following, we use *packet-level request* and *request* interchangeably.

3. We use packet to refer to the minimum data unit that can be processed independently by the service or application, such as a video segment or frame in video-based applications.

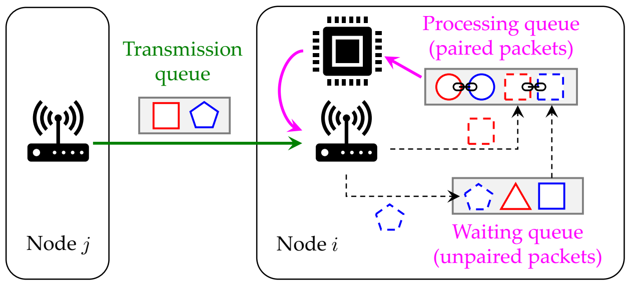

Transmission queue
Processing queue (paired packets)
Node j
Node i
Waiting queue (unpaired packets)

Fig. 5. Illustration of the paired-packet queuing system. Different shapes denote packets associated with different requests, blue and red colors the live and static packets, and solid and dashed lines the current and subsequent time slot, respectively.

Remark 4 (Static Object). A database is composed of multiple objects, e.g., the scene object library in an AR application, and distinct objects may be required by different service requests. We assume that the live packet and static packets belonging to the same packet-level request get *associated*, i.e., the static packets are dedicated for the processing of the associated live packet.

2.4 Queuing System

Each packet (live or static) admitted to the network gets associated with a route for its delivery, and we establish actual queues to accommodate packets waiting for processing or routing.

For each link $(i, j) \in \mathcal{E}$, we create one *transmission* queue collecting all packets – regardless of the client, stage, or type (i.e., live or static) – waiting to cross the link, i.e., packets currently held at node i and having node j as its next hop in the route. In contrast, a novel **paired-packet queuing system** is constructed at each node $i \in \mathcal{V}$, composed of the following queues: (i) the *processing* queue collecting the *paired* live and static packets concurrently present at node i, which are ready for joint processing, and (ii) the *waiting* queue collecting the *unpaired* live or static packets waiting for their in-transit associates, which are not qualified for processing until joining the processing queue upon their associates' arrivals.

An illustrative example is shown in Fig. 5. At the current time slot, the paired-packet processing queue holds a blue and red circle pair, representing live and static packets belonging to the same request, which are ready for processing. At the next time slot, when node i receives the red square packet, it gets paired with the blue square packet held in the waiting queue, and together enter the paired-packet processing queue to be scheduled for processing.

3 Policy Space and Stability Region

In this section, we propose an ALG model to analyze and optimize the data-intensive AgI service delivery problem, based on which we characterize the network stability region.

3.1 Augmented Layered Graph

Recent studies have shown that the AgI service (modeled by SFC) control problem can be transformed into a packet routing problem on a properly constructed *layered graph* [19].

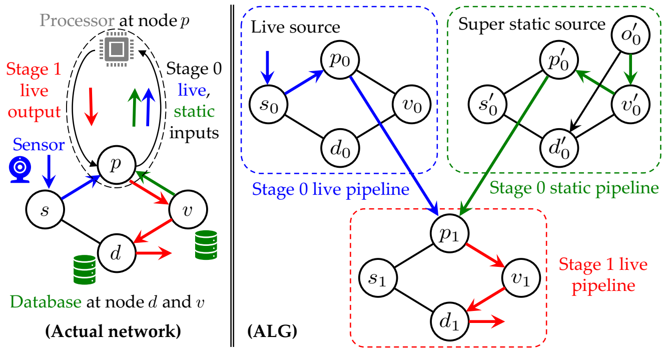


Fig. 6. Illustration of the ALG model for the delivery of AR service.

3.1.1 Topology of the ALG

The ALG associated with service ϕ is composed of $M_\phi + 1$ layers, indexed by layer $0, \cdots, M_\phi$, respectively. Within each layer m, there are two pipelines, referred to as *stage m live* and *static* pipelines, respectively, except for layer M_ϕ, which only includes the live pipeline. Each live pipeline has the same topology as the actual network, while the stage m static pipeline includes an additional *super static source* node o'_m and its outgoing edges to all static sources, i.e., (o'_m, v'_m) with $\forall\, v \in \mathcal{V}(k_m^{(\phi)})$. We note that: (i) The live and static pipelines in layer m accommodate stage m live and static packets, respectively, and represent their associated routing over the network. (ii) With the super static source o'_m created for the static pipeline, it is equivalent to assume that o'_m *is the only static source of database* $k_m^{(\phi)}$.[4] (iii) There are inter-layer edges connecting corresponding nodes in adjacent layers m and $m+1$, which represent processing operations, i.e., the stage m live and static packets pushed through these edges are processed into stage $m+1$ live packets in the actual network.

The example in Fig. 6 illustrates the delivery of a single-function AR application (as shown in Fig. 4) over a 4-node network. The stage $m = 0$ live packet, which arrives to the network at the source node s, and the stage $m = 0$ static packet, which is generated via replication at the static source v, are routed following the blue and green paths to node p, respectively. After getting processed at node p, the produced stage $m+1 = 1$ packet is delivered along the red path to destination d. In the ALG, the highlighted links in different pipelines indicate the routing paths of each packet; (o'_1, v'_1) indicates selecting the static source $v \in \mathcal{V}(k_1^{(\phi)}) = \{v, d\}$ to create the static packet, and (p_1, p_2) and (p'_1, p_2) indicate packet processing at node p.

Mathematically, given the actual network $\mathcal{G}$ and the database placement x, the ALG of service ϕ, denoted by $\mathcal{G}^{(\phi)} = (\mathcal{V}^{(\phi)}, \mathcal{E}^{(\phi)})$, is defined as

$$\mathcal{V}^{(\phi)} = \bigcup_{m=0}^{M_\phi} \mathcal{V}_{\mathrm{L},m}^{(\phi)} \cup \bigcup_{m=0}^{M_\phi - 1} \mathcal{V}_{\mathrm{S},m}^{(\phi)} \tag{4a}$$

$$\mathcal{E}^{(\phi)} = \bigcup_{m=0}^{M_\phi} \mathcal{E}_{\mathrm{L},m}^{(\phi)} \cup \bigcup_{m=0}^{M_\phi - 1} \mathcal{E}_{\mathrm{S},m}^{(\phi)} \cup \bigcup_{m=0}^{M_\phi - 1} \mathcal{E}_{m,m+1}^{(\phi)} \tag{4b}$$

in which (with L/S in the subscripts denoting live/static)

$$\mathcal{V}_{\mathrm{L},m}^{(\phi)} = \{i_m : i \in \mathcal{V}\},\ \mathcal{V}_{\mathrm{S},m}^{(\phi)} = \{i'_m : i \in \mathcal{V}\} \cup \{o'_m\}$$
$$\mathcal{E}_{\mathrm{L},m}^{(\phi)} = \{(i_m, j_m) : (i,j) \in \mathcal{E}\}$$
$$\mathcal{E}_{\mathrm{S},m}^{(\phi)} = \{(i'_m, j'_m) : (i,j) \in \mathcal{E}\} \cup \{(o'_m, v'_m) : v \in \mathcal{V}(k_m^{(\phi)})\}$$
$$\mathcal{E}_{m,m+1}^{(\phi)} = \{(i_m, i_{m+1}), (i'_m, i_{m+1}) : i \in \mathcal{V}\}.$$

Remark 5. Note that, on one hand, the proposed ALG model can capture special cases such as services modeled as SFCs (in which static objects are not relevant) by removing the static pipelines, which reduces to the layered graph model in existing works [19]. On the other hand, extended service models with multiple live/static inputs (see Remark 1) can be flexibly incorporated by adding extra pipelines into each layer.

3.1.2 Flow in the ALG

Let $f_{ij} \geq 0$ denote the network flow associated with edge $(i,j) \in \mathcal{E}^{(\phi)}$ in the ALG, defined as the average packet rate traversing the edge (in *packets per slot*). In particular:

- $f_{i_m j_m}$ and $f_{i'_m j'_m}$ denote the transmission rates of stage m live and static packets over link $(i,j) \in \mathcal{E}$.
- $f_{o'_m v'_m}$ denotes the local replication rate of stage m static packets at the static source $v \in \mathcal{V}(k_m^{(\phi)})$.
- $f_{i_m i_{m+1}}$ and $f_{i'_m i_{m+1}}$ denote the processing rates of stage m live and static packets at node $i \in \mathcal{V}$.

The flow rates must satisfy the following constraints:
(i) *Live flow conservation:* for $\forall\, i \in \mathcal{V}, 0 \leq m \leq M_\phi$,

$$\sum_{j \in \delta^+(i)} f_{i_m j_m} + f_{i_m i_{m+1}} = \sum_{j \in \delta^-(i)} f_{j_m i_m} + \xi_m^{(\phi)} f_{i_{m-1} i_m}, \tag{5}$$

i.e., for stage m live flow, the total outgoing rate of packets that are transmitted and processed equals the total incoming rate of packets that are received and generated by processing. Note that processing stage $m-1$ live packets at rate $f_{i_{m-1} i_m}$ (by function m) produces stage m live packets at rate $\xi_m^{(\phi)} f_{i_{m-1} i_m}$ at node i, by the definition of "scaling factor" in Section 2.2. Define $f_{i_{-1} i_0} = f_{i_{M_\phi} i_{M_\phi + 1}} = 0$.
(ii) *Static flow conservation:* for $\forall\, i \in \mathcal{V}, 0 \leq m \leq M_\phi - 1$,

$$\sum_{j \in \delta^+(i)} f_{i'_m j'_m} + f_{i'_m i_{m+1}} = \sum_{j \in \delta^-(i)} f_{j'_m i'_m} + f_{o'_m i'_m}, \tag{6}$$

i.e., for stage m static flow, the total outgoing rate of packets that are transmitted and processed equals the total incoming rate of packets that are received and generated via replication. Define $f_{o'_m i'_m} = 0$ for $i \notin \mathcal{V}(k_m^{(\phi)})$, i.e., nodes that are not static sources.
(iii) *Data merging:* for $\forall\, i \in \mathcal{V}, 1 \leq m \leq M_\phi$,

$$f_{i'_{m-1} i_m} = \zeta_m^{(\phi)} f_{i_{m-1} i_m}, \tag{7}$$

4. To wit, if node o'_m can provide static packets to node i'_m along the path $(o'_m, v'_m, \cdots, i'_m)$ in the ALG, then, in the actual network, we can select the static source v to produce the packets and send them to node i along the rest of the path. And vice versa.

i.e., the processing rates of static and live packets at each node i are associated by the merging ratio $\zeta_m^{(\phi)}$, as defined in Section 2.2.

Remark 6. We note that multiple edges in the ALG are associated with the same node/link in the actual network. For example, edges (i_m, j_m), (i'_m, j'_m) $\forall\, m$ are associated with link (i,j), and all operations on these edges consume the link's communication resource.

3.2 Policy Space

In the following, we first define the space of policies for data-intensive service delivery under a given database placement x (which is optimized via a separate procedure, as presented in Section 5 and 6), encompassing joint packet processing, routing, and replication decisions. In line with [19] and the increasingly adopted software defined networking (SDN) paradigm, we focus on *centralized/source routing* and *distributed scheduling* policies. To be specific: for each arriving packet, the policy selects a route consisting of (i) routing paths and processing locations for the live packets, and (ii) cache locations (to replicate each static packet) and distribution paths for associated static packets; in addition, each node/link needs to schedule packets for processing/transmission at each time slot:

Route Selection: For each packet-level request, choose a set of edges in the ALG and associated flow rates f satisfying (5) – (7), based on which: (i) the cache selection decision is specified by the replication rate (i.e., $f_{o'_m i'_m}$ units of static packets $k_m^{(\phi)}$ is produced at node i), (ii) the routing path for each packet (live or static) is given by the edges with non-zero rates in the corresponding pipeline, and (iii) the processing location selection is indicated by the processing rates $f_{i_{m-1} i_m}$ and $f_{i'_{m-1} i_m}$, which guarantee that the live and static packets meet at the same node i due to (7).

Packet Scheduling: At each time slot, for each node i and link (i,j), schedule packets from the processing queues (which hold *paired* live and static packets) and transmission queues for corresponding operations, without exceeding associated resource capacities C_i and C_{ij}.

3.2.1 Efficient Policy Space

In this section, we define an efficient policy space in which the routing path of each packet is required to be *acyclic*, without compromising the achievable performance (e.g., throughput, delay, and resource consumption).

More concretely, each request gets associated with an **efficient route (ER)** σ in the ALG, defined as:

- σ includes a sequence of processing locations, denoted by $\{\theta^{(m)} \in \mathcal{V} : 1 \le m \le M_\phi\}$, with corresponding edges in the ALG given by
$$\big\{\big([\theta^{(m)}]_{m-1}, [\theta^{(m)}]_m\big), \big([\theta^{(m)}]'_{m-1}, [\theta^{(m)}]_m\big)\big\}_{m=1}^{M_\phi}$$
where function m is executed at node $\theta^{(m)}$, and we define $\theta^{(0)} = s$ and $\theta^{(M_\phi+1)} = d$.
- σ includes acyclic routing paths for all packets, i.e.,
$$\sigma_{1,m} = \big([\theta^{(m)}]_m, \cdots, [\theta^{(m+1)}]_m\big),\ 0 \le m \le M_\phi,$$
$$\sigma_{2,m} = \big(o'_m, \cdots, [\theta^{(m+1)}]'_m\big),\ 0 \le m \le M_\phi - 1,$$
denote the (multi-hop) paths of stage m live packet (from $[\theta^{(m)}]_m$ to $[\theta^{(m+1)}]'_m$) and stage m static packet (from o'_m to $[\theta^{(m+1)}]'_m$), respectively.

In the efficient policy space, for each client c, the set of all possible ERs, denoted by $\mathcal{F}_c(x)$, is *finite*, and the route selection decision can be represented by
$$A(t) = \{a^{(c,\sigma)}(t) : \sigma \in \mathcal{F}_c(x), c\} \tag{8}$$
where $a^{(c,\sigma)}(t) \ge 0$ denotes the number of requests raised by client c at time t that get associated with σ for delivery, which satisfies
$$\sum_{\sigma \in \mathcal{F}_c(x)} a^{(c,\sigma)}(t) = a^{(c)}(t),\ \forall\, c. \tag{9}$$

Note however that $\mathcal{F}_c(x)$ includes an exponential number of ERs, i.e., $|\mathcal{F}_c(x)| = \Omega(|\mathcal{V}|^{M_\phi})$.

3.2.2 Decision Variables

To summarize, the decision variables include: i) route selection $A(t)$ specifying the distribution of arriving packets to the corresponding ERs at time t, defined in (8); ii) (for each interface) the set of packets scheduled for operation from the corresponding queue; iii) caching vector x, specifying the placement of databases in the network, defined in (1).

3.3 Network Stability Region

In this section, we characterize the *network stability region*, which describes the network capability to support service requests, defined as follows.

Definition 2. The network stability region is defined as the set of arrival vectors $\boldsymbol{\lambda}$ under which there exists an admissible policy to stabilize the actual queues, i.e.,
$$\lim_{t\to\infty} \frac{1}{t}\Big[\sum_{i\in\mathcal{V}} (R_i(t) + R'_i(t)) + \sum_{(i,j)\in\mathcal{E}} R_{ij}(t)\Big] = 0$$
where $R_i(t)$, $R'_i(t)$ and $R_{ij}(t)$ denote the backlogs of the processing queue for node i, waiting queue for node i, and transmission queue for link (i,j) at time t.

Let $\Lambda(x)$ and Λ denote the network stability regions under fixed database placement x and when allowing dynamic replacement, which are characterized in the following.

Theorem 1. For any fixed database placement $x \in \mathcal{X}$, an arrival vector $\boldsymbol{\lambda}$ is interior to the stability region $\Lambda(x)$ if and only if for each client c, there exist probability values
$$\mathbb{P}_c(\sigma) : \sum_{\sigma\in\mathcal{F}_c(x)} \mathbb{P}_c(\sigma) = 1 \text{ and } \mathbb{P}_c(\sigma) \ge 0,$$
such that for each node i and link (i,j):
$$\sum_c \lambda^{(c)} \sum_{\sigma\in\mathcal{F}_c(x)} \rho_i^{(c,\sigma)} \mathbb{P}_c(\sigma) \le C_i \tag{10a}$$
$$\sum_c \lambda^{(c)} \sum_{\sigma\in\mathcal{F}_c(x)} \rho_{ij}^{(c,\sigma)} \mathbb{P}_c(\sigma) \le C_{ij} \tag{10b}$$
where $\rho_i^{(c,\sigma)}$ and $\rho_{ij}^{(c,\sigma)}$ denote the processing and transmission resource loads imposed on node i and link (i,j)

if a packet-level request of client c is delivered by ER σ, given by:

$$\rho_i^{(c,\sigma)} = \sum_m w_{i_{m-1}i_m}^{(c)} \mathbf{1}_{\{(i_{m-1},i_m)\in\sigma\}} \tag{11}$$

$$\rho_{ij}^{(c,\sigma)} = \sum_m \big[w_{i_m j_m}^{(c)} \mathbf{1}_{\{(i_m,j_m)\in\sigma\}} + w_{i'_m j'_m}^{(c)} \mathbf{1}_{\{(i'_m,j'_m)\in\sigma\}} \big]$$

in which $\boldsymbol{w}^{(c)} = \{w_{\imath\jmath}^{(c)} : (\imath,\jmath)\in\mathcal{E}^{(\phi)}\}$ is given by

$$w_{\imath\jmath}^{(c)} = \begin{cases} \Xi_m^{(\phi)} r_{m+1}^{(\phi)} & (\imath,\jmath) = (i_m, i_{m+1}) \\ 0 & (\imath,\jmath) = (i'_m, i_{m+1}) \text{ or } (o'_m, i'_m) \\ \Xi_m^{(\phi)} & (\imath,\jmath) = (i_m, j_m) \\ \Xi_m^{(\phi)} \zeta_{m+1}^{(\phi)} & (\imath,\jmath) = (i'_m, j'_m) \end{cases}. \tag{12}$$

Proof: The proof for necessity is given in Appendix A, and we show the sufficiency by designing an admissible policy, DI-DCNC, in the subsequent section, and proving that it can support any arrival vector $\boldsymbol{\lambda} \in \Lambda(x)$. □

In Theorem 1: (i) The "sum" operation in (11) results from *multiple* ALG edges sharing a *common* node/link (see also Remark 6). (ii) A randomized policy for route selection can be defined based on the probability values $\mathbb{P}_c(\sigma)$, operating as follows: at each time slot, select the ER $\sigma \in \mathcal{F}_c(x)$ to deliver the requests of client c with probability (w.p.) $\mathbb{P}_c(\sigma)$. (iii) The result is valid under the general assumption of Markov-modulated arrivals, in which case the stability region is defined with respect to (w.r.t.) the time average arrival rate $\lambda^{(c)} = \lim_{T\to\infty}(1/T)\sum_{t=0}^{T-1} a^{(c)}(t)$.

Proposition 1. When allowing database replacement, an arrival vector $\boldsymbol{\lambda}$ is interior to the stability region Λ if and only if there exist probability values

$$\mathbb{P}(x) : \sum_{x\in\mathcal{X}} \mathbb{P}(x) = 1 \text{ and } \mathbb{P}(x) \geq 0,$$

and (for each database placement $x \in \mathcal{X}$ and client c)

$$\mathbb{P}_{c,x}(\sigma) : \sum_{\sigma\in\mathcal{F}_c(x)} \mathbb{P}_{c,x}(\sigma) = 1 \text{ and } \mathbb{P}_{c,x}(\sigma) \geq 0,$$

such that for each node i and link (i,j):

$$\sum_{x\in\mathcal{X}} \mathbb{P}(x) \sum_c \lambda^{(c)} \sum_{\sigma\in\mathcal{F}_c(x)} \rho_i^{(c,\sigma)} \mathbb{P}_{c,x}(\sigma) \leq C_i \tag{13a}$$

$$\sum_{x\in\mathcal{X}} \mathbb{P}(x) \sum_c \lambda^{(c)} \sum_{\sigma\in\mathcal{F}_c(x)} \rho_{ij}^{(c,\sigma)} \mathbb{P}_{c,x}(\sigma) \leq C_{ij} \tag{13b}$$

with $\rho_i^{(c,\sigma)}$ and $\rho_{ij}^{(c,\sigma)}$ given by (11).

Proof: See Appendix D. □

In the above proposition, $\mathbb{P}(x)$ represents the distribution of caching vector x over time, resulting from the employed replacement policy. $\mathbb{P}_{c,x}(\sigma)$ plays an equivalent role to $\mathbb{P}_c(\sigma)$ in Theorem 1, i.e., the probability values specifying the route selection policy *under placement* x.

Comparing Theorem 1 and Proposition 1, we find that allowing database replacement is promising to enlarge the stability region. To wit, setting $\mathbb{P}(x) = \mathbf{1}_{\{x=x_0\}}$ in Proposition 1 leads to the same characterization as Theorem 1 with $x = x_0$. The improvement brought by replacement is intuitive under the assumption that database replacement can be performed *instantaneously*, as one can adjust the database placement x at each time slot according to the received requests to optimize the service delivery performance. In Proposition 3, we will show that such result remains valid under restricted communication rate for database replacement, in line with footnote 2.

4 Multi-Pipeline Flow Control

In this section, we present the proposed algorithm, referred to as *data-intensive dynamic cloud network control* (DI-DCNC), which makes joint routing and processing decisions for live and static pipelines, as well as packet scheduling, under fixed database placement x.

We first introduce a single-hop virtual system (in Section 4.1) to derive packet routing decisions (in Section 4.2). Then, we present the packet scheduling policy, and summarize the actions to take in the actual network (in Section 4.3).

4.1 Virtual System

4.1.1 Precedence Constraint

In line with [17], [19], we create a virtual network that has the same topology as the actual network, with a virtual queue associated with each node and link. The *precedence constraint*, which imposes a packet to be transmitted hop-by-hop along its route, is relaxed by allowing a packet upon route selection to be immediately inserted into the virtual queues associated with all links in the route. The virtual queue measures the processing/transmission resource load for the node/link in the virtual system, which is interpreted as the **anticipated** resource load for the corresponding node/link in the actual network.

For example, suppose that a packet gets associated with the route (i_1, i_2, i_3). Then, it immediately impacts the queuing states of link (i_1, i_2) and (i_2, i_3) in the virtual system, as opposed to the actual network, where it cannot enter the queue for link (i_2, i_3) before crossing (i_1, i_2).

We emphasize that the virtual system is only used for route selection and it is not relevant to packet scheduling.

4.1.2 Virtual Queues

Let $\tilde{Q}_i(t)$ and $\tilde{Q}_{ij}(t)$ denote the virtual queues for node $i \in \mathcal{V}$ and link $(i,j) \in \mathcal{E}$, respectively. The queuing dynamics are given by:

$$\tilde{Q}_i(t+1) = \big[\tilde{Q}_i(t) - C_i + \tilde{a}_i(t)\big]^+ \tag{14a}$$

$$\tilde{Q}_{ij}(t+1) = \big[\tilde{Q}_{ij}(t) - C_{ij} + \tilde{a}_{ij}(t)\big]^+ \tag{14b}$$

where C_i and C_{ij} are interpreted as the amount of processing/transmission resource that is "served" at time t; $\tilde{a}_i(t)$ and $\tilde{a}_{ij}(t)$ are "additional" resource loads imposed on the node/link by newly arriving packets. Recall that each request gets associated with a route for delivery upon arrival, which immediately impacts the queuing states of all links in the route, and thus:

$$\tilde{a}_i(t) = \sum_c \sum_{\sigma\in\mathcal{F}_c(x)} \rho_i^{(c,\sigma)} a^{(c,\sigma)}(t) \tag{15a}$$

$$\tilde{a}_{ij}(t) = \sum_c \sum_{\sigma\in\mathcal{F}_c(x)} \rho_{ij}^{(c,\sigma)} a^{(c,\sigma)}(t) \tag{15b}$$

with $\rho_i^{(c,\sigma)}$ and $\rho_{ij}^{(c,\sigma)}$ given by (11).

4.2 Optimal Virtual Network Decisions

4.2.1 Lyapunov Drift Control

Next, we leverage Lyapunov drift control theory to derive a policy that stabilizes the *normalized* virtual queues:

$$\boldsymbol{Q}(t) = \{Q_i(t) \triangleq \tilde{Q}_i(t)/C_i : i \in \mathcal{V}\} \\ \cup \{Q_{ij}(t) \triangleq \tilde{Q}_{ij}(t)/C_{ij} : (i,j) \in \mathcal{E}\}, \tag{16}$$

which have equivalent stability properties as the virtual queues (due to the linear scaling) and can be interpreted as *queuing delays* in the virtual system.

Define the Lyapunov function as $L(t) \triangleq \|\boldsymbol{Q}(t)\|^2/2$, and the Lyapunov drift $\Delta(\boldsymbol{Q}(t)) \triangleq L(t+1) - L(t)$. Then we can derive (as shown in Appendix B.1) the following upper bound of the drift $\Delta(\boldsymbol{Q}(t))$:

$$\Delta(\boldsymbol{Q}(t)) \\ \leq B - \|\boldsymbol{Q}(t)\|_1 + \sum_{c} \sum_{\sigma \in \mathcal{F}_c(x)} O^{(c,\sigma)}(t)\, a^{(c,\sigma)}(t) \tag{17}$$

where B is a constant, and $O^{(c,\sigma)}(t)$ is referred to as the weight of ER σ, given by:

$$O^{(c,\sigma)}(t) = \sum_{i \in \mathcal{V}} \frac{Q_i(t)}{C_i} \rho_i^{(c,\sigma)} + \sum_{(i,j) \in \mathcal{E}} \frac{Q_{ij}(t)}{C_{ij}} \rho_{ij}^{(c,\sigma)} \tag{18a}$$

$$= \sum_{i \in \mathcal{V}} \sum_{m} \tilde{w}_{i_{m-1} i_m}^{(c)}(t) \mathbf{1}_{\{(i_{m-1}, i_m) \in \sigma\}} + \sum_{(i,j) \in \mathcal{E}} \sum_{m} \Big[\\ \tilde{w}_{i_m j_m}^{(c)}(t) \mathbf{1}_{\{(i_m, j_m) \in \sigma\}} + \tilde{w}_{i'_m j'_m}^{(c)}(t) \mathbf{1}_{\{(i'_m, j'_m) \in \sigma\}} \Big] \tag{18b}$$

in which we plug in (11), and

$$\tilde{w}_{ij}^{(c)}(t) = \begin{cases} \frac{w_{ij}^{(c)} Q_i(t)}{C_i} & (\imath, \jmath) = (i_m, i_{m+1}), (i'_m, i_{m+1}) \\ \frac{w_{ij}^{(c)} Q_{ij}(t)}{C_{ij}} & (\imath, \jmath) = (i_m, j_m), (i'_m, j'_m) \end{cases} \tag{19}$$

with $\boldsymbol{w}^{(c)}$ given by (12).

The proposed algorithm is designed to minimize the upper bound (17) over the route selection decision $A(t)$ given by (8), or equivalently,

$$\min_{A(t)} \sum_{c} \sum_{\sigma \in \mathcal{F}_c(x)} O^{(c,\sigma)}(t)\, a^{(c,\sigma)}(t), \text{ s.t. (9).} \tag{20}$$

4.2.2 Route Selection

Given the linear structure of (20), the optimal route selection decision is given by:

$$a^{\star\,(c,\sigma)}(t) = a^{(c)}(t) \mathbf{1}_{\{\sigma = \sigma^\star\}} \tag{21}$$

where

$$\sigma^\star = \arg\min_{\sigma \in \mathcal{F}_c(x)} O^{(c,\sigma)}(t), \tag{22}$$

i.e., all requests of client c arriving at time t are delivered by the *min-ER*, i.e., the ER with the minimum weight, and the remaining problem is to find the min-ER among the exponential number of ERs in $\mathcal{F}_c(x)$.

To this end, we create a *weighted ALG* where each edge $(\imath, \jmath)$ in the ALG is assigned the weight $\tilde{w}_{ij}^{(c)}(t)$ given by (19), under which the weight of the ER σ (18b) equals to the sum of individual edge weights. In the rest of this section, we propose a dynamic programming algorithm to find the min-ER based on the weighted ALG.

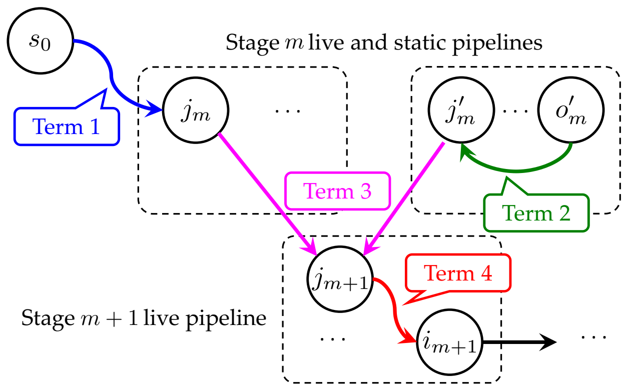


Fig. 7. Illustration of the weight components in Eq. (23).

Define:

- **ER weight matrix** W of size $(M_\phi + 1) \times |\mathcal{V}|$, where $W(m,i)$ is the *minimum* weight to deliver the stage m live packet to node i, *optimized over* all previous packet processing, routing, and replication decisions.
- **Processing location matrix** P of size $M_\phi \times |\mathcal{V}|$, where $P(m,i)$ is the optimal processing location of function m, to deliver the stage m live packet to node i.

The ultimate goal is to find $W(M_\phi, d)$ and the associated ER $\sigma^\star$, and we propose to derive W row-by-row (or layer-by-layer in the ALG). To be specific, suppose row m of W, i.e., $\{W(m,j) : j \in \mathcal{V}\}$, is given. Then, we can derive each element on row $m+1$, e.g., $W(m+1,i)$, in two steps:

First, assume that function $m+1$ is executed at node j. Then, we can optimize cache selection and routing decisions to minimize the weight, i.e.,

$$W_j(m+1,i) = W(m,j) + \mathrm{SPW}(o'_m, j'_m) \\ + \tilde{w}_{j_m j_{m+1}}^{(c)}(t) + \mathrm{SPW}(j_{m+1}, i_{m+1}) \tag{23}$$

where $\mathrm{SPW}(\imath, \jmath)$ denotes the weight of shortest path (SP) $\mathrm{SP}(\imath, \jmath)$ from node $\imath$ to $\jmath$ in the weighted ALG. As depicted in Fig. 7, the four terms represent: (i) the min-weight to deliver the stage m live packet to node j, (ii) the min-weight to replicate and route the stage m static packet to node j, (iii) the computation load at node j, and (iv) the min-weight to route the stage $m+1$ live packet to node i, respectively.

Second, we optimize the processing location decision to minimize the overall weight, i.e.,

$$W(m+1,i) = \min_{j \in \mathcal{V}} W_j(m+1,i), \tag{24a}$$

$$P(m+1,i) = \arg\min_{j \in \mathcal{V}} W_j(m+1,i). \tag{24b}$$

Repeat the above procedure to derive all entries of W and P. We then propose the following **back-tracing** procedure to derive the min-ER $\sigma^\star$, to *deliver the stage M_ϕ live packet to node d*: starting from destination d, the optimal processing location of function M_ϕ, $\theta^{(M_\phi)}$, is the (M_ϕ, d) entry of matrix P. The remaining problem is to find the optimal decisions to *deliver the stage $M_\phi - 1$ live packet to node $\theta^{(M_\phi)}$*, which has the same structure as the original problem and can be solved by repeating the above procedure, as described in Algorithm 1 (step 6 to 8).

Algorithm 1 Dynamic Programming to Find the min-ER

```
Input: $\tilde{\boldsymbol{w}}(t)$; Output: min-ER $\sigma^\star$, optimal weight $W^\star(x)$.
1: Initialization: $W(0,i) \leftarrow \mathrm{SPW}(s_0, i_0)$ for $\forall\, i \in \mathcal{V}$.
2: for $m = 0, \cdots, M_\phi - 1$ and $i \in \mathcal{V}$ do
3:     Calculate row $(m+1)$ for $W$ and $P$ by (24).
4: end for
5: Let $\theta^{(M_\phi)} = P(M_\phi, d)$ and $\sigma^\star = \mathrm{SP}([\theta^{(M_\phi)}]_{M_\phi}, d_{M_\phi})$.
6: for $m = M_\phi, \cdots, 1$ do
7:     $\theta^{(m-1)} \leftarrow P(m-1, \theta^{(m)})$ (note that $\theta^{(0)} = s$), and
       $\sigma^\star \leftarrow \sigma^\star \cup \mathrm{SP}(o'_{m-1}, [\theta^{(m)}]'_{m-1}) \cup ([\theta^{(m)}]'_{m-1}, [\theta^{(m)}]_m)$
         $\cup\, \mathrm{SP}([\theta^{(m-1)}]_{m-1}, [\theta^{(m)}]_{m-1}) \cup ([\theta^{(m)}]_{m-1}, [\theta^{(m)}]_m)$.
8: end for
9: Return (i) min-ER $\sigma^\star$ and (ii) $W^\star(x) = W(M_\phi, d)$.
```

4.3 Optimal Actual Network Decisions

Next, we present control decisions in the actual network.

We adopt the route selection decisions made in the virtual network in Section 4.2. In addition, we adopt the extended nearest-to-origin (ENTO) policy [19] for packet scheduling:

At each time slot t, for each node / link, give priority to the packets which have crossed the smallest number of edges in the ALG from the corresponding processing / transmission queue.

We note that ENTO is a *distributed* packet scheduling policy. The processing queue holds paired live and static packets, and we define the number of crossed hops for a packet-pair to be that of its live packet component.

To sum up, the proposed DI-DCNC algorithm is described in Algorithm 2.

Algorithm 2 DI-DCNC

```
1: for $t \geq 0$ do
2:     For each client $c$, all requests received at time $t$ get
       associated with the min-ER found by Algorithm 1.
3:     Each link transmits packets and each node processes
       paired-packets according to ENTO.
4:     Update the virtual queues by (14).
5: end for
```

4.4 Performance Analysis

4.4.1 Throughput

Under any fixed database placement, DI-DCNC is throughput optimal, as described in the following theorem.

Theorem 2. For any fixed database placement $x \in \mathcal{X}$ and arrival rate $\boldsymbol{\lambda}$ interior to the network stability region $\Lambda(x)$, all actual queues are rate stable under DI-DCNC.

Proof: See Appendix B. □

4.4.2 Complexity

We can take advantage of the following facts to simplify the calculation of (23) when implementing Algorithm 1:

$$\mathrm{SPW}(\imath, \jmath) = w_{ij}^{(c)}\, \mathrm{SPW}_0(i,j),\ \forall\, (\imath,\jmath) = (i_m, j_m),\ (i'_m, j'_m)$$

where $\mathrm{SPW}_0(i,j)$ denotes the SP distance in the weighted graph, which has the same topology as the actual network with the weight of each edge $(i,j) \in \mathcal{E}$ given by $Q_{ij}(t)/C_{ij}$. In addition, we note that

$$\mathrm{SPW}(o'_m, j'_m) = \min_{i \in \mathcal{V}(k_m^{(\phi)})} \mathrm{SPW}(i'_m, j'_m). \tag{25}$$

Therefore, we can implement Algorithm 1 as follows:

(i) Calculate the pairwise SP distance, i.e., $\{\mathrm{SPW}_0(i,j) : (i,j) \in \mathcal{V} \times \mathcal{V}\}$ by Floyd-Warshall [36, Section 25.2], with complexity $\mathcal{O}(|\mathcal{V}|^3)$.

(ii) In each iteration (step 3 in Algorithm 1): calculate $\mathrm{SPW}(o'_m, j'_m)$ for $\forall\, j \in \mathcal{V}$ by (25), with complexity $\mathcal{O}(|\mathcal{V}|^2)$. Then, calculate (23) for each (i,j) pair, with complexity $\mathcal{O}(|\mathcal{V}|^2)$. The total complexity to calculate the entire matrix is thus given by $\mathcal{O}(M_\phi |\mathcal{V}|^2)$.

(iii) Perform back-tracing, with complexity $\mathcal{O}(M_\phi)$.

To sum up, the overall complexity of Algorithm 1 is given by $\mathcal{O}(|\mathcal{V}|^3 + M_\phi |\mathcal{V}|^2)$.

4.4.3 Discussions on Delay Performance

As observed in the numerical experiments (in Section 7), the designed DI-DCNC algorithm can achieve good delay performance. Note however that we cannot say DI-DCNC is delay optimal because: (i) it places the focus on *queuing delay* without taking into account the *hop-distance* of the selected path, which can become an important delay component in low-congestion regimes; and (ii) the actual service delay should be taken as the *maximum* over the concurrent live and static pipelines, while the ER weight (18b) is indicative of the *aggregate* delay (i.e., the sum of two). Addressing these challenges is of interest for future work.

5 Max-Throughput Database Placement

The second part of this paper tackles the *processing-aware database placement* problem, with this section focusing on the setting of *fixed* database placement (and next section designing *dynamic* database replacement policies).

5.1 Problem Formulation

The goal is to design a fixed database placement policy to optimize the network's throughput performance, together with the flow (processing and routing) control decisions.

5.1.1 Variables

We define two variables, representing the database placement and flow control decisions, respectively, as follows:

- *Caching vector* $x = \{x_{i,k} : i \in \mathcal{V}, k \in \mathcal{K}\}$, where $x_{i,k}$ is a binary variable indicating if database k is cached at node i ($x_{i,k} = 1$) or not ($x_{i,k} = 0$).
- *Flow variables* $f = \{f_{\imath\jmath}^{(c)} : (\imath,\jmath) = (i_m, j_m) \in \mathcal{E}^{(\phi)}, c\}$ and $f' = \{f_{ij}^{\prime(k)} : (i,j) \in \mathcal{E}, k \in \mathcal{K}\}$, where $f_{\imath\jmath}^{(c)}$ denotes the flow rate of live packets of client c on edge $(\imath,\jmath) \in \mathcal{E}^{(\phi)}$ in the ALG, and $f_{ij}^{\prime(k)}$ the flow rate of static packets of database k on link $(i,j) \in \mathcal{E}$ in the actual network, respectively.[5]

5. The static flow defined in this section represents the sum of the individual static flows (on the same link (i,j) and of the same database k) *over all clients*, i.e., $f_{ij}^{\prime(k)} = \sum_{c,m} f_{i'_m j'_m} \mathbf{1}_{\{k_m^{(\phi)} = k\}}$.

5.1.2 Constraints

We impose two classes of constraints on the variables:

(i) **Capacity** constraints, which limit the 3C network resource usage, i.e., the incurred resource consumption shall not exceed the corresponding capacities. To be specific, the processing rate at each node $i \in \mathcal{V}$ and the transmission rate over each link $(i,j) \in \mathcal{E}$ must satisfy

$$\sum_{c,m} r_m^{(\phi)} f_{i_{m-1} i_m}^{(c)} \le C_i, \ \sum_{c,m} f_{i_m j_m}^{(c)} + \sum_{k \in \mathcal{K}} f_{ij}^{\prime(k)} \le C_{ij}, \quad (26)$$

and the storage constraint (2): $\sum_{k\in\mathcal{K}} F_k\, x_{i,k} \le S_i, \forall\, i \in \mathcal{V}$.

(ii) **Service chaining** constraints, which impose the relationship between input and output flows as they traverse network nodes and undergo service function processing. For live flows, the conservation law is given by:

$$\sum_{j\in\delta^+(i)} f_{i_m j_m}^{(c)} + f_{i_m i_{m+1}}^{(c)} = \sum_{j\in\delta^-(i)} f_{j_m i_m}^{(c)} + \xi_m^{(\phi)} f_{i_{m-1} i_m}^{(c)} + \lambda^{(c)} \mathbf{1}_{\{i_m = s_0\}}, \ \forall\, c, m, i : i_m \neq d_{M_\phi}, \quad (27)$$

and for the destination node:

$$f_{d_{M_\phi} j_{M_\phi}}^{(c)} = 0, \ \forall\, j \in \delta^+(d). \quad (28)$$

For the static flows (of database k), the conservation law can be summarized as

$$(1 - x_{i,k}) \Big(\sum_{j\in\delta^+(i)} f_{ij}^{\prime(k)} + f_i^{\prime(k)} - \sum_{j\in\delta^-(i)} f_{ji}^{\prime(k)} \Big) = 0, \quad (29)$$

with the processing rate of static packets at node i given by

$$f_i^{\prime(k)} \triangleq \sum_{c,m} \zeta_m^{(\phi)} f_{i_{m-1} i_m}^{(c)} \mathbf{1}_{\{k_m^{(\phi)} = k\}}. \quad (30)$$

The static flow conservation law (29) can be described as follows: *for each node i that is not a static source*, i.e., $x_{i,k} = 0$, the static flow of database k must satisfy the flow conservation constraint (see (6) for detailed illustration):

$$\sum_{j\in\delta^+(i)} f_{ij}^{\prime(k)} + f_i^{\prime(k)} = \sum_{j\in\delta^-(i)} f_{ji}^{\prime(k)}, \quad (31)$$

and (29) is true; *for any static source i*, i.e., $x_{i,k} = 1$ (and thus $1 - x_{i,k} = 0$), (29) is true. We note that (31) does not necessarily hold at the static sources, because they can perform in-network packet replication: an operation known to violate the flow conservation law [20].

5.1.3 Objective

We assume that the arrival rates of all clients' requests, $\{\lambda^{(c)} : \forall\, c\}$, are governed by a service request distribution. To be specific, the arrival rates are given by

$$\big\{ \lambda^{(c)} = p^{(c)} \lambda : \sum_c p^{(c)} = 1 \big\}, \quad (32)$$

and we use λ to measure the throughput performance. This objective is employed targeting better fairness performance, compared to another widely used metric of *sum arrival rate*, i.e., $\sum_c \lambda^{(c)}$, which favors service requests with lighter resource load (to improve the total throughput, provided the same network resources).

Remark 7. Note that the *service request distribution* defined above is w.r.t. clients $c = (s, d, \phi)$ (see Section 2.3). The *service popularity distribution*, which is w.r.t. services ϕ, can be derived as its marginal distribution. Furthermore, the *content popularity distribution*, which is w.r.t. databases k, can be derived based on the service popularity distribution and the associated service parameters (i.e., scaling factor and merging ratio).

Remark 8. If the actual service request distribution is unknown, a uniform distribution is used by default. Besides, other than representing the service request distribution, the values of $p^{(c)}$ can be designed for admission control, customer prioritization, etc.

5.2 Proposed Design

To sum up, the problem is formulated as follows:

$$\begin{aligned} &\max\ \lambda && (33a)\\ &\text{s.t.}\quad \lambda^{(c)} \ge p^{(c)}\lambda,\ \forall\, c && (33b)\\ &\qquad \text{Capacity constraints (26), (2)} && (33c)\\ &\qquad \text{Chaining constraints (27) – (30)} && (33d)\\ &\qquad x \in \{0,1\}^{|\mathcal{K}|\times|\mathcal{V}|} \text{ and } f, f' \succeq 0. && (33e) \end{aligned}$$

We note that (33) is a MIP problem due to (29), which is not a linear constraint due to the cross terms of x and f'. To improve tractability, we propose to replace (29) with the following linear constraint:

$$\sum_{j\in\delta^+(i)} f_{ij}^{\prime(k)} + f_i^{\prime(k)} - \sum_{j\in\delta^-(i)} f_{ji}^{\prime(k)} \le C_{i,k}^{\max} x_{i,k} \quad (34)$$

where $C_{i,k}^{\max}$ is a constant, given by

$$C_{i,k}^{\max} = \sum_{j\in\delta^+(i)} C_{ij} + C_i \max_{k_m^{(\phi)} = k} \big(\zeta_m^{(\phi)} / r_m^{(\phi)} \big). \quad (35)$$

We claim that the resulting MILP problem:

$$\max\ \lambda,\ \text{s.t. } (33b), (33c), (27), (28), (34), (35), (30), (33e) \quad (36)$$

has the same optimal solution as (33).

Proof: See Appendix E. □

In general, the MILP problem (36) is still NP-hard, which can incur high complexity to find the exact solution. However, there are many software toolboxes designed to deal with general MILP problems, which can find approximate solutions that trade off accuracy with running time. For example, we use the widely adopted `intlinprog` function in MATLAB to implement the proposed design. In addition, it can serve as a good starting point for future studies to design approximation algorithms.

5.3 Performance Analysis

In this section, we present an equivalent characterization of the stability region under a given database placement.

Proposition 2. For any fixed database placement $x \in \mathcal{X}$, an arrival vector $\boldsymbol{\lambda}$ is interior to the stability region $\Lambda(x)$ if and only if there exist flow variables $f, f' \succeq 0$ satisfying (26) – (30).

Proof: In Appendix C, we show that the sets of $\boldsymbol{\lambda}$ described in this proposition and Theorem 1 are equal, completing the proof. Furthermore, the result also applies to Markov-modulated arrivals, as Theorem 1 does. □

Proposition 2 shows that the proposed database placement policy can achieve max-throughput.

6 Database Replacement Policies

In this section, we show that the benefit of database replacement to enlarge the stability region remains unchanged when the communication rate for database replacement (referred to as *replacement rate*) is restricted, using the proposed *time frame* structure, under which we develop two replacement policies to handle time-varying service demand statistics.

6.1 Low-Rate Replacement

In Section 3.3, we illustrated the benefit of database replacement assuming that replacement can be performed instantaneously, which can impose a high requirement on the replacement rate. In the following proposition, we show that the same throughput performance can be achieved under arbitrarily low replacement rate.

Proposition 3. For any arrival vector interior to the stability region, there exists a replacement policy achieving an $[\mathcal{O}(T), \mathcal{O}(1/T)]$ tradeoff between average virtual queue backlog and replacement rate.

Proof: See Appendix D.2.2. We propose the time frame structure and design a reference policy (including T as a parameter), shown to achieve the tunable performance. □

In the reference policy, we consider a two-timescale system, where processing and transmission decisions are made on a per time slot basis, while database replacement decisions are made on a per time frame basis, with each frame $\mathcal{T}$ including T consecutive slots. The replacement is launched at the beginning of each frame, which must be completed by the end of the frame. The policy is throughput-optimal for any given T, and the required replacement rate can be arbitrarily close to zero by pushing $T \to \infty$, with a tradeoff in queue backlog (and thus delay performance).

In the rest of the section, we adopt the time frame structure to design two heuristic database replacement policies, based on estimated *service request distribution* and *database score*, respectively. We note that the proposed design can flexibly incorporate advanced prediction techniques, which plays an equivalent role to estimation, but can enhance the timeliness of the quantities.[6]

6.2 Rate-Based Replacement

The first policy takes advantage of the max-throughput database placement policy described in Section 5. To handle time-varying demand statistics, we calculate the empirical service request distribution over each frame $\mathcal{T}$ as follows

$$\hat{p}^{(c)} = \frac{\sum_{t\in\mathcal{T}} a^{(c)}(t)}{\sum_{c'} \sum_{t\in\mathcal{T}} a^{(c')}(t)}. \tag{37}$$

We then solve the MILP problem (36) based on $\{\hat{p}^{(c)}\}$ to derive the updated database placement, and each node can perform the replacement accordingly.

While straightforward, this policy exhibits three limitations. First, it neglects the existing resource loads in the network, which can lead to sub-optimal solution. Second, the updated database placement is designed independent with the current placement, which can impose a high requirement on the replacement rate. Finally, it requires solving the MILP problem in an online manner, which can reduce the accuracy of the approximate solution.

6. We note that an estimation-based policy derives estimates over frame τ and executes the replacement decisions during frame $\tau + 1$. The new placement will take effect in frame $\tau + 2$.

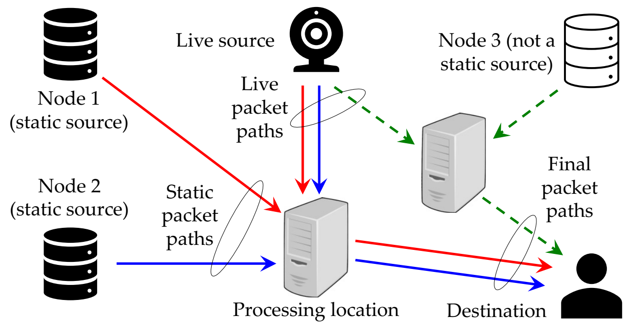


Fig. 8. The min-ERs (denoted by red, blue, and green arrows) for the delivery of an AR service over the network, assuming nodes 1, 2, and 3 are selected to provision the static packet, respectively.

6.3 Score-Based Replacement

The second policy is motivated by the "min-ER" rule for route selection (derived in Section 4.2), in which we propose to evaluate the benefit for each node i to cache database k by *database score* (or *score* for brevity), defined as follows.

Definition 3 (Score). For each instance, i.e., a given request ψ and network states (queuing states, database placement), the score of database k at node i is the difference of the min-ER weights assuming node i does not cache the database, and the opposite, i.e.,

$$u_{i,k}(\psi) = W^\star(x^-(i,k)) - W^\star(x^+(i,k)) \tag{38}$$

where $W^\star(x)$ is the min-ER weight given by Algorithm 1; $x^-(i,k)$ and $x^+(i,k)$ denote caching vectors equal to x, but with $x_{i,k} = 0$ and $x_{i,k} = 1$, respectively.

In particular, for a given database k: for a static source node, the score is the *increment* of min-ER weight if it does not cache database k; otherwise, the score is the *reduction* of min-ER weight if the node caches database k.

We illustrate the definition by the example in Fig. 8. Let W_1, W_2, and W_3 denote the min-ER weights assuming that node 1, 2, and 3 are selected to provision the static packet, respectively (note that node 3 is not a static source, and it is assumed to cache the database to derive the green ER and associated weight W_3), with $W_3 < W_1 < W_2$. Then, node 1 is selected as the static source, serving as the benchmark. The database score at each node is derived as follows. Node 1: if it *does not* cache database k, node 2 will be selected, leading to a greater weight W_2, and thus $u_{1,k} = W_2 - W_1$. Node 2: if it *does not* cache database k, the cache selection decision does not change, leading to the same weight W_1, and thus $u_{2,k} = W_1 - W_1 = 0$. Node 3: if it *caches* database k, node 3 will be selected as the static source, leading to a reduced weight W_3, and thus $u_{3,k} = W_1 - W_3$.

We note that the above definition of score (i) assumes unchanged caching policies at the other network nodes, (ii) requires finding for $x^-(i,k)$ and $x^+(i,k)$ the corresponding

TABLE 2
Clients, i.e., (source, destination, service), and Service Function Specs, i.e., (scaling factor, workload [GHz/Gbps], object name, merging ratio)

Client (s, d, ϕ)	$(1, 9, \phi_1)$	$(3, 7, \phi_2)$	$(7, 3, \phi_3)$	$(9, 1, \phi_4)$
Func 1 $(\xi_1^{(\phi)}, r_1^{(\phi)}, k_1^{(\phi)}, \zeta_1^{(\phi)})$	(0.83, 7.1, 1, 0.92)	(0.94, 10.0, 3, 0.52)	(0.75, 8.7, 5, 1.48)	(0.60, 8.4, 7, 0.91)
Func 2 $(\xi_2^{(\phi)}, r_2^{(\phi)}, k_2^{(\phi)}, \zeta_2^{(\phi)})$	(1.06, 5.8, 2, 1.06)	(1.22, 7.7, 4, 0.65)	(1.31, 9.2, 6, 1.97)	(1.34, 7.4, 8, 1.22)

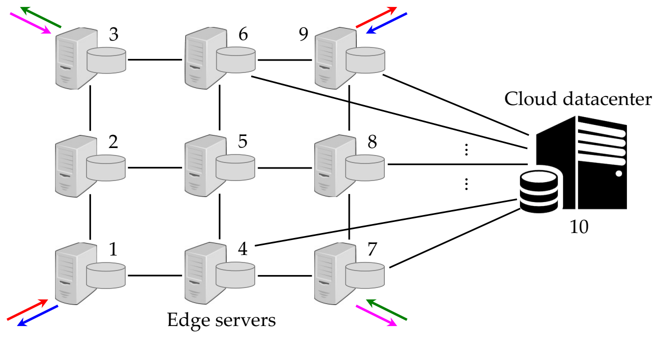


Fig. 9. The studied edge cloud network, including 9 edge servers (node 1 to 9) and a cloud datacenter (node 10). Arrows of the same color indicate the source-destination pairs of each client.

min-ERs (which, in particular, can include different processing locations), and (iii) accounts for a single instance, and the sum score of all instances within a time frame is a proper metric to evaluate the overall score, i.e.,

$$U_{i,k} = \sum_{t \in \mathcal{T}} \sum_{\psi \in \mathcal{A}(t)} u_{i,k}(\psi) \tag{39}$$

where $\mathcal{A}(t)$ denotes the received requests at time t.

Given the obtained scores, we formulate an optimization problem with the goal of *maximizing the total score* to find the updated database placement for each node $i \in \mathcal{V}$, i.e.,

$$\max_{x_i \in \{0,1\}^{|\mathcal{K}|}} \sum_{k \in \mathcal{K}} U_{i,k}\, x_{i,k}, \text{ s.t. } \sum_{k \in \mathcal{K}} F_k\, x_{i,k} \le S_i. \tag{40}$$

The above problem, known as *0/1 knapsack problem*, admits a dynamic programming solution with pseudo-polynomial complexity $\mathcal{O}(|\mathcal{K}|S_i)$ [37]. Let $U_i^\star$ and $x_i^\star$ denote the optimal value and solution, respectively.

Finally, we find the node with the largest total score, i.e., $i^\star = \arg\max_i U_i^\star$, and *only* replace its databases according to $x_{i^\star}^\star$, which is referred to as *asynchronous* update, in line with the definition of score assuming *unchanged* caching policies at the other network nodes.

There are three factors that can impact the performance of this policy. First, the proposed score metric focuses on each individual node (for tractability), and cannot capture the coupling between them. Second, we use the observed queuing states to calculate the score, which in turn are impacted by the database placement. Finally, the asynchronous update can be less efficient for database replacement and lead to slower adaptation.[7]

7. Multiple nodes, whose database scores are calculated independently, can update their caching policies synchronously. While out of this paper's scope, this extension is promising to accelerate adaptation.

7 Numerical Results

As mentioned in Section 1.3, the methodology proposed in this paper targets the general class of data-intensive AgI services, and the main goal of this section is to illustrate 1) the effects of multi-pipeline flow control, and 2) tradeoffs among 3C network resources, under the proposed methodology to validate its strength and applicability to a wide range of scenarios.

7.1 Experiment Setup

Consider a mesh MEC network composed of 9 edge servers and a cloud datacenter, connected by wired links, as shown in Fig. 9. Each edge server is equipped with 4 processors of frequency 2.5 GHz, and each link between them has 1 Gbps transmission capacity. The cloud datacenter is equipped with 8 identical processors; it is connected to all edge servers, and each link has 20 Mbps transmission capacity.[8] The length of each time slot is 1 ms.

There are $|\mathcal{K}| = 8$ databases, and each database has the same size of $F = 1$ Gb. In the following, we quantify the storage capacity of edge servers in *number of databases*. Assume that the cloud datacenter has all databases stored.

Consider 4 clients requesting different services. Each service is composed of 2 functions, with parameters shown in Table 2. In line with the aforementioned goal of experiment validation, we employ a representative range of input parameters, including distributed source-destination pairs, multiple processing functions, different scaling/workload/merging ratios, and substantial distribution of databases. The size of each packet is 1 kb, and the arrivals are modeled by i.i.d. Poisson processes with λ Mbps.

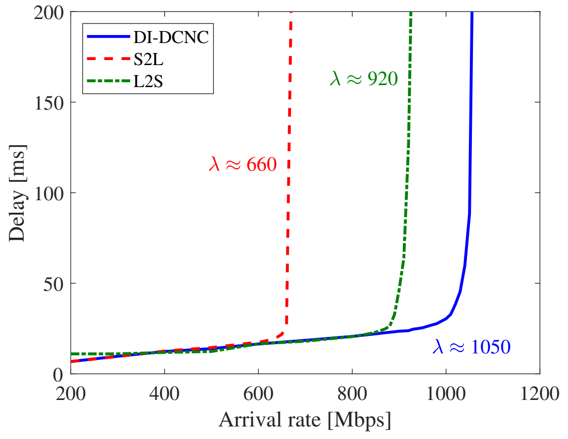
7.2 Multi-Pipeline Flow Control

We first demonstrate the performance of DI-DCNC under a given database placement, where database $k = 1, \cdots, 8$ are stored at node $i = 1, \cdots, 4, 6, \cdots, 9$, respectively. Two benchmark algorithms are employed for comparison:[9]

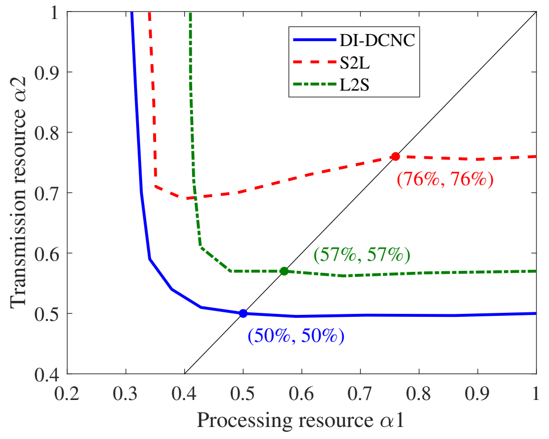
- *Static-to-live (S2L)*, which makes individual routing decisions for the live packet [19], and then routes the static packet to the selected processing node from the *nearest* static source along the *shortest path* (in the weighted ALG).
- *Live-to-static (L2S)*, which makes routing decisions for the live packet by restricting processing locations to the static sources (and use local static packet).

8. The communication resources are dedicated for the delivery of packets belonging to service requests. Requirements for (database) replacement rate are depicted in Fig. 12b.

9. Both S2L and L2S do not consider joint control of multi-pipelines: they focus on the communication loads of either live or static packets.



(a) Network stability region.



(b) Resource occupation.

Fig. 10. Performance of DI-DCNC (under a given database placement).

7.2.1 *Network Stability Region*

First, we study network stability regions attained by the algorithms, and depict the average delay under different arrival rates (which is set to ∞ if the queues are not stable).

As shown in Fig. 10a, DI-DCNC attains good delay performance over a wide range of arrival rates; when λ crosses a critical point (≈ 1.05 Gbps), the average delay blows up, indicative of the stability region boundary. Similar behaviors are observed from S2L and L2S. Comparing the three algorithms, DI-DCNC outperforms the benchmark algorithms in terms of the achieved throughput: 1.05 Gbps (DI-DCNC) $>$ 920 Mbps (L2S) $>$ 660 Mbps (S2L); in other words, DI-DCNC can better exploit network resources to improve the throughput. We clarify that the throughput attained by S2L improves when increasing the communication resources, and can outperform L2S [1].

We also notice that the delay attained by DI-DCNC is very similar, but not lower, than the benchmarks in low-congestion regimes. As discussed in Section 4.4.3, DI-DCNC is designed to reduce the *aggregate queuing* delay of both live and static data pipelines; such objective, while closely related to (especially in high-congestion regimes), is not exactly equivalent to the actual service delay, which depends on the *maximum* delay between the two concurrent pipelines. In addition, DI-DCNC neglects the hop-distance of the selected path, which is the dominant component in the low-congestion regimes.

7.2.2 *Resource Occupation*

Next, we study the resource occupation of the algorithms. We assume that the available processing and transmission capacities of each node and link are given by α_1 and α_2 (in *percentage*) of corresponding maximum budgets, respectively. We then define the *feasible region* as the collection of (α_1, α_2) pairs under which the delay requirement is fulfilled. Let arrival rate $\lambda = 500$ Mbps and average delay requirement $= 30$ ms.

Fig. 10b depicts the feasible regions attained by the algorithms. Since lower latency can be attained with more resources, i.e., $(\alpha_1, \alpha_2) \to (1, 1)$, the feasible regions are to the upper-right of the border lines.[10] As we can observe, DI-DCNC can save the most network resources. In particular, when $\alpha_1 = \alpha_2 = \alpha$, the resource saving ratios, i.e., $1 - \alpha$, of the algorithms are: 50% (DI-DCNC) $>$ 43% (S2L) $>$ 24% (L2S). Another observation is: S2L is communication-constrained, compared to its sensitivity to computation resources. To wit: in the horizontal direction (when $\alpha_2 = 1$), it can achieve a maximum saving ratio of $\approx 70\%$, which is comparable to DI-DCNC; while the maximum saving ratio is $\approx$ 25% for communication resources (when $\alpha_1 = 1$), and there is a large gap between its performance and that of DI-DCNC ($\approx 50\%$). The reason is that S2L neglects the communication load of static packet routing, leading to additional communication resource consumption. In contrast, L2S is processing-constrained, because only the processing resources at static sources are available for use.

7.3 Processing-aware Database Placement

Next, we evaluate the proposed database placement and replacement policies, employing DI-DCNC for flow control.

7.3.1 *Fixed Placement Policy*

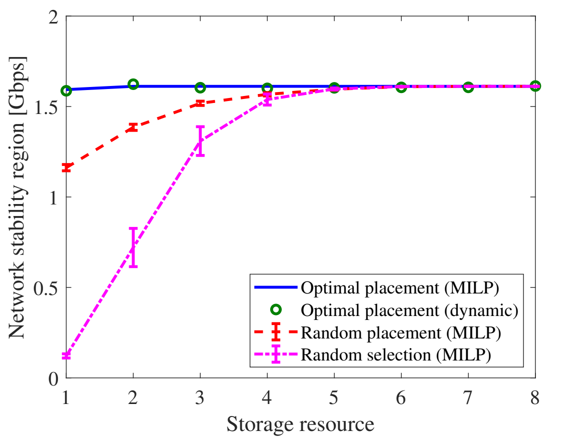
First, we focus on the setting of fixed database placement, assuming that each edge server can cache S databases. We evaluate the proposed max-throughput policy, employing two random policies as benchmarks:

- *Random selection:* each edge server randomly selects S different databases to cache.
- *Random placement:* each edge server caches S different databases, which are jointly selected to maximize the diversity of databases stored at edge servers.[11]

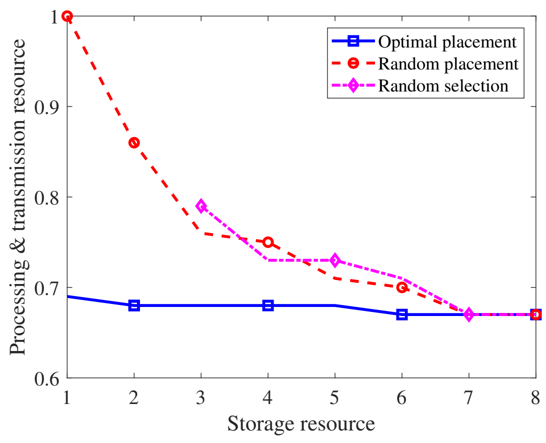
Similar performance metrics, i.e., network stability region and resource occupation, are studied, and the results are shown in Fig. 11.

10. We note that the feasible region achieved by S2L is not convex.

11. The cached databases at the edge servers are selected as follows. Generate a random permutation of sequence $\{1, \cdots, |\mathcal{K}|\}$ and repeat it for $\lceil |\mathcal{V}|S/|\mathcal{K}| \rceil$ times. Let $\mathcal{D}$ and $\mathcal{D}_i$ denote the resulting sequence and its ith element, respectively. Then, database $\mathcal{D}_{(i-1)S+1}, \cdots, \mathcal{D}_{iS}$ are cached at edge server i (note that these are different databases since every sub-sequence in $\mathcal{D}$ of length $S \le |\mathcal{K}|$ includes distinct values).



(a) Network stability region (error-bars denote standard deviations).



(b) Resource occupation.

Fig. 11. Performance of max-throughput database placement policy.

(i) *Network Stability Region:* Fig. 11a shows the throughput performance of the three policies: for the proposed policy, we solve the MILP problem (36) and present the obtained result, as well as the observed stability region under the derived max-throughput database placement; for the benchmark policies, we plug in randomly generated placement x into (36), solve the remaining linear programs, and present averages and standard deviations of the results (in 100 realizations).

As we can observe, for each policy, the attained throughput grows with storage resource. Among the three policies, the proposed max-throughput policy outperforms the random benchmarks, especially when the storage resource is limited. For example, when $S = 1$, the proposed policy achieves the highest throughput ($\approx$ 1.59 Gbps), which is 37% better than random placement and far beyond random selection. The results validate the effects of caching policy design on the throughput performance, including (i) which databases to cache (comparing random placement and random selection), and (ii) where to store the databases (comparing proposed policy and random placement). Finally, we note that results given by the MILP (36) agree with the observed stability regions, validating Proposition 2.

(ii) *Resource Occupation:* Next, we study the tradeoff between 3C network resources, assuming $\lambda = 1$ Gbps, average delay requirement = 30 ms, and $\alpha_1 = \alpha_2 = \alpha$. For each random benchmark, we select a representative placement that attains a throughput performance closest to the corresponding mean value (shown in Fig. 11a).

As we can observe in Fig. 11b, increasing the storage resource at the edge servers leads to a larger computation/communication resource saving ratio. In particular, the performance of the policies converges when each node has sufficient space to cache all databases (i.e., $S = |\mathcal{K}|$). When the storage resource is limited (e.g., $S = 1$), the proposed policy can achieve a computation/communication resource saving ratio (69%) that is close to the optimal value ($\approx$ 67%), outperforming the random benchmarks.

7.3.2 Replacement Policies

Finally, we evaluate the proposed database replacement policies. For each client, we model the arrivals of service requests by a Markov-modulated process, i.e., the arrival rate follows a Markov process (described in the following), and the number of arrivals at a single time slot is a Poisson variable. At each time slot, the service request distribution is a *permutation* of the Zipf distribution with $\gamma = 1$ [13]: we sort the clients by the arrival rate in descending order, and w.p. 10^{-6}, we randomly select $\varphi \in \{1, 2, 3\}$ and exchange the arrival rates of the φ-th and $(\varphi + 1)$-th clients.[12]

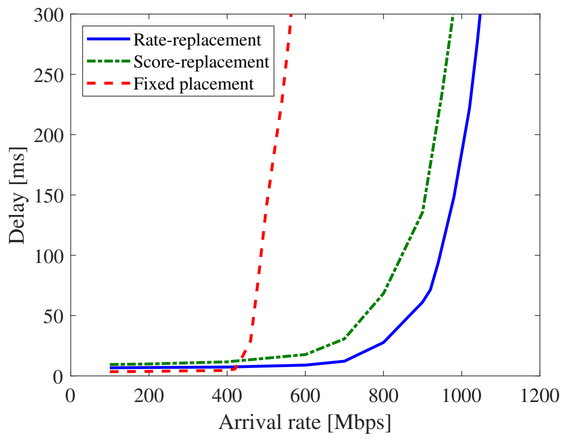
Each edge server, except node $i = 4, 5, 6$, is allowed to cache $S = 1$ database.[13] The initial database placement is given by the proposed max-throughput policy assuming uniform service request distribution. The two proposed replacement policies are evaluated. For fair comparisons, we set the same running time for both of them (to calculate estimated quantities and solve corresponding problems).

Fig. 12a shows the throughput performance of the two replacement policies, both of which effectively boost the throughput performance ($\approx$ 1 Gbps) compared to fixed placement ($\approx$ 0.5 Gbps), despite the slightly worse delay performance in low-congestion regimes (e.g., $\lambda \leq 400$ Mbps). Fig. 12b demonstrates the effects of time frame size on the policies' throughput performance and replacement rate requirements (i.e., the average downloading rate from the cloud datacenter to all edge servers). For each policy, as frame size grows, the frequency of database replacement reduces, leading to reduced replacement rate, with sub-optimal throughput.[14] Comparing the two proposed policies, we find that: rate-based policy achieves better throughput, which is also less sensitive to the frame size (note that the blue-solid curve is more flat); while score-based policy

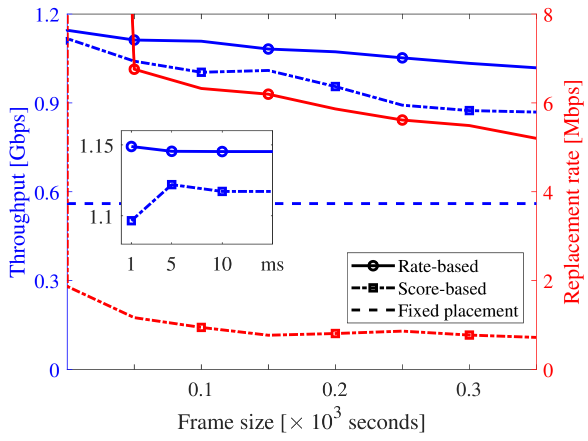
12. Under this setting, the expected time for service request distribution to change is 10^6 ms $\approx$ 15 min. We note that the time average arrival rate for all clients are equal (i.e., uniform distribution).

13. Under this setting, the total storage resources at the edge servers are 6, which cannot support caching all the databases (since $|\mathcal{K}| = 8$).

14. The time frame size controls the tradeoff between the accuracy and timeliness of the estimates, as can be observed from the score-based policy (the frame size of 5 ms leads to the largest throughput).



(a) Network stability regions (frame size = 100 s).



(b) Effects of frame size on throughput and replacement rate.

Fig. 12. Performance of rate- and score-based replacement policies.

imposes a much lower requirement on replacement rate, due to the asynchronous update of database placement that is designed depending on current network states.

8 Conclusions

We investigated the problem of joint 3C control for the efficient delivery of data-intensive services, composed of multiple service functions with multiple (live/static) input streams. We first characterized network stability regions based on the proposed ALG model, which incorporates multiple pipelines for input streams. Two problems are addressed: (i) multi-pipeline flow control, in which we derived a throughput-optimal policy, DI-DCNC, to coordinate packet processing, routing, and replication decisions for multiple pipelines, and (ii) processing-aware database placement, in which we proposed a max-throughput database placement policy by jointly optimizing 3C decisions, as well as the rate- and score-based database replacement policies. Via numerical experiments, we demonstrated the superior performance of multiple pipeline coordination and integrated 3C design in delivering next-generation data-intensive real-time stream-processing services.

References

[1] Y. Cai, J. Llorca, A. M. Tulino, and A. F. Molisch, "Dynamic control of data-intensive services over edge computing networks," in *Proc. IEEE Global. Telecomm. Conf.*, Rio de Janeiro, Brazil, Dec. 2022, pp. 5123–5128.

[2] H. Feng, J. Llorca, A. M. Tulino, and A. F. Molisch, "On the delivery of augmented information services over wireless computing networks," in *Proc. IEEE Int. Conf. Commun.*, Paris, France, May 2017, pp. 1–7.

[3] Y. Cai, J. Llorca, A. M. Tulino, and A. F. Molisch, "Compute- and data-intensive networks: The key to the Metaverse," in *2022 1st International Conference on 6G Networking (6GNet)*, Paris, France, Jul. 2022, pp. 1–8.

[4] "The programmable cloud network – A primer on SDN and NFV," Alcatel-Lucent, Paris, France, White Paper, Jun, 2013. [Online]. Available: https://whitepapers.us.com/.

[5] Y. Sun, Z. Chen, M. Tao, and H. Liu, "Communications, caching, and computing for mobile virtual reality: Modeling and tradeoff," *IEEE Trans. Commun.*, vol. 67, no. 11, pp. 7573–7586, Nov. 2019.

[6] W. Xu, Y. Shen, N. Bergmann, and W. Hu, "Sensor-assisted multi-view face recognition system on smart glass," *IEEE Trans. Mobile Comput.*, vol. 17, no. 1, pp. 197–210, Jan. 2018.

[7] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 587–597, Mar. 2018.

[8] Y. Cai, J. Llorca, A. M. Tulino, and A. F. Molisch, "Mobile edge computing network control: Tradeoff between delay and cost," in *Proc. IEEE Global. Telecomm. Conf.*, Taipei, Taiwan, Dec. 2020, pp. 1–6.

[9] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Service placement and request routing in MEC networks with storage, computation, and communication constraints," *IEEE/ACM Trans. Netw.*, vol. 28, no. 3, pp. 1047–1060, Jun. 2020.

[10] M. Huang, W. Liang, Y. Ma, and S. Guo, "Maximizing throughput of delay-sensitive NFV-enabled request admissions via virtualized network function placement," *IEEE Trans. Cloud Comput.*, vol. 9, no. 4, pp. 1535–1548, Oct.-Dec. 2021.

[11] Y. Yue, B. Cheng, M. Wang *et al.*, "Throughput optimization and delay guarantee VNF placement for mapping SFC requests in NFV-enabled networks," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 4, pp. 4247–4262, Dec. 2021.

[12] M. Barcelo, A. Correa, J. Llorca, A. M. Tulino, J. L. Vicario, and A. Morell, "IoT-cloud service optimization in next generation smart environments," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 4077–4090, Oct. 2016.

[13] K. Poularakis, J. Llorca, A. M. Tulino, and L. Tassiulas, "Approximation algorithms for data-intensive service chain embedding," in *Mobihoc '20*, Virtual Event, USA, Oct. 2020, pp. 131–140.

[14] Y. Yue, B. Cheng, X. Liu, M. Wang, B. Li, and J. Chen, "Resource optimization and delay guarantee virtual network function placement for mapping SFC requests in cloud networks," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 2, pp. 1508–1523, Jun. 2021.

[15] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *IEEE Trans. Autom. Control*, vol. 37, no. 12, pp. 1936–1948, Dec. 1992.

[16] M. J. Neely, *Stochastic network optimization with application to communication and queueing systems*. San Rafael, CA, USA: Morgan & Claypool, 2010.

[17] A. Sinha and E. Modiano, "Optimal control for generalized network flow problems," *IEEE/ACM Trans. Netw.*, vol. 26, no. 1, pp. 506–519, Feb. 2018.

[18] 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.

[19] 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 Trans. Netw.*, vol. 29, no. 4, pp. 1760–1773, Aug. 2021.

[20] Y. Cai, J. Llorca, A. M. Tulino, and A. F. Molisch, "Decentralized control of distributed cloud networks with generalized network flows," arXiv:2204.09030. [Online]. Available: https://arxiv.org/abs/2204.09030, Apr. 2022.

[21] M. Ji, G. Caire, and A. F. Molisch, "Wireless device-to-device caching networks: Basic principles and system performance," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 1, pp. 176–189, Jan. 2016.

[22] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, "FemtoCaching: Wireless content delivery through distributed caching helpers," *IEEE Trans. Inf. Theory*, vol. 59, no. 12, pp. 8402–8413, Dec. 2013.

[23] N. Golrezaei, A. F. Molisch, A. G. Dimakis, and G. Caire, "Femtocaching and device-to-device collaboration: A new architecture for wireless video distribution," *IEEE Commun. Mag.*, vol. 51, no. 4, pp. 142–149, Apr. 2013.

[24] M. Gregori, J. Gómez-Vilardebó, J. Matamoros, and D. Gündüz, "Wireless content caching for small cell and D2D networks," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 5, pp. 1222–1234, May 2016.

[25] Z. Ding, P. Fan, G. K. Karagiannidis, R. Schober, and H. V. Poor, "NOMA assisted wireless caching: Strategies and performance analysis," *IEEE Trans. Commun.*, vol. 66, no. 10, pp. 4854–4876, Oct. 2018.

[26] A. Liu and V. K. N. Lau, "Exploiting base station caching in MIMO cellular networks: Opportunistic cooperation for video streaming," *IEEE Trans. Signal Process.*, vol. 63, no. 1, pp. 57–69, Jan. 2015.

[27] M. Ji, A. M. Tulino, J. Llorca, and G. Caire, "Order-optimal rate of caching and coded multicasting with random demands," *IEEE Trans. Inf. Theory*, vol. 63, no. 6, pp. 3923–3949, Jun. 2017.

[28] G. S. Paschos, G. Iosifidis, M. Tao, D. Towsley, and G. Caire, "The role of caching in future communication systems and networks," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 6, pp. 1111–1125, Jun. 2018.

[29] B. Liu, K. Poularakis, L. Tassiulas, and T. Jiang, "Joint caching and routing in congestible networks of arbitrary topology," *IEEE Internet Things J.*, vol. 6, no. 6, pp. 10 105–10 118, Dec. 2019.

[30] S. Ioannidis and E. Yeh, "Jointly optimal routing and caching for arbitrary network topologies," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 6, pp. 1258–1275, Jun. 2018.

[31] Y. Zhou, F. R. Yu, J. Chen, and Y. Kuo, "Communications, caching, and computing for next generation HetNets," *IEEE Wirel. Commun.*, vol. 25, no. 4, pp. 104–111, Aug. 2018.

[32] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, Fourthquarter 2017.

[33] A. Ndikumana, N. H. Tran, T. M. Ho, Z. Han, W. Saad, D. Niyato, and C. S. Hong, "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.

[34] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 1, pp. 856–868, Jan. 2019.

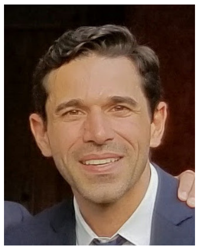
[35] K. Kamran, E. Yeh, and Q. Ma, "DECO: Joint computation scheduling, caching, and communication in data-intensive computing networks," *IEEE/ACM Trans. Netw.*, vol. 1, no. 99, pp. 1–15, 2021.

[36] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. Cambridge, MA, USA: MIT press, 2009.

[37] J. Kleinberg and Éva Tardos, Algorithm design. Delhi, India: Pearson Education India, 2006.

Yang Cai (S'17) received B.E. and M.S. degrees from the Department of Electronic Engineering, Tsinghua University, in 2015 and 2018, respectively. Now he is a Ph.D. candidate in the Department of Electrical Engineering at University of Southern California, with the Annenberg Graduate Fellowship award from 2018 to 2022.

He is currently working on modeling, analysis, evaluation, and control of next-generation computing networks (e.g., distributed cloud, mobile edge, and fog computing) and services (e.g., autonomous driving, augmented/virtual reality, and industrial automation).



Jaime Llorca (S'03–M'09) is a Technology Consultant and Research Professor with the NYU Tandon School of Engineering, NY. He previously held a Senior Research Scientist position with the Network Algorithms Group at Nokia Bell Labs, NJ, a Research Scientist position with the End-to-End Networking Group at Alcatel-Lucent Bell Labs, NJ, and a post-doctoral position with the Center for Networking of Infrastructure Sensors, MD. He received M.S. and Ph.D. degrees in Electrical and Computer Engineering from the University of Maryland, College Park, MD. His research interests are in the field of network algorithms, optimization, machine learning, and distributed control, with applications to next generation communication networks, distributed/edge cloud, and content distribution. He has made fundamental contributions to the mathematics of content delivery and distributed cloud networks, including pioneering cooperative caching, network coding, and cloud network control algorithms. He has authored more than 100 peer-reviewed publications and 20 patents. He currently serves as Associate Editor for the IEEE/ACM Transactions on Networking. He is a recipient of the 2007 IEEE ISSNIP Best Paper Award, the 2016 IEEE ICC Best Paper Award, and the 2015 Jimmy H.C. Lin Award for Innovation.

Antonia M. Tulino (F'13) is a full professor at Università degli Studi di Napoli Federico II. She held research positions with the Center for Wireless Communications in Oulu, Princeton University, a Università degli Studi del Sannio, Italy and Bell Labs, NJ. Since 2019, she is Research Professor at Dep. of Electrical and Computer Engineering NYU Tandon School of Engineering, also director of the 5G Academy jointly organized by the Università degli Studi di Napoli, Federico II and several leading company in digital transformation. Her research interests lay in the area of communication networks approached with the complementary tools provided by signal processing, information theory, and random matrix theory. From 2011 to 2013, she has been a member of the Editorial Board of the IEEE Transactions on Information Theory and in 2013, she was elevated to IEEE Fellow. From 2019 to 2021, she has been chair of the Information Theory society Fellows Committee.

She has received several paper awards, including the 2009 Stephen O. Rice Prize in the Field of Communications Theory. She was recipient of the UC3M-Santander Chair of Excellence from 2018 to 2019 and selected by the National Academy of Engineering for the Frontiers of Engineering program in 2013.

Andreas F. Molisch (S'89–M'95–SM'00–F'05) received his degrees (Dipl.Ing. 1990, PhD 1994, Habilitation 1999) from the Technical University Vienna, Austria. He spent the next 10 years in industry, at FTW, AT&T (Bell) Laboratories, and Mitsubishi Electric Research Labs (where he rose to Chief Wireless Standards Architect). In 2009 he joined the University of Southern California (USC) in Los Angeles, CA, as Professor, and founded the Wireless Devices and Systems (WiDeS) group. In 2017, he was appointed to the Solomon Golomb – Andrew and Erna Viterbi Chair.

His research interests revolve around wireless propagation channels, wireless systems design, and their interaction. Recently, his main interests have been wireless channel measurement and modeling for 5G and beyond 5G systems, joint communication-caching-computation, hybrid beamforming, UWB/TOA based localization, and novel modulation/multiple access methods. Overall, he has published 5 books (among them the textbook "Wireless Communications", third edition in 2022), 21 book chapters, 280 journal papers, and 370 conference papers. He is also the inventor of 70 granted (and more than 10 pending) patents, and co-author of some 70 standards contributions. His work has been cited more than 59,000 times, his h-index is > 100, and he is a Clarivate Highly Cited Researcher.

Dr. Molisch has been an Editor of a number of journals and special issues, General Chair, Technical Program Committee Chair, or Symposium Chair of multiple international conferences, as well as Chairperson of various international standardization groups. He is a Fellow of the National Academy of Inventors, Fellow of the AAAS, Fellow of the IEEE, Fellow of the IET, an IEEE Distinguished Lecturer, and a member of the Austrian Academy of Sciences. He has received numerous awards, among them the IET Achievement Medal, the Technical Achievement Awards of IEEE Vehicular Technology Society (Evans Avant-Garde Award) and the IEEE Communications Society (Edwin Howard Armstrong Award), and the Technical Field Award of the IEEE for Communications, the Eric Sumner Award.