Optimal Dynamic Orchestration in NDN-based Computing Networks

Hao Feng, Yi Zhang, Srikathyayani Srikanteswara, Marcin Spoczynski, Gabriel Arrobo, Jing Zhu, Nageen Himayat

Intel Labs, USA, Email: {hao.feng, yi.zhang, srikathyayani.srikanteswara, marcin.spoczynski, gabriel.arrobo, jing.z.zhu, nageen.himayat}@intel.com

Abstract—Named Data Networking (NDN) offers promising advantages in deploying next-generation service applications over distributed computing networks. We consider the problem of dynamic orchestration over a NDN-based computing network, in which nodes can be equipped with communication, computation, and data producing resources. Given a set of services with function-chaining structures, we address the design of distributed online algorithm that controls each node to make adaptive decisions on flowing service requests, committing function implementations, and/or producing data. We design a Service Discovery Assisted Dynamic Orchestration (SDADO) algorithm that reduces the end-to-end (E2E) delay of delivering the services, while providing optimal throughput performance. The proposed algorithm hybrids queuing-based flexibility and topology-based discipline, where the topological information is not pre-available but obtained through our proposed service discovery mechanism. We provide throughput-optimality analysis for SDADO, and then provide numerical results that confirm our analysis and demonstrates reduced round-trip E2E delay.

Index Terms—dynamic orchestration, distributed computing networks, Named Data Networking, function chain.

I. INTRODUCTION

Next-generation network service applications with high throughput and low latency requirements, such as online gaming, real-time remote sensing, augmented reality, etc., are explosively dominating the internet traffic. On the other hand, with the significant growth in both aspects of quantity and performance, edge devices provide promising capability of tackling computing tasks. Therefore, elastically dispersing computing workload to edge gradually lead the solutions to deploying the next-generation network services. However, with arbitrary network scale and topology, three questions are naturally raised: i) where to execute the network functions and produce source data; ii) how to steer the data flowing toward appropriate compute and data resources; iii) how to dynamically adapt the decisions to changing service demands. To solve these issues, centralized solutions have been extensively explored [1][2], but have been challenged due to their scalability and cost [3], which in turn makes distributed solutions appealing.

In distributed and dynamic in-network-computing paradigm, it has been found that Named Data Networking (NDN)-based computing orchestration framework offers advantages over traditional edge computing orchestration [4], where NDN obviates having a central registry that tracks all resources. In context of NDN, resources can be reached using name-based queries regardless their locations. When network topology is changing, each device can send a new query with the name remaining constant even if the device’s address changes. With these benefits, NDN overlaying IP has been explored as a promising solution for computing networks [5].

Existing popular NDN routing schemes, e.g. Best Route, Multi-cast, focus on helping nodes to make forwarding decisions, and the target network is only for data delivery [6]. Although modifications can be done to extend these algorithms to computing networks, they still lack enough adaptability to time-varying traffics. On the other hand, dynamic network control policies have been studied in [7], where the proposed backpressure-based algorithm DCNC has shown promising adaptability to dynamic environment but lacks routing discipline, which results in compromised delay performance. An enhanced algorithm EDCNC, which disciplines the backpressure routing using topological information, is further proposed in Ref. [7]. However, EDCNC’s performance heavily relies on the choice of a global control parameter, whose value has no clear clue to choose given an arbitrary network. The gap between NDN and backpressure-based dynamic orchestration has been firstly addressed in [8] on orchestrating single-function network services, where the proposed scheme DECO allows each node to make adaptive decisions on whether or not to accept NDN computing requests. Nevertheless, routing is assumed to be predetermined in [8], which may not be the case in many networking scenarios.

In this paper, we propose the Service Discovery Assisted Dynamic Orchestration (SDADO) algorithm that better addresses orchestration’s adaptability and discipline than the above state-of-art. We seek to deploy function-chaining services in NDN-based distributed computing networks, where the contributions are summarized as follows: i) we develop a framework of sequentially committing NDN-based chaining computing/data requests; ii) we propose a name-based service discovery mechanism to search for topological information processed with targeted function-chaining structures; iii) we propose a novel SDADO orchestration algorithm that achieves reduced E2E latency and throughput-optimality with theoretical analysis and numerical evaluations.

The paper is as organized follows. Section II presents the system model. Section III describes the service discovery
mechanism. The SDADO algorithm’s description, analysis, and numerical evaluation are respectively presented in Section IV, V, and VI. The paper is concluded in Section VII.

II. SYSTEM MODEL

A. Network and Service Model

We consider a time-slotted network system with slots normalized to integer units $t \in \{0, 1, 2 \cdots \}$. A computing network $G = (V, E)$ with arbitrary topology consists of $|V|$ nodes interconnected by $|E|$ communication links. Each node $i$ in the network represents a network unit, e.g., user equipment, access point, edge server, or data center, etc., and let $(i, j)$ represent the link from node $i$ to $j$. Denote $O(i)$ as the set of neighbor nodes having links incident to node $i$. Let $C_{ij}(t)$ represent the maximum transmission rate over link $(i, j)$ at time $t$, whose value evolves ergodically, and its statistical average $\bar{C}_{ij}$ can be estimated by averaging $C_{ij}(t)$ over time.

A network service $\phi$ is described by a chain of $K_\phi$ functions plus the data producing. We use the pair $(\phi, k)$ with $k \in \{0, 1, \cdots, K_\phi \}$ to denote the $k$-th function of service $\phi$, where we index the data producing of service $\phi$ by $(\phi, 0)$ and call it function $(\phi, 0)$. We denote by $C_{ij}(\phi, k)$ the processing complexity of implementing function $(\phi, k)$, where $1 \leq k \leq K_\phi$. That is, when a data packet is processed by implementing function $(\phi, k)$, it consumes $C_{ij}(\phi, k)$ units of computing cycles. Denote $\mathcal{C}_\phi$ as the set of service $\phi$’s consumers; denote $\mathcal{P}(\phi, k)$ as the set of nodes that are equipped with compute or data resources to execute function $(\phi, k)$, $0 \leq k \leq K_\phi$; denote by $C^a_i$ the processing capacity (with unit of, e.g., cycles/slot) at node $i \in \mathcal{P}(\phi, k)$, $1 \leq k \leq K_\phi$, and by $C^a_{ij}$ as the data producing capacity at node $i \in \mathcal{P}(\phi, 0)$.

In context of NDN, the deployment of a streaming service $\phi$ is a process of delivering interest packet streams carrying requests from a consumer to one or multiple data producers, followed by a process of delivering back the data packet streams. Each data packet is flowing through the reverse path traveled by its corresponding interest packet. In order to guarantee that data is processed in the function-chaining order, interest packets carrying the requests flow through resource-equipped nodes that commit implementing the service’s functions in the reversed chaining order, i.e., $(\phi, K_\phi) \rightarrow \cdots \rightarrow (\phi, 0)$.

Streams of interest/data packets delivered for a service can be modeled into chaining stages as shown in Fig. 1. We define each stage of interest packets serving consumer $c \in \mathcal{C}_\phi$ requesting for a function $(\phi, k)$ by an interest commodity and use $(\phi, k, c)$ to index it, and with the same index, define each stage of data packets output from function $(\phi, k)$ serving consumer $c$ by a data commodity. In NDN paradigm, the commodity index of an interest/data packet can be inferred from its name, and a processing commitment can update an interest packet’s commodity by updating its name. We let $z(\phi, k)$, $0 \leq k \leq K_\phi$, represent the size of a data commodity $(\phi, k, c)$ packet for all $c \in \mathcal{C}_\phi$, and assume that the network traffic is dominated by the data packets because the size of an interest packet is negligible relative to $z(\phi, k)$.

B. Queuing Model

We define by $a_i^c(\phi, k)(t)$ the number of requests for service $\phi$ generated by consumer $c$ at time $t$ and by $\lambda_i^c(\phi, k)$ its expected value. We assume that $a_i^c(\phi, k)$ is independently and identically distributed (i.i.d) across timeslots and is upper bounded by $A_{max}$. At each time $t$, every node buffers the received interest and data packets into queues respectively according to their commodities. Each interest/data queue builds up from the transmission of interest/data packets from neighbors, request/data generations, and/or local processing-commitment/processing. Each node makes local orchestration decisions by controlling the queuing of interest packets, while the data packets are forwarded backward and get processed as committed according to the Pending Interest Table (PIT), and their queues are simply updated in First-In-First-Out mode. Fig. 2 shows an example that a computing node equipped with function $(\phi, k)$ makes decisions on whether or not to commit processing for the arrived interest commodity $(\phi, k, c)$ packets. Once a commitment is made, an interest packet is transported from commodity $(\phi, k, c)$ queue to $(\phi, k - 1, c)$ queue with its commodity index updated.

We define $Q_i(\phi, k, c)(t)$ as the number of interest commodity $(\phi, k, c)$ packets queued in node $i$ at the beginning of timeslot $t$; define $s_{ij}^{(\phi, k, c)}(t)$ as the assigned number of interest commodity $(\phi, k, c)$ packets to forward over link $(i, j)$ in timeslot $t$; define $p_i(\phi, k, c)(t)$ as the assigned number of interest commodity $(\phi, k, c)$ packets to commit in timeslot $t$ for function $(\phi, k)$. Then node $i$ has the following queuing dynamics:

$$
Q_i(\phi, k, c)(t+1) \leq Q_i(\phi, k, c)(t) - \sum_{j \in O(i)} b_{ij}^{(\phi, k, c)}(t) - p_i(\phi, k, c)(t) + \sum_{j \in O(i)} b_{ij}^{(\phi, k, c)}(t) + p_i(\phi, k, c)(t) + a_i^c(\phi, k)(t)\mathbb{1}(k = K_\phi),
$$

where $\mathbb{1}(\cdot)$ is the indicator function of $\cdot$; $p_i(\phi, k, c)(t) = 0$ if $k > K_\phi$ or $i \notin \mathcal{P}(\phi, k)$; $\mathbb{1}(\cdot)$ is the indicator function of $\cdot$; $p_i(\phi, k, c)(t) = 0$ if $k > K_\phi$ or $i \notin \mathcal{P}(\phi, k)$.
C. System Constraints

The orchestration dynamically controls the action vector $[(b_{ij}^{(\phi,k,c)}(t)), (p_{ij}^{(\phi,k,c)}(t))]$ subject to the constraints in (2):

$$
\lim_{t \to \infty} \frac{Q_{ij}^{(\phi,k,c)}(t)}{t} = 0 \text{ with prob. } 1, \forall i, \phi, k, c, \quad (2a)
$$

$$
\sum_{(\phi,k,c)} z(\phi,k) b_{ij}^{(\phi,k,c)}(t) \leq C_{ji}, \forall (i,j) \in \mathcal{E}, t, \quad (2b)
$$

$$
\sum_{(\phi,k,c), i} p_{ij}^{(\phi,k,c)}(t) \leq C^p_i, \forall i \in P(\phi), t, \quad (2c)
$$

$$
\sum_{(\phi,k,c), i} (p_{ij}^{(\phi,k,c)}(t)) \leq C^d_p, \forall i \in P(0), t, \quad (2d)
$$

$$
\frac{b_{ij}^{(\phi,k,c)}(t)}{z(\phi,k)} \in \mathbb{Z}^+, \forall (\phi,k,c,i,j) \in \mathcal{E}, t. \quad (2e)
$$

where (2a) characterizes the network rate stability [9]; (2b)-(2d) present the capacity constraints for interest forwarding, processing commitment, and data producing. Note that each interest packet flowing over a link eventually results in a data packet flowing backward over the reverse link, we have (2b) showing that interest packet flow over link $(i,j)$ reflects the backward data packet flow constrained by the expected capacity of link $(j,i)$.

III. Service Discovery

Since knowledge of compute/data resource distribution across the network is helpful for making efficient orchestration decisions, in this paper, we propose that a node can initiate function-chaining service discovery to fetch this knowledge in the form of the minimum abstract distances that interest/data packets will travel to reach all the required resources. This discovery procedure is composed by i) a phase of generating and propagating discovery interest packets sequentially searching the targeted functions in the reverse chaining order and ii) a phase of sending back discovery data packets gathering and updating the minimum abstract distance values.

A. Searching via Discovery Interest Packets

For the targeted function chain segment $(\phi, 0), \cdots, (\phi, k), 0 \leq k \leq K_\phi$, the discovery interest packets search the resources in order $(\phi, k) \rightarrow \cdots \rightarrow (\phi, 0)$, where function $(\phi, k)$ is the immediate searching target. Each discovery interest packet is forwarded by multi-cast, i.e., all the faces of a Forwarding Information Base (FIB) entry are chosen as the forwarding faces except the incoming face. In context of NDN, the name of a discovery interest packet $p$ incorporates the chaining functions’ names, e.g., “$(\phi, k) \leftarrow \cdots \leftarrow (\phi, 0)$”.

Once a node having function $(\phi, k)$ is reached by packet $p$, a new discovery interest packet $p'$ with name “$(\phi, k-1) \leftarrow \cdots \leftarrow (\phi, 0)$” is generated and multi-cast out to search for function $(\phi, k-1)$. In the meanwhile, packet $p$ is still forwarded out searching for other nodes having function $(\phi, k)$ if its hop limit is not reached, with the purpose of letting the current node know where to offload function $(\phi, k)$’s tasks if needed in the service deployment.

Fig. 3(a)-3(c) present the searching phase of a service discovery example. Node 1 in Fig. 3(a) initiates the discovery by generating a discovery interest packet $I_2$ with name “$f_2 \leftarrow f_1 \leftarrow f_0$”, and $I_2$ flows to node 2 having function $f_2$.

In Fig. 3(b), node 2 generates a new discovery interest packet $I_1$ with name “$f_1 \leftarrow f_0$”, and copies of $I_1$ propagate and reach nodes 4 and 7 having function $f_1$, and in the meanwhile. $I_2$ copies are also forwarded. Fig. 3(c) shows that both node 4 and 7 generate the packets $I_0$ with name “$f_0$”, and copies of $I_0$ propagate and reach the data producer (node 5). In addition of $I_2$’s forwarding, node 4 and 7 forward $I_1$ copies that reach each other for their mutual discovery.

With the purpose of detecting multi-paths, the loop detection used in service discovery is different from the existing mechanism used in the standard NDN based on detecting nonce number collision (see Ref. [6]). In service discovery, the nonce number of each discovery interest packet is updated at each hop to avoid nonce number collision among different packet copies. A loop detection in service discovery enables the discovery interest packet to carry a stop list that records the IDs of sequentially traveled nodes. If a node having received a discovery interest packet detects that its own node-ID belongs to the packet’s stop list, a loop is identified. This mechanism identifies a loop when a single discovery interest packet revisits a node but tolerates the case that different copies of the same discovery interest packet arrive at the same node via different paths, which were also identified as a loop according to the standard NDN. For example, if a discovery interest packet sequentially passes by nodes $a \rightarrow b \rightarrow \cdots \rightarrow v \rightarrow a$, then a loop is identified at node $a$; if two copies of the same discovery interest packet respectively pass by nodes $a \rightarrow b_1 \rightarrow \cdots \rightarrow b_n \rightarrow v$ and $a \rightarrow c_1 \rightarrow \cdots \rightarrow c_m \rightarrow v$, this case is identified as a multi-path rather than a loop.

B. Feedback via Discovery Data Packets

A discovery data packet with name “$(\phi, k) \leftarrow \cdots \leftarrow (\phi, 0)$” is generated in either of the two cases: i) for $k \geq 1$, a node having function $(\phi, k)$ has received/gathered one or multiple discovery data packets with name “$(\phi, k-1) \leftarrow \cdots \leftarrow (\phi, 0)$”; ii) for $k = 0$, a node having function $(\phi, 0)$, i.e., data producer, has received a discovery interest packet with name “$(\phi, 0)$”.

Fig. 3: A service discovery example with targeted function chain segment $f_0, f_1, f_2$, where $I_0, I_1, I_2$ are discovery interest packets, and $D_0, D_1, D_2$ are discovery data packets.
Each discovery data packet flows backward, i.e., if a discovery data packet with name \( (\phi, k) \) flows over link \((j, i)\), a discovery interest packet with the same name should have flowed over link \((i, j)\).

Fig. 3(d)-3(f) present the feedback phase of a service discovery example. The data producer (node 5) in Fig. 3(d) generates copies of discovery data packet \( D_0 \) with name \( "f_0" \) that flow backward until reaching node 4 and 7. In Fig. 3(e), node 4 and 7 generate the discovery data packets \( D_1 \) with name \( "f_2 \leftarrow f_1 \leftarrow f_0" \), and \( D_1 \) copies not only flow to node 2 but also flow between node 4 and 7 as their mutual discovery feedback. Fig. 3(f) shows that node 2 generates discovery data packet \( D_2 \) with name \( "f_3 \leftarrow f_2 \leftarrow f_1 \leftarrow f_0" \) that flows back to the discovery initiator (node 1). We further model implementing a function (or data producing) at a node by going through a "processing link" (or "data producing link"). As is shown in Fig. 4, adding the processing links and data producing links onto a feedback routing path forms an augmented path, which characterizes the trajectory traveled by a relaying series of data packets going through a function chain segment. These discovery data packets gather, update and deliver the minimum abstract distance values along the augmented path, and the path’s abstract distance is calculated by summing the abstract lengths of all the links on it. We measure an abstract length as follows:

- for a communication link \((i, j)\), its abstract length \( l_{ij} \) can be defined as \( l_{ij} = 1/c_{ij} \);
- for a processing link at computing node \( i \), its abstract length \( l_{ip} \) is defined as \( l_{ip} = r_{i}(\phi, k)/z(\phi, k)c_{ip} \);
- for a data-producing link at data producer \( i \), its abstract length \( l_{ip} \) is \( l_{ip} = 1/c_{ip} \).

We denote by \( L_{ij}(\phi, k) \) the minimum abstract distance to travel to node \( i \) going through function chain segment \((\phi, 0), \ldots, (\phi, k)\), whose value is updated according to Algorithm 1 at each hop of discovery data packets’ flowing.

**Algorithm 1** Update of the minimum abstract distances \( L_{ij}(\phi, k) \):

1. Initialize \( L_{i}(\phi, k) = L_{j}(\phi, k) = +\infty \) for all \( 0 \leq k \leq K_{\phi} \).
2. for \( k = 0 \) to \( K_{\phi} \) do
3. if \( k = 0 \) and node \( i \in P_{(\phi, 0)} \) then
4. \( L_{i}(0) \leftarrow 1 \).
5. else if \( k > 0 \) and node \( i \in P_{(\phi, k)} \) then
6. \( L_{i}(\phi, k) \leftarrow \min\{l_{ij} + L_{j}(\phi, k) \mid j \in \mathcal{O}(i)\} \).
7. else
8. \( L_{i}(\phi, k) \leftarrow \min\{l_{ij} + L_{j}(\phi, k) \} \).
9. end if
10. end for

In context of NDN, we propose the SDADO framework that enables each node to make allocation decisions for the arrived and queued interest packets based on the observed 1-hop-range interest backlogs and the discovered minimum abstract distances. Given Assumption 1 and 2, Algorithm 2-6 present SDADO at each node \( i \) in timeslot \( t \).

**Assumption 1.** If \( L_{i}(\phi, k) \neq L_{j}(\phi, k') \), there exists a constant \( \delta_L \) such that \( |L_{i}(\phi, k) - L_{j}(\phi, k')| \geq \delta_L \) for all \((\phi, k), (\phi', k')\), \( i, j \).

**Assumption 2.** \( C_{ji}, C_{ip}, C_{jp} \), \( z(\phi, k), r(\phi, k) \) satisfy at least one of the two conditions: i) \( C_{ji} \) and \( C_{ip} \) are integer multiples of \( z(\phi, k) \), and \( C_{jp} \) is integer multiples of \( r(\phi, k) \), for all \((i, j), (\phi, k)\); ii) \( z(\phi, k) = z \) and \( r(\phi, k) = r \) for all \( \phi, k \).

Assumption 1 is easily satisfied given quantization of abstract distance values, where \( \delta_L \) can be the quantization granularity. Assumption 2 is used to prove the throughput optimality of Algorithm 2-6, where max-weight-matching is executed to allocate resources in each timeslot. In general, however, SDADO does not lose throughput-optimality even without Assumption 2 but requires knapsack operations to allocate resources (see Section V for details).

**Algorithm 2** SDADO-data-producing at node \( i \in \bigcup_{\phi, c_{i} > 0} P_{(\phi, 0, c)} \) in timeslot \( t \):

Require: \( U_{i}(\phi, 0, c)(t), \forall (\phi, c) \).  
Ensure: \( p_{i}(\phi, 0, c) \) \( t \), \( \forall (\phi, c) \).

1. for all \((\phi, c)\) such that \( i \in P_{(\phi, 0)} \) do
2. Initialize \( p_{i}(\phi, 0, c)(t) = 0 \).
3. end for
4. Choose \( p_{i}(\phi, k, c) \) \( t \) \( \arg \max_{(\phi, c)} \{ U_{i}(\phi, 0, c)(t) \} \).
5. if \( U_{i}(\phi, k, c)(t) > 0 \) then
6. \( p_{i}(\phi, 0, c)(t) \leftarrow \frac{C_{ip}}{\sum_{k, c} p_{i}(\phi, 0, c)} \).
7. end if

**Algorithm 3** SDADO-processing-commitment at node \( i \in \bigcup_{\phi, c_{i} > 0} P_{(\phi, k, c)} \) in timeslot \( t \):

Require: \( U_{i}(\phi, k, c)(t), \forall (\phi, k, c) \) \( > 0 \).
Ensure: \( p_{i}(\phi, k, c)(t), \forall (\phi, k, c) \).

1. for all \((\phi, k, c)\) such that \( i \in P_{(\phi, k)} \) do
2. \( W_{i, pr}(\phi, k, c)(t) \leftarrow U_{i}(\phi, k, c)(t) - \sum_{(\phi, k, c-1)(t)} z(\phi, k, c-1) \).
3. Initialize \( p_{i}(\phi, k, c)(t) = 0 \).
4. end for
5. Choose \( p_{i}(\phi, k, c) \) \( t \) \( \arg \max_{(\phi, k, c)} \{ W_{i, pr}(\phi, k, c)(t) \} \).
6. if \( W_{i, pr}(\phi, 0, c)(t) > 0 \) then
7. \( p_{i}(\phi, 0, c)(t) \leftarrow \frac{C_{ip}}{\sum_{k, c} p_{i}(\phi, 0, c)} \).
8. end if

Defining the virtual backlog \( U_{i}(\phi, k, c)(t) = z(\phi, k)Q_{i}(\phi, k, c)(t) \) as interest commodity \((\phi, k, c)\) backlog weighted by data.
As is shown in Algorithm 4, node $i$ pre-processes $L_i$ to prepare for making the forwarding decisions by calculating the abstract distance value $L_{i,ij}$ via each outgoing link and the differential abstract distance value $\Delta L_{i,ij}^m$ (line 3), and then sorting the $L_{i,ij}$ among all the outgoing links for each commodity $m$ (line 5) and sorting the $\Delta L_{i,ij}^m$ among all the commodities over each outgoing link $(i,j)$ (line 8). For each commodity $m$, the sorted links’ order of the $L_{i,ij}$, denoted by $J_{i,m}$, are used to prioritize the scheduling order of the outgoing links in Algorithm 5; for each outgoing link $(i,j)$, the sorted commodities’ order of the $\Delta L_{i,ij}^m$ denoted by $M_{i,j}$, are used by Algorithm 6 in determining whether a commodity will be assigned transmission rate over link $(i,j)$. This pre-processing only needs to run once before the online implementation of SDADO in stationary networks or repeat infrequently in networks with time-varying topology.

As is shown in Algorithm 4, node $i$ pre-processes $L_i$ to prepare for making the forwarding decisions by calculating the abstract distance value $L_{i,ij}^m$ via each outgoing link and the differential abstract distance value $\Delta L_{i,ij}^m$ (line 3), and then sorting the $L_{i,ij}$ among all the outgoing links for each commodity $m$ (line 5) and sorting the $\Delta L_{i,ij}^m$ among all the commodities over each outgoing link $(i,j)$ (line 8). For each commodity $m$, the sorted links’ order of the $L_{i,ij}$, denoted by $J_{i,m}$, are used to prioritize the scheduling order of the outgoing links in Algorithm 5; for each outgoing link $(i,j)$, the sorted commodities’ order of the $\Delta L_{i,ij}^m$ denoted by $M_{i,j}$, are used by Algorithm 6 in determining whether a commodity will be assigned transmission rate over link $(i,j)$. This pre-processing only needs to run once before the online implementation of SDADO in stationary networks or repeat infrequently in networks with time-varying topology.

As is shown in Algorithm 4, node $i$ pre-processes $L_i$ to prepare for making the forwarding decisions by calculating the abstract distance value $L_{i,ij}^m$ via each outgoing link and the differential abstract distance value $\Delta L_{i,ij}^m$ (line 3), and then sorting the $L_{i,ij}$ among all the outgoing links for each commodity $m$ (line 5) and sorting the $\Delta L_{i,ij}^m$ among all the commodities over each outgoing link $(i,j)$ (line 8). For each commodity $m$, the sorted links’ order of the $L_{i,ij}$, denoted by $J_{i,m}$, are used to prioritize the scheduling order of the outgoing links in Algorithm 5; for each outgoing link $(i,j)$, the sorted commodities’ order of the $\Delta L_{i,ij}^m$ denoted by $M_{i,j}$, are used by Algorithm 6 in determining whether a commodity will be assigned transmission rate over link $(i,j)$. This pre-processing only needs to run once before the online implementation of SDADO in stationary networks or repeat infrequently in networks with time-varying topology.
Algorithm 5 describes the SDADO online forwarding of interest packets at node $i$ in timeslot $t$. Node $i$ prioritizes the commodities by sorting metrics $\Theta^m_i(t)$ (line 8) in non-increasing order, whose formulation (line 6) intuitively implies giving higher forwarding priority to interest commodity with higher delay-reduction gain $\Delta L^m_{ij}(t)$ weighted by congestion-reduction gain $\Delta U^m_{ij}(t)$. Node $i$ then schedules commodities for forwarding according to this priority order, where $\text{AsgmtInd}_{ij}(t)$ indicates whether link $(i,j)$ has been scheduled (line 10-19), and the kernel operation is to implement “FwdRateAsgmt” shown in Algorithm 6.

Algorithm 6 describes “FwdRateAsgmt” (called in line 16 of Algorithm 5) judging whether to allocate the average reverse link capacity of a outgoing link $(i,j)$ to a targeted commodity $\bar{m}'$. Line 1 shows four conditions that commodity $\bar{m}'$ has to satisfy in order to be scheduled. The operations in line 2-14 are used to further determine the eligibility of allocating forwarding resource to interest commodity $\bar{m}'$ over link $(i,j)$, which involves comparing two bounding values $B^h_{ij}, \bar{m}'$ and $B^p_{ij}, \bar{m}'$ calculated based on the metrics $\omega^m_{ij}(t)$ (line 3 and 5-10). As will be shown in the proof of Theorem 1 (see [10]) in Section V, scheduling the commodity that satisfy the four conditions and the condition $B^h_{ij}, \bar{m}' \geq B^p_{ij}, \bar{m}'$ in each timeslot guarantees the network stability.

The complexity of the pre-processing shown by Algorithm 4 at node $i$ is dominated by sorting $\{\Delta L^m_{ij} : \forall j \in \mathcal{O}(i)\}$ for each commodity $m$ and sorting $\{\Delta L^m_{ij} : \forall m\}$ for each outgoing link $(i,j)$ and therefore is $O(|\mathcal{O}(i)|N \log(|\mathcal{O}(i)|N))$. In each timeslot, i) both Algorithm 2 and 3 implemented by node $i$ have linear complexity with respect to the number of commodities, i.e., $O(N)$; ii) dominated by calculating $B^h_{ij}, \bar{m}'(t)$ and $B^p_{ij}, \bar{m}'(t)$ for each link $(i,j)$ and each commodity $\bar{m}'$, the overall complexity of implementing Algorithm 5 composed with Algorithm 6 at node $i$ is $O(N^2|\mathcal{O}(i)|)$.

V. ALGORITHM ANALYSIS

In this section, we analyze throughput-optimality of SDADO and then further describe the SDADO algorithm without Assumption 2. We start with the following lemma.

**Lemma 1.** With Assumption 1 and 2, there exists a coefficient $\eta_{ij}(t)$ satisfying $0 \leq \eta_{ij}(t) \leq \frac{h_i/s_{ij}}{\gamma(y_{ij})}$ such that the optimal solution $b^*_{ij}(t)$ of the problem

$$\max \sum_m [\Delta U^m_{ij}(t) + \eta_{ij}(t) \Delta L^m_{ij}] z^m b^m_{ij}(t),$$

s.t. $\sum_m z^m b^m_{ij}(t) \leq \bar{C}_{ij}, \ b^m_{ij}(t) \in \mathbb{Z}^+$,

and the SDADO-interest-forwarding solution $b_{ij}(t)$ obtained in Algorithm 4-6 satisfy $Z_{\eta_{ij}(t)}(b_{ij}(t)) \geq Z_{\eta_{ij}(t)}(b^*_{ij}(t)) - [\bar{h}_i/s_{ij} - 1]n_0\bar{C}_{ij}$ where $Z_{\eta_{ij}(t)}(\ast)$ represents the objective function (3a) of the $\bar{b}^m_{ij}(t)$ given parameter $\eta_{ij}(t)$.

**Proof.** The detailed proof can be found in [10].

Let $\chi$ represent a vector with the same length as $U(t)$ having the $\lambda^0\hat{z}(\epsilon; K_\alpha)$ as the $(c, (\phi, K_\alpha, c))$th elements and 0 elsewhere: let $\Lambda$ represent the computing network capacity region [7] defined as the closure of all $\chi$ that can be stabilized by the computing network. Then based on Lemma 1, we can further show that SDADO is throughput optimal:

**Theorem 1.** With Assumption 1 and 2, for any $\chi$ in the interior of $\Lambda$, i.e., $\exists \chi > 0$ such that $(\chi + \epsilon \chi) \in \Lambda$, SDADO in Algorithm 2-6 stabilizes the network, i.e.,

$$\lim_{t \to \infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \sum_{m,i} U_i^m(\tau) \leq B_0 + B_1 \epsilon,$$

with prob. 1, where $B_0$ and $B_1$ are constants depending on the network parameters $\bar{C}_{ij}, c_i, C_i, z_i, \epsilon_i, \bar{r}_{ij}, \epsilon_{ij}$.

**Proof.** The detailed proof can be found in [10].

The satisfaction of (4) indicates strong stability of the network [9]. In fact, Assumption 2 can be removed without affecting the throughput-optimality of SDADO, but we need to make modifications on Algorithm 2, 3, and 6: i) the max-weight-matching in line 4-7 of Algorithm 2 and line 5-8 of Algorithm 3 can be replaced by knapsack; ii) instead of implementing line 12 in Algorithm 6, we calculate $\eta_{ij}(t) = \min\{B^h_{ij}, \bar{m}' + 1, \frac{1}{2}[B^p_{ij}, \bar{m}' + 1 + B^h_{ij}, \bar{m}']\}$ and plug it into (3a), and then solve (3) by knapsack. Thus, we further have the following corollary.

**Corollary 1.** With Assumption 1, for any $\chi$ in the interior of $\Lambda$, the SDADO algorithm with knapsacks stabilizes the network.

VI. NUMERICAL RESULTS

In this section, we evaluate performance of the SDADO scheme and compare it with state-of-the-art: the DCNC and EDCNC schemes [7] and the Best Route scheme [6] extended to NDN-based computing networks. Fig. 5(a) shows the simulated network with Fog topology. The network and service settings are listed in Table I.

**TABLE I: Network and Service Settings**

| Wireless Link Index | 15-25 | 25-29 |
|---------------------|-------|-------|
| Small Scale Fading  | Rician, Rician-factor= 10dB. | Rayleigh |
| Min. Avg. Rx-SNR    | 5dB given Edge-Tx. | 0dB given Mesh-Tx. |
| Other Settings      | • 10MHz bandwidth; | • Free space pathloss; | • Log-normal shadowing. |

| Service Functions   | (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2). | |
|---------------------|-------------------------------------------------|---|
| $P_{\phi,0}$        | 10, 12, 14. | 15, 17, 19. |
| $\rho(z, k)$ (KB)   | 10-19 | 10-13 |
| $r(z, k)$ (cycles/packet) | $z(1,0) = 5 \times 10^4$; | $z(2, 1) = 5 \times 10^3$; |
|                    | $z(1,2) = 5 \times 10^3$; | $z(2, 2) = 20$. |
|                    | $r(1,1) = 2 \times 10^4$; | $r(2, 1) = 5 \times 10^4$; |
|                    | $r(1,2) = 2 \times 10^4$. | $r(2, 2) = 10^4$. |

Fig. 5(b) shows the average round trip E2E delay performances of DCNC, Best Route, EDCNC, and SDADO evolving with respect to average request generation rate per consumer per service denoted by $\lambda^0$. In low traffic scenario ($\lambda^0 \leq 7.0$ count/timeslot), Best Route, EDCNC, and SDADO achieve much lower E2E delay than DCNC. This is because the former three algorithms enable topological information
to dominate routing and guide the interest packets flowing along short paths, while DCNC has to explore paths by sending interest packets in different directions, which sacrifices delay performance. As $\lambda^c_0$ exceeds 7.0 count/timeslot, the E2E delay of Best Route sharply increases to unbearable magnitude indicating that its throughput limit is exceeded. In contrast, DCNC, EDCNC, SDADO still support high traffics due to their flexibility, while SDADO and EDCNC outperform DCNC by achieving lower E2E delay ($3.6 \times 35.5 \times$ gain). Although Fig. 5(b) shows that SDADO and EDCNC achieve the similar E2E delay, SDADO is much easier to implement in the distributed manner. To implement EDCNC, one has to heuristically find a global bias coefficient $\eta = 10^8$ in the simulation and distribute it among all the nodes. However, given an arbitrary network, there is no clear clue of efficiently finding a proper value of $\eta$, even it significantly influences the E2E delay. In contrast, SDADO does not use any fixed global bias coefficient but adaptively hybrids queuing-based flexibility and topology-based discipline in orchestration.

Throughput performances are further demonstrated in Fig. 5(c) by showing the time average interest backlog accumulations evolving with traffics. When $\lambda^c = 7.0$ count/timeslot, the average total interest backlog under Best Route exhibits a sharp increase, illustrating its throughput limit. In comparison, the sharp backlog increase under DCNC, EDCNC, and SDADO occurs at $\lambda^c_0 = 17.0$ count/timeslot exhibiting $2.4 \times$ throughput gain. In addition, throughput limits of DCNC, EDCNC, and SDADO are the same because all of them are throughput optimal. In the meanwhile, backlog accumulation levels of SDADO and EDCNC are much lower than DCNC indicating significantly reduced E2E delay. The time average accumulated data backlogs are shown in Fig. 5(d), which verifies the similar trend.

VII. CONCLUSIONS

In this paper, we consider dynamic orchestration problem in NDN-based distributed computing networks to deliver next-generation services having function-chaining structures. We propose the SDADO algorithm that dynamically makes forwarding, computation commitment, and data producing decisions on NDN-interests by adaptively hybridizing queuing-based flexibility and topology-based discipline, where the topological information is obtained via the proposed service discovery mechanism. Under SDADO, the resulting round-trip E2E delay of the service deployment is reduced while the network throughput is maximized. Theoretical analysis and simulation results have confirmed our conclusions.

REFERENCES

[1] H. Topcuoglu, S. Hariri, and M.-Y. Wu, “Performance-effective and low-complexity task scheduling for heterogeneous computing,” IEEE transactions on parallel and distributed systems, vol. 13, no. 3, pp. 260–274, 2002.
[2] L. Yang, J. Cao, S. Tang, T. Li, and A. T. Chan, “A framework for partitioning and execution of data stream applications in mobile cloud computing,” in 2012 IEEE Fifth International Conference on Cloud Computing, 2012, pp. 794–802.
[3] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog computing and its role in the internet of things,” in Proceedings of the first edition of the MCC workshop on Mobile cloud computing, 2012, pp. 13–16.
[4] R. Pirmagomedov, S. Srikanteswara, D. Moltchanov, G. Arbo, Y. Zhang, N. Himayat, and Y. Koucheryavy, “Augmented computing at the edge using named data networking,” in 2020 IEEE Globecom Workshops (GC Wkshps), 2020, pp. 1–6.
[5] S. Luo, S. Zhong, and K. Lei, “IP/NDN: A multi-level translation and migration mechanism,” in NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium, 2018, pp. 1–5.
[6] A. Afanasyev and et. al., “NFD developer’s guide,” NDN. Technical Report NDN-0021. https://named-data.net/publications/techreports/, 2021.
[7] H. Feng, J. Llorca, A. M. Tulino, and A. F. Molisch, “Optimal dynamic cloud network control,” IEEE/ACM Transactions on Networking, vol. 26, no. 5, pp. 2118–2131, 2018.
[8] K. Kamran, E. Yeh, and Q. Ma, “DECO: Joint computation, caching and forwarding in data-centric computing networks,” in Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing, 2019, pp. 111–120.
[9] M. J. Neely, “Stochastic network optimization with application to communication and queuing systems,” Synthesis Lectures on Communications and Networking, vol. 3, no. 1, pp. 1–211, 2010.
[10] H. Feng, Y. Zhang, S. Srikanteswara, M. Spoczynski, M. Gupta, G. Arbo, J. Zhu, and N. Himayat, “Optimal dynamic orchestration in NDN-based computing networks,” in https://arxiv.org/abs/2208.09871.