Joint Network Resource and Service Optimization in Distributed Computing Networks

Jianan Zhang, Peng Li, and Yichen Guo

***Abstract*—To support elastic services with flexible processing configurations in distributed computing networks, we propose a joint network resource and service control policy that optimizes routing, scheduling, processing locations, and the selection of processing configurations. The flexibility of service configurations introduces a new degree of freedom for improving network performance. First, we formulate the maximum elastic service flow problem and develop a polynomial-time algorithm to compute the network's maximum throughput for supporting elastic services that can be processed by any available chain of service functions. Under stochastic service arrivals, we design a dynamic control policy that, upon each service arrival, chooses the service configuration, processing locations, and routing. In addition, we develop a scheduling policy to maximize network throughput. Furthermore, we extend the policy to maximize network utility by assigning a utility value to each service configuration. We prove that this utility maximization policy achieves an expected total network utility that is near-optimal.**

I. Introduction

Modern network applications are increasingly built on elastic services deployed in distributed computing networks. These applications often require intensive computation and have dynamic workloads. Examples include automated machine learning (AutoML) pipelines, semantic communication systems, and cooperative autonomous driving platforms. For example, an AutoML workflow runs successive training and validation tasks and performs neural architecture search processes that are highly resource-intensive. Similarly, semantic communication merges signal processing with on-device artificial intelligence inference, and autonomous driving platforms involve vehicles cooperatively sharing sensor data and learning tasks. The common thread in these scenarios is the need for flexible and efficient service deployment that meets strict performance requirements despite time-varying loads. This need motivates the design of elastic services, which adaptively leverage distributed computation and communication resources to maintain performance targets.

Elastic services are characterized by having multiple possible service configurations, each of which consumes different network resources and offers different utility or performance. For example, neural architecture search in AutoML produces candidate neural networks that differ in depth, width, and connectivity; each candidate has a distinct computational footprint and achieves different accuracy and latency [1]. MnasNet [2] develops a factorized hierarchical search algorithm for the chain of blocks that compose a convolutional neural network that have various accuracy and latency tradeoffs. A semantic communication service may include interchangeable encoding and decoding functions (from lightweight compression to deep neural network codecs) with adjustable bit rates. A concrete example is DeepSC [3], a deep learning enabled semantic communication framework whose compression ratio can be tuned by scaling the latent feature dimension, ranging from lightweight feature vectors to high-fidelity representations. An autonomous driving perception service may allow variable image resolution or frame rate between camera, edge server, and vehicles. A recent example is EdgeDuet [4], which offloads fine-grained image tiles to an edge server while processing lower-resolution frames locally; by adjusting the tile size (spatial resolution) and the sampling rate, the system trades bandwidth for detection accuracy in real-time vision. This elasticity enables a service to scale its computation and communication requirements up or down, or to substitute function variants, in response to resource availability and performance targets.

Network service providers increasingly use network function virtualization to deploy service function chains, which are ordered sequences of virtual network functions that process traffic. The static service function placement problem seeks an optimal deployment of service functions across a network and routing of traffic through these functions to meet demands at minimal cost or resource usage. A joint function placement and routing problem was studied in [5] and the minimum cost was obtained by solving an integer linear program. A bi-criteria near-optimal service function placement algorithm under capacity and path length constraints was studied in [6]. Given fixed function placements, near-optimal routing algorithms were proposed in [7]. Approximation algorithms were developed in [8], [9] to maximize network utility or to minimize network resource costs by optimizing the placement and routing of functions.

Managing service function chains in a distributed computing network is challenging when user demand fluctuates dynamically. In contrast to static placement, dynamic control of service function chains focuses on real-time decisions for routing and function scheduling under time-varying traffic. A mobility-driven service placement was developed in [10] to minimize the average latency using Lyapunov optimization. A backpressure-based algorithm was developed in [11] to jointly choose processing locations and routes for unicast flows to maximize network throughput. For more complex traffic patterns, a throughput-optimal control policy for arbitrary mixed-cast service function chain traffic was developed in [12], based on the universal max-weight algorithm [13]. Subsequent works have applied reinforcement learning to dynamic control of service chains (e.g., for video streaming) to handle even more

complex decision spaces [14], [15].

While most previous works studied network resource optimization problems to support services with fixed resource requirements, an elastic service function chain model was studied in [16], [17]. In that model, a processing-and-routing graph represents service functions as nodes and dependencies as directed edges; a service request is complete if it is served along any source-to-destination path in this graph. Approximation algorithms were developed to maximize throughput within an $O(\log n)$ factor of optimal [16] and to maximize utility close to the optimal [17], under the assumption that each link's capacity exceeds the demand of any single service by a logarithmic factor. Both algorithms considered integer (unsplittable) requests and ignored stochastic arrivals. In this paper, we first allow fractional flows to characterize the network capacity region, and then design dynamic control policies for stochastic service arrivals that achieve the network capacity.

The central challenge addressed in this paper is the joint optimization of network resources and service configuration for elastic services under both communication and computation constraints. The goal is to maximize service throughput or utility while respecting node computation capacities and link bandwidth constraints. Furthermore, the problem is often dynamic, calling for adaptive online solutions. In summary, joint network resource and elastic service optimization requires careful coordination of which functions to execute, where to execute them, and how to transfer data between them, in order to meet application-specific performance requirements efficiently. The main contributions of this paper are summarized below.

- We develop a modeling framework for joint network and service optimization and formally define the maximum elastic service flow problem. We compute the maximum service flow rate that can be processed by any chain of service functions.
- We develop a throughput-optimal dynamic control algorithm that optimizes service configuration, processing locations, and routing upon the arrival of each service, and a scheduling policy that stabilizes the network under stochastic arrivals.
- We extend the dynamic control algorithm to maximize expected network utility in scenarios where each service configuration yields a different concave utility function. We prove that this utility-aware policy attains near-optimal total network utility.

The remainder of this paper is organized as follows. Section II introduces the models for distributed computing networks and elastic services. Section III studies the maximum elastic service flow problem to characterize the network capacity region. Section IV develops a dynamic algorithm for joint routing, scheduling, function processing, and service configuration to maximize network throughput under stochastic arrivals. Section V extends the algorithm to maximize network utility when different service configurations have different utilities. Section VI presents simulation results, and Section VII concludes the paper.

II. System Model

In this section, we first present a model for distributed computing networks, and then develop a model for elastic services.

A. Distributed Computing Network Model

A distributed computing network is modeled by a directed graph $G(V, E)$, where V denotes the set of nodes and E denotes the set of links. A node is either a forwarding node such as a router, or a computing node that can process services in addition to forwarding packets. The capacity of a computing node v is denoted by μ_v. A directed link $(u, v) \in E$ represents a communication channel from node u to node v with bandwidth capacity μ_{uv}.

B. Elastic Service Models

A service consists of one or more service functions, each of which represents the processing of a subtask. In the simplest case, a service contains a single service function. A service function ϕ is characterized by its computation and communication resource requirements. For each unit of input flow, function ϕ requires r^ϕ units of computation resources, and produces ξ^ϕ units output flow.

A *service function chain* Φ contains a chain of functions that must be processed in order $\phi_1 - \phi_2 - \cdots - \phi_M$. The output of function ϕ_i is the input of function ϕ_{i+1}. To process a unit service flow (i.e., a unit entering ϕ_1), service function ϕ_i requires $x^{\phi_i} = r^{\phi_i} \Pi_{j=1}^{i-1} \xi^{\phi_j}$ computation resources, and outputs $w^{\phi_i} = \Pi_{j=1}^{i} \xi^{\phi_j}$ units flow.

In many applications, there are multiple ways to complete a given task, and the service's processing can be elastic. We use an elastic service function chain to model this flexibility. For example, in AutoML, different network architectures and hyperparameters yield different deep learning models, each capable of performing the task with its own resource demands and accuracy trade-offs.

For an elastic service function, both the computation resource r^ϕ and the flow scaling factor ξ^ϕ can be variables. For example, for the neural architecture search space in [18], a convolutional layer with a stride of 2 on a $32 \times 32 \times 32$ feature map requires approximately 4×10^6 FLOPs for a 3×3 filter, but 13×10^6 FLOPs for a 5×5 filter. Moreover, using 24 output channels produces a $32 \times 32 \times 24$ feature map, whereas 36 channels produce $32 \times 32 \times 36$. Different configurations require different computation resources and output different data volumes. We now formally define elastic services.

Definition 1. *An* elastic service function $\tilde{\phi}$ *is a service function that can be implemented by any one of a set of alternative functions* $\{\phi_1, \phi_2, \ldots, \phi_K\}$.

Definition 2. *An* elastic service function chain $\tilde{\Phi}$ *is a service chain that can be implemented by any one of a set of alternative function chains* $\{\Phi_1, \Phi_2, \ldots, \Phi_K\}$.

An elastic service function chain can have a variable number of service functions and variable computation and communication requirements. A *configuration* of service function chain

is an ordered sequence of deterministic service functions. The *cardinality* of an elastic service function chain $|\tilde{\Phi}|$ is the number of different configurations of the service function chain.

Among the configurations $\{\Phi_1, \Phi_2, \ldots, \Phi_K\}$ of an elastic service function chain $\tilde{\Phi}$, some service functions may appear in more than one configuration. Enumerating all configurations has high complexity and the number of configurations could be exponential in the number of service functions. To manage this complexity, we adopt a graph-based representation for elastic service chains, similar to the approach in [17].

Graph representation of an elastic service function chain. We represent an elastic service function chain $\tilde{\Phi}$ as a directed acyclic graph $G_{\tilde{\Phi}}$, where nodes represent service functions and edges represent function dependencies. In $G_{\tilde{\Phi}}$, there is a dummy source node ϕ_s and a dummy destination node ϕ_t, and any path from ϕ_s to ϕ_t represents a configuration of the elastic service function chain.

Example. Suppose that $\tilde{\Phi}$ consists of a chain of elastic service functions $\tilde{\phi}^1, \tilde{\phi}^2, \ldots, \tilde{\phi}^n$, where there are $\Pi_{i=1}^n |\tilde{\phi}^i|$ configurations. Instead of enumerating all these configurations, a layered graph represents the dependency of the functions, where nodes in layer-i represent the configurations of $\tilde{\phi}^i$ and nodes between adjacent layers are connected as a complete bipartite graph where the edges are directed from layer-i to layer-$(i+1)$ (Fig. 1). Any path from ϕ_s to ϕ_t represents a configuration of the elastic service function chain, and the service is complete if the functions in any path are executed in order.

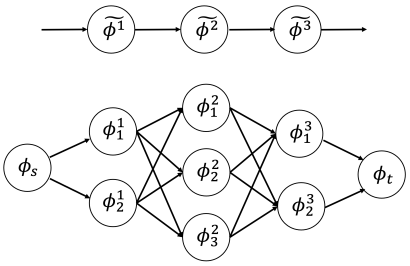


Fig. 1. A chain of elastic service functions $\tilde{\phi}^1 - \tilde{\phi}^2 - \tilde{\phi}^3$ (top), and its graph representation $G_{\tilde{\Phi}}$ (bottom).

In the above example, $G_{\tilde{\Phi}}$ is a layered graph. Note that the algorithms in this paper apply to arbitrary directed acyclic graphs where edges may cross layers and $\phi_s \to \phi_t$ paths may have different lengths.

III. Maximum Elastic Service Flow

In this section, we study the maximum elastic service flow problem, defined as follows.

Maximum elastic service flow problem: Given a distributed computing network $G(V, E)$ with node computation capacity constraints and link communication capacity constraints, and an elastic service function chain $\tilde{\Phi}$, determine the maximum rate of service flow from a source node $s \in V$ to a destination node $t \in V$ that can be processed through $\tilde{\Phi}$. This flow may utilize any available configuration of the elastic service function chain.

We assume that the flow is splittable: the total service flow can be divided among multiple service chain configurations. We jointly optimize flow routing, function computation, and flow splitting across configurations to maximize the total service throughput.

To solve this problem, we construct a *product graph* $\mathcal{G}_{\tilde{\Phi}}$ that captures both routing and processing of an elastic service function chain flow. For each node $\phi_a \in G_{\tilde{\Phi}}$, there is a copy of the computing network G in $\mathcal{G}_{\tilde{\Phi}}$, which is denoted by $G(\phi_a)$. If there is an edge $(\phi_a, \phi_c) \in G_{\tilde{\Phi}}$, then there are directed edges from the computation nodes in $G(\phi_a)$ to their corresponding replicas in $G(\phi_c)$. In $\mathcal{G}_{\tilde{\Phi}}$, flows in $G(\phi_a)$ represent flows that have been processed by ϕ_a, and flows on the edges from $u(\phi_a)$ to $u(\phi_c)$ represent processing ϕ_c at node u on traffic that was output by ϕ_a. One caveat is that ϕ_t is a dummy destination node in $G_{\tilde{\Phi}}$ which does not represent function computation. If $(\phi_c, \phi_t) \in G_{\tilde{\Phi}}$, flows on the cross-layer edge from $G(\phi_c)$ to $G(\phi_t)$ do not represent function processing or consume computation resource. For simplicity, if ϕ_c is processed at a node $v \in G$, then we assume that the flow processed by function ϕ_c enters $G(\phi_t)$ through the edge $(v(\phi_c), v(\phi_t))$ and is transmitted to $t(\phi_t)$ through the edges in $G(\phi_t)$. See Fig. 2 for an illustration of the construction of the product graph.

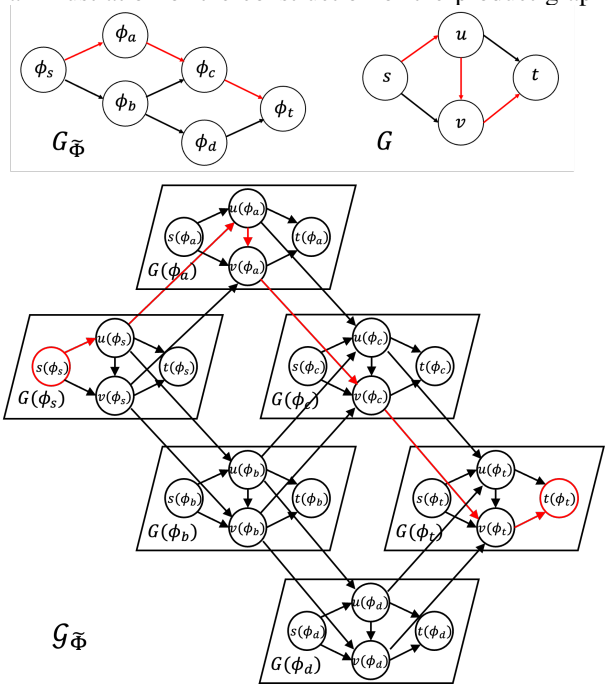


Fig. 2. Construction of the product graph. A flow that traverses edges $(s,u),(u,v),(v,t)$ where ϕ_a is computed at u and ϕ_c is computed at v is represented by the red path in $\mathcal{G}_{\tilde{\Phi}}$.

Theorem 1. *Every path from $s(\phi_s)$ to $t(\phi_t)$ in $\mathcal{G}_{\tilde{\Phi}}$ represents the routing, function computation, and service configuration of a service flow from s to t in G. Moreover, a service flow from s to t in G can be decomposed into a collection of paths from $s(\phi_s)$ to $t(\phi_t)$ in $\mathcal{G}_{\tilde{\Phi}}$ and possibly cycles.*

Proof: Consider a path $P \in \mathcal{G}_{\tilde{\Phi}}$ from $s(\phi_s)$ to $t(\phi_t)$. The path traverses a sequence of copies of G, denoted by $\{G(\phi_s), G(\phi_{P,1}), G(\phi_{P,2}), \ldots, G(\phi_{P,N}), G(\phi_t)\}$.

Then, by the construction of the product graph $\mathcal{G}_{\tilde{\Phi}}$, an edge $(u(\phi_{P,i-1}), u(\phi_{P,i})) \in P$ between $G(\phi_{P,i-1})$ and $G(\phi_{P,i})$ represents the computation of function $\phi_{P,i}$ at node u, and an edge $(u(\phi_{P,i}), v(\phi_{P,i})) \in P$ in $G(\phi_{P,i})$ represents the transmission of flow that has been processed by function $\phi_{P,i}$ on link (u, v). The flow is processed by services functions $\phi_{P,1} - \phi_{P,2} - \cdots - \phi_{P,N}$ in order. A corner case is that the edge from $G(\phi_{P,N})$ to $G(\phi_t)$ does not represent function computation. The edges in $G(\phi_{P,N})$ and $G(\phi_t)$ transmit the flow that has been processed by function $\phi_{P,N}$. Therefore, every path P represents the routing, function computation, and service configuration from s to t.

To prove that a service flow from s to t in G can be decomposed into a collection of paths, we first consider a flow that is processed by a single service configuration. Apply the path decomposition algorithm that iteratively computes a path from $s(\phi_s)$ to $t(\phi_t)$ whose edges carry positive flows, and extract the smallest flow from the path while considering flow scaling and maintaining residual flows non-negative. The union of these paths is the flow. Finally, consider a flow processed by multiple service configurations. The flow is a union of flows, each of which is processed by a fixed service configuration. Applying the path decomposition algorithm iteratively again generates the paths from $s(\phi_s)$ to $t(\phi_t)$. The detailed proof is omitted and is available upon request. ∎

Theorem 1 shows that the routing, computation, and configuration of an elastic service flow in G can be represented by a flow in $\mathcal{G}_{\tilde{\Phi}}$. Therefore, we next develop algorithms for flow optimization based on $\mathcal{G}_{\tilde{\Phi}}$ to study the maximum elastic service flow problem. Moreover, the flow decomposition allows us to view the elastic service flow problem from the maximum multi-commodity service flow perspective, where a commodity is defined using the tuple (s, t, Φ_i) for $\Phi_i \in \tilde{\Phi}$.

Theorem 2. *The maximum elastic service flow is equivalent to the maximum multi-commodity service flow, where each commodity corresponds to a configuration of the service function chain and has the same source and destination nodes.*

Proof: On one hand, if we allocate flows to individual configurations, their sum constitutes a feasible elastic service flow, since by definition the elastic flow can be split among any of the configurations. On the other hand, any feasible elastic flow can be decomposed (by Theorem 1) into flows along specific configurations, which are the commodities. Therefore, the maximum total flow obtainable by distributing traffic among configurations (multi-commodity flow) is exactly the maximum elastic service flow. ∎

Our product graph construction uses one copy of G per node in $G_{\tilde{\Phi}}$. An alternative construction in [17] uses one copy of G per edge in $G_{\tilde{\Phi}}$, which can lead to a larger graph (potentially quadratic in size for dense $G_{\tilde{\Phi}}$).

A. Flow-based formulation

We now formulate a polynomial-size linear program to compute the maximum elastic service flow. We introduce flow variables on the edges of $\mathcal{G}_{\tilde{\Phi}}$ and enforce a generalized flow-conservation law that accounts for flow scaling at processing nodes. Consider a node $u(\phi_a) \in \mathcal{G}_{\tilde{\Phi}}$. Let $\delta^-(\phi_a)$ denote the set of service functions that precede ϕ_a, i.e., $\phi_i \in \delta^-(\phi_a)$ if $(\phi_i, \phi_a) \in G_{\tilde{\Phi}}$. Let $\delta^+(\phi_a)$ denote the set of service functions that follow ϕ_a, i.e., $\phi_i \in \delta^+(\phi_a)$ if $(\phi_a, \phi_i) \in G_{\tilde{\Phi}}$. Let $\delta^-(u)$ denote the set of neighbors with edges into u, i.e., $j \in \delta^-(u)$ if $(j, u) \in G$. Let $\delta^+(u)$ denote the set of neighbors with edges out of u, i.e., $j \in \delta^+(u)$ if $(u, j) \in G$. The generalized flow conservation law at node $u(\phi_a) \in \mathcal{G}_{\tilde{\Phi}}$ is as follows.

$$\sum_{j \in \delta^-(u)} f_{j(\phi_a)u(\phi_a)} + \sum_{\phi_i \in \delta^-(\phi_a)} \frac{\xi^{\phi_a}}{r^{\phi_a}} f_{u(\phi_i)u(\phi_a)} = \sum_{j \in \delta^+(u)} f_{u(\phi_a)j(\phi_a)} + \sum_{\phi_i \in \delta^+(\phi_a)} \frac{1}{r^{\phi_i}} f_{u(\phi_a)u(\phi_i)}. \quad (1)$$

The generalized flow conservation law is similar to what was developed in [12], and has the following differences. In an elastic service-function chain, the service function ϕ_a can be preceded and followed by several other functions. Accordingly, it may receive traffic from multiple upstream function outputs, process the traffic, and then forward it to multiple downstream functions. If ϕ_a receives traffic from $\phi_i \in \delta^-(\phi_a)$, the computation resource for each unit traffic is r^{ϕ_a} and the output scaling is ξ^{ϕ_a}. For computation rate $f_{u(\phi_i)u(\phi_a)}$ on edge $(u(\phi_i), u(\phi_a))$, the output flow rate of function ϕ_a incoming to node u is $\frac{\xi^{\phi_a}}{r^{\phi_a}} f_{u(\phi_i)u(\phi_a)}$. If it forwards traffic to $\phi_i \in \delta^+(\phi_a)$, the computation rate $f_{u(\phi_a)u(\phi_i)}$ on edge $(u(\phi_a), u(\phi_i))$ is r^{ϕ_i} times the outgoing flow rate from node u. One caveat is that if $\phi_t \in \delta^+(\phi_a)$, then we define $r^{\phi_t} = \xi^{\phi_t} = 1$ for flow conservation, although $f_{u(\phi_a)u(\phi_t)}$ does not consume computation resources.

We can now state the flow-based linear program for the maximum elastic service flow problem. The objective is to maximize the arrival rate of the elastic service flow that can be processed. Eq. (4) guarantees that the arrival rate to s equals the sum of the outgoing flow rates from $s(\phi_s)$ to nearby nodes and the processing at node s through any function $\phi_i \in \delta^+(\phi_s)$. Eq. (2) imposes communication capacity constraints on the links, and Eq. (3) imposes computation capacity constraints on the nodes.

$$\max_{\boldsymbol{f} \geq \mathbf{0}} \quad \lambda$$

$$\text{s.t.} \quad \sum_{\phi_i \in G_{\tilde{\Phi}}} f_{u(\phi_i)v(\phi_i)} \leq \mu_{uv}, \quad \forall (u, v) \in E, \quad (2)$$

$$\sum_{(\phi_i, \phi_j) \in G_{\tilde{\Phi}}, \phi_j \neq \phi_t} f_{(u(\phi_i)u(\phi_j))} \leq \mu_u, \ \forall u \in V, \quad (3)$$

$$\lambda = \sum_{j \in \delta^+(s)} f_{s(\phi_s)j(\phi_s)} + \sum_{\phi_i \in \delta^+(\phi_s)} \frac{1}{r^{\phi_i}} f_{s(\phi_s)s(\phi_i)}, \quad (4)$$

$$\text{flow conservation Eq. (1)}, \forall u \in \mathcal{G}_{\tilde{\Phi}} \setminus \{s(\phi_s), t(\phi_t)\}.$$

The flow-based linear programming formulation has a polynomial number of variables and constraints, and therefore can be solved in polynomial time. However, it does not explicitly enumerate individual service configurations or routing, which makes it somewhat abstract for designing a dynamic policy. We therefore develop an equivalent path-based formulation next, which will directly inform our online algorithm.

B. Path-based formulation

We next develop a path-based formulation. Consider a path P in $\mathcal{G}_{\tilde{\Phi}}$ from $s(\phi_s)$ to $t(\phi_t)$. The path P corresponds to a flow in G that is processed by a configuration of the elastic service function chain. The configuration is represented by a path Φ_P from ϕ_s to ϕ_t in $G_{\tilde{\Phi}}$, due to Theorem 1. Let $\Phi_P = \{\phi_s, \phi_{P,1}, \phi_{P,2}, \dots, \phi_{P,N}, \phi_t\}$. For a unit service flow input to $\phi_{P,1}$, service function $\phi_{P,i}$ requires $x^{\phi_{P,i}} = r^{\phi_{P,i}} \Pi_{j=1}^{i-1} \xi^{\phi_{P,j}}$ computation resources to process, and outputs $w^{\phi_{P,i}} = \Pi_{j=1}^{i} \xi^{\phi_{P,j}}$ units flow. Consequently, in $\mathcal{G}_{\tilde{\Phi}}$, if P contains an edge from $G(\phi_{P,i-1})$ to $G(\phi_{P,i})$, then the flow rate on the edge is $x^{\phi_{P,i}}$. If P contains an edge within $G(\phi_{P,i})$, then the flow rate on the edge is $w^{\phi_{P,i}}$. For unified notation, define $\phi_{P,0} = \phi_s$ and $w^{\phi_{P,0}} = 1$, since no processing has occurred. Define $\phi_{P,N+1} = \phi_t$, $w^{\phi_{P,N+1}} = w^{\phi_{P,N}}$ and $x^{\phi_{P,N+1}} = 0$, since the flow has been processed by all functions in P before entering $\phi_{P,N+1}$, and no additional processing is required at $\phi_{P,N+1}$.

Consider the capacity constraint on the link $(u, v) \in G$. Let λ_P denote the units of input service flow on P. If P contains an edge $(u(\phi_{P,i}), v(\phi_{P,i}))$, then the service flow contributes $w^{\phi_{P,i}} \lambda_P$ units flow to (u, v). Let $\mathcal{P}$ denote the set of paths in $\mathcal{G}_{\tilde{\Phi}}$. The total flow rate cannot exceed the link capacity.

$$\sum_{P \in \mathcal{P}} \sum_{\{\phi_{P,i} | (u(\phi_{P,i}), v(\phi_{P,i})) \in P\}} w^{\phi_{P,i}} \lambda_P \le \mu_{uv}, \ \forall (u, v) \in G. \quad (5)$$

Consider the computation capacity constraint on node u. If P contains an edge $(u(\phi_{P,i-1}), u(\phi_{P,i}))$, then the service flow occupies $x^{\phi_{P,i}} \lambda_P$ computation resources of u.

$$\sum_{P \in \mathcal{P}} \sum_{\{\phi_{P,i} | (u(\phi_{P,i-1}), u(\phi_{P,i})) \in P\}} x^{\phi_{P,i}} \lambda_P \le \mu_u, \ \forall u \in G. \quad (6)$$

The objective is to maximize the sum of flow rates on all paths $P \in \mathcal{P}$, under Constraints (5), (6) and non-negativity constraints $\lambda_P \ge 0, \forall P \in \mathcal{P}$.

$$\max \sum_{P \in \mathcal{P}} \lambda_P. \quad (7)$$

The path-based formulation is clean and the characterization of the path in $\mathcal{G}_{\tilde{\Phi}}$ provides a building block for algorithms that compute the minimum-cost path as a subroutine developed later in this paper.

Both the flow-based formulation and the path-based formulation compute the maximum elastic service flow λ^*, which characterizes the capacity region of the network. The path-based formulation is convenient for algorithm design. It shows that supporting the maximum flow involves finding flows on possibly many paths. In the next section, we will develop a dynamic algorithm that, at each time slot, effectively chooses one path (one configuration, route, and processing locations) for each arriving service to achieve the optimal split over time.

IV. Dynamic Joint Network and Service Optimization to Maximize Throughput

Having characterized the maximum achievable throughput λ^* for elastic services, we now turn to dynamic scenarios with stochastic arrivals. We develop a dynamic algorithm for joint optimization of packet routing, function computation, elastic service configuration, and scheduling to maximize the throughput.

Consider a time-slotted system $t \in \{0, 1, 2, \dots\}$. Let $A(t)$ denote the number of exogenous service arrivals to s at the beginning of slot t. We assume that $A(t)$ is independently and identically distributed over time. Let $\lambda = \mathbb{E}(A(t))$ denote the average arrival rate of the elastic service. We aim to develop a throughput-optimal policy that stabilizes the network for any λ below the capacity λ^* computed in Section III.

We introduce a set of virtual queues $\{\tilde{Q}_{uv}(t), \forall (u, v) \in E\}$ and $\{\tilde{Q}_u(t), \forall u \in V\}$ corresponding to the links and nodes of the distributed computing network G. If a unit service enters $\mathcal{G}_{\tilde{\Phi}}$ and uses path P, then it contributes a total load of $\sum_{\{\phi_{P,i} | (u(\phi_{P,i}), v(\phi_{P,i})) \in P\}} w^{\phi_{P,i}}$ to the link (u, v) (Section III-B). Suppose that these loads enter the virtual queue of link (u, v) immediately. If $A(t)$ services enter the network at time t, the number of arrivals to $\tilde{Q}_{uv}(t)$ is denoted by $A_{uv}(t)$.

$$A_{uv}(t) = \sum_{\{\phi_{P,i} | (u(\phi_{P,i}), v(\phi_{P,i})) \in P\}} w^{\phi_{P,i}} A(t).$$

Similarly, a unit service contributes a total load of $\sum_{\{\phi_{P,i} | (u(\phi_{P,i-1}), u(\phi_{P,i})) \in P\}} x^{\phi_{P,i}}$ to the computing node u. Suppose that these loads enter the virtual queue of node u immediately. For $A(t)$ service arrivals, the number of arrivals to $\tilde{Q}_u(t)$ is denoted by $A_u(t)$.

$$A_u(t) = \sum_{\{\phi_{P,i} | (u(\phi_{P,i-1}), u(\phi_{P,i})) \in P\}} x^{\phi_{P,i}} A(t).$$

The service rates of the virtual queues are the link transmission rates or node computation rates. The virtual queue lengths evolve according to the following recursions, where $(a)^+ = \max(a, 0)$.

$$\tilde{Q}_{uv}(t+1) = (\tilde{Q}_{uv}(t) + A_{uv}(t) - \mu_{uv})^+, \quad (8a)$$

$$\tilde{Q}_u(t+1) = (\tilde{Q}_u(t) + A_u(t) - \mu_u)^+. \quad (8b)$$

We study the joint dynamic routing and service configuration policy π^* by choosing $P \in \mathcal{G}_{\tilde{\Phi}}$ that minimizes

$$\begin{aligned} &\sum_{(u,v) \in E} \tilde{Q}_{uv}(t) A_{uv}(t) + \sum_{u \in V} \tilde{Q}_u(t) A_u(t) \\ =&A(t) \Big\{ \sum_{(u(\phi_{P,i}), v(\phi_{P,i})) \in \mathcal{G}_{\tilde{\Phi}}} w^{\phi_{P,i}} \tilde{Q}_{uv}(t) 1\{(u(\phi_{P,i}), v(\phi_{P,i})) \in P\} \\ &+ \sum_{(u(\phi_{P,i-1}), u(\phi_{P,i})) \in \mathcal{G}_{\tilde{\Phi}}} x^{\phi_{P,i}} \tilde{Q}_u(t) 1\{(u(\phi_{P,i-1}), u(\phi_{P,i})) \in P\} \Big\}. \end{aligned} \quad (9)$$

Let the unit cost of using the edge $(u(\phi_{P,i}), v(\phi_{P,i})) \in \mathcal{G}_{\tilde{\Phi}}$ be $w^{\phi_{P,i}} \tilde{Q}_{uv}(t)$ and the unit cost of using the edge $(u(\phi_{P,i-1}), u(\phi_{P,i})) \in \mathcal{G}_{\tilde{\Phi}}$ be $x^{\phi_{P,i}} \tilde{Q}_u(t)$. To compute the minimum-cost path, the classical shortest-path algorithm does not directly work since the unit costs depend on the choice of path P. The reason is that different service functions may have different flow scalings. For example, for two service chains

Algorithm 1 Minimum-cost path computation using dynamic programming

Input: Product graph $\mathcal{G}_{\tilde{\Phi}}$, source node $s(\phi_s)$, destination node $t(\phi_t)$, unit cost $c(e)$ for each edge $e \in \mathcal{G}_{\tilde{\Phi}}$.

Output: Minimum-cost path from $s(\phi_s)$ to $t(\phi_t)$ in $\mathcal{G}_{\tilde{\Phi}}$.

1: Compute the pairwise minimum-cost path in each copy of G in $\mathcal{G}_{\tilde{\Phi}}$. Let $d(u(\phi_i), v(\phi_i))$ denote the minimum cost of the path from $u(\phi_i)$ to $v(\phi_i)$.
2: **for** ϕ_i in reverse topological order among nodes in $G_{\tilde{\Phi}}$ **do**
3: **for** $u(\phi_i) \in G(\phi_i)$ **do**
4: Let $d(u(\phi_i), t(\phi_t))$ denote the minimum cost of serving one unit output flow of service function ϕ_i from node u to node t. Note that the cost includes the cost of processing all successor service functions and the cost of transmission.
5: **for** $v(\phi_i) \in G(\phi_i)$ **do**
6: Let $d(v(\phi_i))$ denote the minimum cost of serving one unit output flow of service function ϕ_i from node u to node t, given that the successor service function is processed at node v.
7: $d(v(\phi_i)) = \min_{\phi_j \in \delta^+(\phi_i)}[r^{\phi_j} c(v(\phi_i), v(\phi_j)) + \xi^{\phi_j} d(v(\phi_j), t(\phi_t))]$.
8: **end for**
9: $d(u(\phi_i), t(\phi_t)) = \min_{v(\phi_i)}\{d(u(\phi_i), v(\phi_i)) + d(v(\phi_i))\}$.
10: **end for**
11: **end for**
12: The minimum cost of the path from $s(\phi_s)$ to $t(\phi_t)$ is $d(s(\phi_s), t(\phi_t))$. The path can be reconstructed by tracking which minimizing ϕ_j and $v(\phi_j)$ were chosen in each step and concatenating the paths.

$\phi_s - \phi_a - \phi_c$ and $\phi_s - \phi_b - \phi_c$, if the flow scaling of ϕ_a and ϕ_b is different, then for a unit service flow entering ϕ_s, the input flow to ϕ_c is different depending on whether ϕ_a or ϕ_b has been processed.

We develop Algorithm 1 based on dynamic programming to compute the minimum-cost path P. At each time slot t, the unit cost for $(u(\phi_i), v(\phi_i))$ is $\tilde{Q}_{uv}(t)$ for all $\phi_i \in G_{\tilde{\Phi}}$, and the unit cost for $(u(\phi_i), u(\phi_j))$ is $\tilde{Q}_u(t)$ for all $\phi_j \in \delta^+(\phi_i), \phi_i, \phi_j \in G_{\tilde{\Phi}}$. The performance of the algorithm is given by Theorem 3.

Theorem 3. *The minimum-cost path can be computed in $O(|V|^2|V_{\tilde{\Phi}}|^2 + |V||E| + |V|^2 \log |V|)$ time for the elastic service arrivals at each time slot using Algorithm 1, where $|V|$ is the number of nodes in G, $|E|$ is the number of edges in G, and $|V_{\tilde{\Phi}}|$ is the number of nodes in $G_{\tilde{\Phi}}$.*

Proof: In Step 1, the pairwise minimum-cost path can be computed using the shortest-path algorithm on one copy of G, and the other copies have the same edge costs and the same minimum-cost paths. In a directed graph G with $|V|$ nodes and $|E|$ edges and non-negative edge weights, Dijkstra's algorithm can be implemented using Fibonacci heaps with time complexity $O(|E|+|V|\log|V|)$ [19]. Running Dijkstra's algorithm $|V|$ times, once from each node, computes all-pairs shortest paths in $O(|V||E| + |V|^2 \log |V|)$ time.

Both $d(v(\phi_i))$ and $d(u(\phi_i), t(\phi_t))$ are the cost-to-go functions to serve one unit output flow of service function ϕ_i. In Step 7, $d(v(\phi_i))$ is obtained by restricting the computation of the immediate successor service function $\phi_j \in \delta^+(\phi_i)$ at node v. The optimal successor service function ϕ_j^* is obtained by taking the minimum of the sum cost. To process one unit flow, the r^{ϕ_j} units computation incurs $r^{\phi_j} c(v(\phi_i), v(\phi_j))$ cost and the scaled ξ^{ϕ_j} units output flow incurs $\xi^{\phi_j} d(v(\phi_j), t(\phi_t))$ cost from node v to the destination. Since $G_{\tilde{\Phi}}$ is a directed acyclic graph, there exists a reverse topological order of ϕ_i, such that $d(v(\phi_j), t(\phi_t))$ $(\forall \phi_j \in \delta^+(\phi_i))$ has been computed before $d(u(\phi_j), t(\phi_t))$. The computation of Step 7 takes $O(|V_{\tilde{\phi}}|)$ time since $|\delta^+(\phi_i)| < |V_{\tilde{\phi}}|$.

Cost $d(u(\phi_i), t(\phi_t))$ is obtained by choosing the optimal computation node v^* for processing the immediate successor service function that minimizes the sum of the cost of transmitting flow from $u(\phi_i)$ to $v(\phi_i)$ and the cost-to-go $d(v(\phi_i))$. The computation of Step 9 takes $O(|V|)$ time.

The minimum-cost path P is the concatenation of edges $(v^*(\phi_i), v^*(\phi_j^*))$, which represent processing function ϕ_j^* at node v^*, and the shortest paths between two consecutive function computation nodes.

Step 2 requires $O(|V_{\tilde{\Phi}}|)$ iterations. Steps 3 and 5 each require $O(|V|)$ iterations. Therefore, the total time complexity from Step 2 to Step 11 is $O(|V|^2|V_{\tilde{\Phi}}|^2)$. The total time complexity of the algorithm is $O(|V|^2|V_{\tilde{\Phi}}|^2 + |V||E| + |V|^2 \log |V|)$. ∎

The minimum cost path computed by the Algorithm 1 determines the transmission routes, processing locations, and the configuration of the elastic service according to Theorem 1. The running time is polynomial even though there could be an exponential number of service configurations and paths. If at each time slot t, a policy π^* chooses the minimum cost path according to Algorithm 1, then Theorem 4 shows that the policy stabilizes the virtual queuing system for any arrival rate $\lambda < \lambda^*$.

Theorem 4. *Under routing policy π^*, the virtual queue process $\{\tilde{Q}(t)\}$ is strongly stable for any $\lambda < \lambda^*$, where strong stability is defined as*

$$\limsup_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \Big\{ \sum_{(u,v)\in E} \mathbb{E}[\tilde{Q}_{uv}(t)] + \sum_{u\in V} \mathbb{E}[\tilde{Q}_u(t)] \Big\} < \infty.$$

The proof follows from the standard Lyapunov drift analysis by defining a quadratic Lyapunov function $L(\tilde{Q}(t)) = \sum_{(u,v)\in E} \tilde{Q}_{uv}^2(t) + \sum_{u\in V} \tilde{Q}_u^2(t)$, and then computing the one-slot conditional drift of $L(\tilde{Q}(t))$.

$$\begin{aligned}\Delta(\tilde{Q}(t)) &\stackrel{\text{def}}{=} \mathbb{E}\big(L(\tilde{Q}(t+1)) - L(\tilde{Q}(t))|\tilde{Q}(t)\big) \\ &\le B + 2 \sum_{(u,v)\in E} \tilde{Q}_{uv}(t)\big\{\mathbb{E}[A_{uv}(t)|\tilde{Q}(t)] - \mu_{uv}\big\} \\ &\quad + 2 \sum_{(u,v)\in E} \tilde{Q}_u(t)\big\{\mathbb{E}[A_u(t)|\tilde{Q}(t)] - \mu_u\big\}, \qquad (10)\end{aligned}$$

where B is finite if the second moment of $A(t)$ is finite. Since π^* minimizes Eq. (9), we compare $\Delta(\tilde{Q}(t))$ under policy π^* with that under a randomized policy that supports $\lambda^*/(1+\epsilon)$ for any $\epsilon > 0$, and prove that the drift $\mathbb{E}[L(\tilde{Q}(t+1)) - L(\tilde{Q}(t))]$ is negative when the queue length $\tilde{Q}(t)$ is large; hence, we obtain the strong stability result. The detailed proof is omitted.

Stability of virtual queues implies that the average arrival rate to each link (u,v) is no more than the transmission capacity μ_{uv}, and the average processing load at node u is no more than the computation capacity μ_u. The total load that a service contributes to the queue Q_{uv} at a physical link (u,v) is the same as the load that it contributes to virtual queue $\tilde{Q}_{uv}$ under the same routing, and the total load that it contributes to the queue Q_u at a computing node u is the same as the load that it contributes to virtual queue $\tilde{Q}_u$ under the same service function processing location. An extended nearest-to-origin (ENTO) scheduling policy guarantees queue stability if the average arrival rate is no more than the service rate in each queue [12], [13], [20]. Under the ENTO policy, when two packets are in the same queue, the packet that has traversed a smaller number of hops has higher priority for transmission. The processing of a service function also counts towards a hop. Using a similar proof as that in [12], Theorem 5 shows that the physical queues are rate stable.

Theorem 5. *Under routing policy π^* and the ENTO scheduling policy, all physical queues are rate stable for any arrival rate $\lambda < \lambda^*$, where rate stability is defined as*

$$\lim_{t\to\infty} \frac{Q_{uv}(t)}{t} = 0, \quad \textit{w.p. } 1, \ \forall (u,v) \in E;$$

$$\lim_{t\to\infty} \frac{Q_u(t)}{t} = 0, \quad \textit{w.p. } 1, \ \forall u \in V.$$

V. Network Utility Maximization for Elastic Services

The above throughput-optimal policy does not differentiate between service configurations as long as they can be processed; in practice, different configurations may yield different utilities. For example, a high-quality video processing configuration might provide better user experience (higher utility) but use more resources than a low-quality configuration. In this section, we study how to maximize overall utility in the elastic service scenario.

Each configuration $\Phi_P \in \tilde{\Phi}$ is associated with a utility function $U_{\Phi_P}(r_{\Phi_P})$, which gives the utility (per unit time) of serving at rate r_{Φ_P} using configuration Φ_P. We assume the utility functions $U_{\Phi_P}(r_{\Phi_P})$ are increasing and concave. The Network Utility Maximization (NUM) problem is to choose a control policy π that maximizes the total utility of the delivered service flow, subject to network stability. Let $R^{\pi}_{\Phi_P}(T)$ denote the number of services processed by configuration Φ_P and delivered to the destination under π by time T. Let $Q^{\pi}_{uv}(T)$ denote the queue length at link (u,v) and let $Q^{\pi}_{u}(T)$ denote the queue length at node u under π at time T. The NUM problem finds a policy π^* that solves the following problem.

$$\begin{aligned} \max_{\pi} \quad & \mathbb{E} \sum_{\Phi_P \in \tilde{\Phi}} U_{\Phi_P}(r_{\Phi_P}) \qquad (11)\\ \text{s.t.} \quad & \lim_{T\to\infty} \frac{R^{\pi}_{\Phi_P}(T)}{T} = r_{\Phi_P}, \forall \Phi_P \in \tilde{\Phi}, \text{ w.p. } 1, \\ & \lim_{T\to\infty} \frac{1}{T}\Big[\sum_{(u,v)\in E} Q^{\pi}_{uv}(T) + \sum_{u\in V} Q^{\pi}_{u}(T)\Big] = 0, \text{ w.p. } 1. \end{aligned}$$

In traditional NUM, the optimal flow rate r lies on the boundary of the capacity region if there is enough backlog [21]. There exists a policy that simultaneously maximizes network throughput and utility. However, to support elastic services, the policy that maximizes utility may not necessarily maximize throughput. Consider an example of a single computation node in G where the computation resource is the bottleneck and has capacity μ. To process a single elastic service function $\tilde{\phi}$, one configuration ϕ_1 requires 1 unit computation resource for each unit service flow and has 1 unit utility after completing the service, and another configuration ϕ_2 requires 2 units computation resource for each unit service flow and has 3 units utility after completing the service. Suppose that communication resources are sufficient. The policy to maximize network throughput is to choose configuration ϕ_1, and the maximum service flow rate is μ. The policy to maximize network utility is to choose configuration ϕ_2, and the maximum utility is $3\mu/2$ while the service flow rate is $\mu/2$, where there is a gap to the maximum throughput.

To solve NUM, we apply the drift-plus-penalty framework [21]–[23] to minimize the objective function

$$\begin{aligned} &\sum_{(u,v)\in E} \tilde{Q}_{uv}(t)A_{uv}(t) + \sum_{u\in V} \tilde{Q}_u(t)A_u(t) - \beta \sum_{\Phi_P\in\tilde{\Phi}} U_{\Phi_P}(A_{\Phi_P}(t)) \\ &= \sum_{\Phi_P\in\tilde{\Phi}} \Big\{ A_{\Phi_P}(t) \sum_{(u(\phi_{P,i}),v(\phi_{P,i}))\in\mathcal{G}_{\tilde{\Phi}}} w^{\phi_{P,i}} \tilde{Q}_{uv}(t) 1\{(u(\phi_{P,i}),v(\phi_{P,i}))\in P\} \\ &\qquad + A_{\Phi_P}(t) \sum_{(u(\phi_{P,i-1}),u(\phi_{P,i}))\in\mathcal{G}_{\tilde{\Phi}}} x^{\phi_{P,i}} \tilde{Q}_{u}(t) 1\{(u(\phi_{P,i-1}),u(\phi_{P,i}))\in P\} \\ &\qquad - \beta U_{\Phi_P}(A_{\Phi_P}(t)) \Big\}. \qquad (12) \end{aligned}$$

Eq. (12) differs from Eq. (9) in the last penalty term, where β is a positive constant and $U_{\Phi_P}(A_{\Phi_P}(t))$ is the utility of processing $A_{\Phi_P}(t)$ services using service configuration Φ_P.

Since the utility function depends on the configuration Φ_P, the minimum-cost routing Eq. (9) should be computed for each configuration $\Phi_P \in \tilde{\Phi}$, instead of using Algorithm 1 which computes the minimum-cost routing over all configurations. The computation of the minimum-cost routing $P(t)$ applies the classical shortest path algorithm for each $\Phi_P \in \tilde{\Phi}$. Given Φ_P, the shortest path from $s(\phi_s)$ to $t(\phi_t)$ can be computed by restricting the search space within the copies of G that correspond to service functions in Φ_P and the edges connecting adjacent copies. The cost of an edge $(u(\phi_{P,i}), v(\phi_{P,i}))$ is the product of virtual queue length $\tilde{Q}_{uv}(t)$ and the scaling factor $w^{\phi_{P,i}}$. The cost of an edge $(u(\phi_{P,i-1}), u(\phi_{P,i}))$ is the product of virtual queue length $\tilde{Q}_u(t)$ and the scaling factor

$x^{\phi_{P,i}}$. Recall that $w^{\phi_{P,i}}$ and $x^{\phi_{P,i}}$ are defined in Section III-B. Denote the shortest path corresponding to Φ_P as $P^*(\Phi_P)$ and its length by $C_{\Phi_P}(t)$.

The optimal number of services processed by configuration Φ_P of the elastic service function chain at time t is denoted by $r^*_{\Phi_P}(t)$, and is obtained by solving the following mathematical program.

$$\begin{aligned} \min \sum_{\Phi_P \in \tilde{\Phi}} & \Big[C_{\Phi_P}(t) r_{\Phi_P}(t) - \beta U_{\Phi_P}(r_{\Phi_P}(t)) \Big] \qquad (13) \\ \text{s.t.} \sum_{\Phi_P \in \tilde{\Phi}} & r_{\Phi_P}(t) \leq A(t), \\ & r_{\Phi_P}(t) \geq 0, \quad \forall \Phi_P \in \tilde{\Phi}. \end{aligned}$$

If $U_{\Phi_P}(x)$ is a linear function $U_{\Phi_P}(x) = h_{\Phi_P} x$, then the optimal solution is obtained by selecting Φ_{P^*} that minimizes $C_{\Phi_P}(t) - V h_{\Phi_P}$, and setting $r^*_{\Phi_{P^*}}(t) = A(t)$ if $C_{\Phi_{P^*}}(t) - V h_{\Phi_{P^*}} < 0$ and $r^*_{\Phi_{P'}}(t) = 0$ for all the other $\Phi_{P'} \in \tilde{\Phi} \setminus \{\Phi_{P^*}\}$. If $C_{\Phi_P}(t) - V h_{\Phi_P} \geq 0$, it is optimal to not admit any packets. If $U_{\Phi_P}(x)$ is concave and differentiable, then it can be solved using the optimal flow [24] techniques where $C_{\Phi_P} - V U'_{\Phi_P}(r^*_{\Phi_P}(t))$ achieves the same non-positive minimum for $r^*_{\Phi_P}(t) > 0$, and $C_{\Phi_{P'}} - V U'_{\Phi_{P'}}(r^*_{\Phi_{P'}}(t))$ is larger than the minimum for all $r^*_{\Phi_{P'}}(t) = 0$.

The dynamic joint network and service control is summarized in Algorithm 2.

Algorithm 2 Dynamic Joint Network and Service Control to Maximize Network Utility.

Initialization: $\tilde{Q}_{uv}(0) = \tilde{Q}_u(0) = 0$, $\forall (u,v) \in E, u \in V$.
At each time slot t:

1) **Route Selection**. For an incoming elastic service, construct a product graph $\mathcal{G}_{\tilde{\Phi}}$. For each $\Phi_P \in \tilde{\Phi}$, compute the shortest path $P^*(\Phi_P)$ from $s(\phi_s)$ to $t(\phi_t)$, where the cost of an edge $(u(\phi_{P,i}), v(\phi_{P,i}))$ is $w^{\phi_{P,i}} \tilde{Q}_{uv}(t)$, and the cost of an edge $(u(\phi_{P,i-1}), u(\phi_{P,i}))$ is $x^{\phi_{P,i}} \tilde{Q}_u(t)$. Path $P^*(\Phi_P)$ specifies the routing and function processing nodes.
2) **Admission Control and Service Configuration Selection**. Compute $r^*_{\Phi_P}(t), \forall \Phi_P \in \tilde{\Phi}$ that minimize (13). If $r^*_{\Phi_P}(t) > 0$, then $r^*_{\Phi_P}(t)$ service arrivals are processed by service configuration Φ_P.
3) **Packet Scheduling**. Each physical link transmits packets and each computation node processes packets according to the ENTO policy.
4) **Virtual Queues Update**. Virtual queue lengths evolve according to Eq. (8), where $A_{uv(t)} = \sum_{\Phi_P \in \tilde{\Phi}} \sum_{\{\phi_{P,i} | (u(\phi_{P,i}), v(\phi_{P,i})) \in P\}} w^{\phi_{P,i}} r^*_{\Phi_P}(t)$ and $A_u = \sum_{\Phi_P \in \tilde{\Phi}} \sum_{\{\phi_{P,i} | (u(\phi_{P,i-1}), u(\phi_{P,i})) \in P\}} x^{\phi_{P,i}} r^*_{\Phi_P}(t)$.

Algorithm 2 ensures network stability and that the achieved utility is within $O(1/\beta)$ of the optimal. The proof of Theorem 6 is based on Lyapunov optimization and follows similar techniques in [22]. The main difference is that admission control for different commodities can be computed independently in [22], while different service configurations share the same arrival and the decisions are computed jointly in our problem (13), similar to the Type 1 flow control constraint in [21]. The detailed proof is omitted due to space constraints.

Theorem 6. *Algorithm 2 solves the NUM problem* (11) *and achieves a utility of at least* $U^* - O(1/\beta)$*, where* U^* *is the optimal utility and* $\beta > 0$*.*

VI. Simulations

We evaluate the proposed dynamic control algorithms via simulation, considering both randomly generated elastic service chains and specific elastic service chains that model neural architecture search. Our results verify that the dynamic algorithms achieve optimal throughput and utility, and perform significantly better than baselines. For simplicity, we consider linear utility functions and Poisson arrivals.

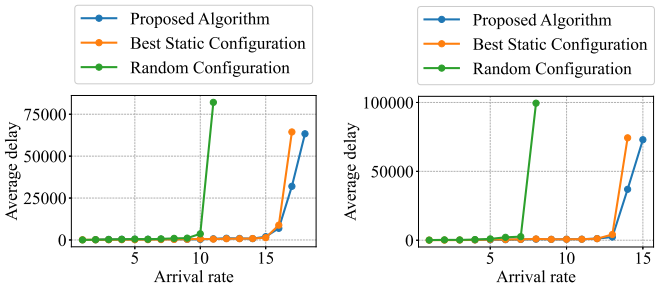
We compare our proposed algorithm with two baseline algorithms: (i) a Random Configuration policy that randomly chooses a service configuration at each time slot, and (ii) a Best Static Configuration policy that chooses the best fixed service configuration over all time slots. For throughput maximization, the best static service configuration maximizes throughput. For utility maximization, the best static service configuration has the maximum utility coefficient.

A. Evaluation on Randomly Generated Elastic Service Chains

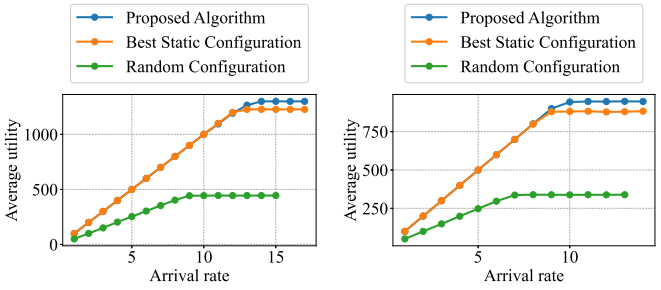
First, we simulate a synthetic elastic service chain with varying sizes. Consider the graph representation for an elastic service function chain $G_{\tilde{\Phi}}$ with l layers, m functions per layer, and any function in layer i can feed into any function in layer $i+1$ with i.i.d. probability d. We test two sizes: (i) Small: $l = 6, m = 8, d = 0.5$ and (ii) Medium: $l = 8, m = 10, d = 0.6$. The computation requirement for each function is an integer uniformly distributed between 1 and 5, and the communication scaling factor is uniform at random from 0.5 to 2.

We generate a randomly connected computing network comprising 30 nodes with an average degree of 2. Each node's computation capacity is a uniform random integer between 1 and 5, while each link has a bidirectional communication capacity that is a uniform random integer between 1 and 10. We evaluate throughput over 10^5 time slots. For utility maximization, we set the control parameter $\beta = 100$ and run the simulation for 10^6 time slots.

Fig. 3 shows the average packet delay and average utility for elastic service chains of varying sizes. Our dynamic algorithm maintains very low delays up until λ approaches the capacity λ^*, whereas the baseline policies experience rapidly increasing delays (and instability) at much lower rates. The flexibility of service configuration increases network throughput, since requests are served by more than one configuration. For utility maximization, our algorithm achieves similar utility compared with the Best Static Configuration policy when arrival rates are low, where all arrivals can be processed by the configuration that has the highest utility. When arrival rates are high, our policy may choose a configuration that has a lower utility



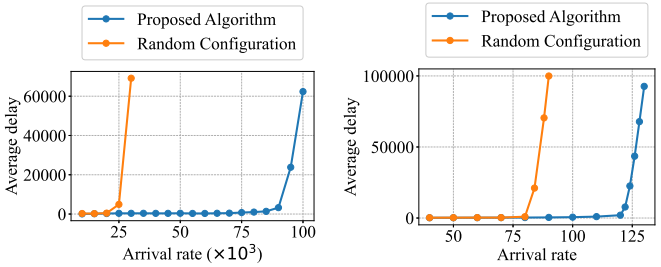
(a) Small: Average packet delay. (b) Medium: Average packet delay.



(c) Small: Average utility. (d) Medium: Average utility.

Fig. 3. Evaluation on randomly generated elastic service chains.

to complete the service, but occupies even lower network resources, yielding a higher average utility.



(a) NASNet: Average packet delay. (b) Llama-NAS: Average packet delay.

Fig. 4. Evaluation on neural architecture search applications.

B. Evaluation on Neural Architecture Search Applications

We next simulate two elastic service function chains drawn from neural architecture search.

1) NASNet (CIFAR-10) case [25]: The first scenario is based on the NASNet search space for image classification. The network architecture consists of interconnected normal cells and reduction cells. For CIFAR-10, a typical model has three blocks of normal cells (which preserve feature map size) interleaved with two reduction cells (which downsample the feature map). We treat the number of normal cells as elastic: each block can contain between 1 and 7 normal cells (as in [25], which sets 7 as the max). Thus, the service chain has the structure: input $\rightarrow$ (Block 1: $\tilde{\phi}_1$ normal cells) $\rightarrow$ (Reduction cell) $\rightarrow$ (Block 2: $\tilde{\phi}_2$ normal cells) $\rightarrow$ (Reduction cell) $\rightarrow$ (Block 3: $\tilde{\phi}_3$ normal cells) $\rightarrow$ output. Each $\tilde{\phi}_i$ is an elastic function representing a sequence of 1–7 normal cells.

We estimate resource requirements as follows: for a 32×32 image with 32 initial channels, each normal cell involves roughly 2.4×10^7 FLOPs, and each reduction cell about 9×10^6 FLOPs. The normal cells do not change the spatial resolution, while each reduction cell halves the image height and width (reducing data size by a factor of 4). We simulate a computing network of 9 nodes arranged in a 3×3 grid. Each node has a computation capacity of 1 TFLOP/s, and each link can transfer up to 60 GB/s.

2) Llama-NAS (Transformer) case [26]: The elasticity lies in the intermediate layer widths of a Transformer model. We consider a model of 24 layers, where each layer's feed-forward network (MLP) can have either 5504 or 11008 hidden units (these two choices come from [26]). Thus, each layer is an elastic function $\tilde{\phi}$ with two options. We estimate that processing one token (sequence length 1000) with one layer of width 5504 requires about 1.06×10^{11} FLOPs, and with width 11008 requires 2.11×10^{11} FLOPs. The sequence length remains 1000 throughout (no change in data size and $\xi_\phi = 1$). We simulate a 16-node physical network (4×4 grid) with each node offering 20 TFLOP/s and each link offering 60 GB/s bandwidth.

Due to the large number of configurations, we compare our algorithm with the Random Configuration baseline. Our algorithm is able to support significantly higher arrival rates before delays grow. The throughput is approximately 3 times that of the Random Configuration for NASNet and 50% higher for Llama-NAS. Since larger size models have higher accuracy, we assign larger utilities to larger models. The utilities are chosen in the range of 5 to 10. To validate admission control and service configuration under overload, we consider arrival rates outside the capacity region. Specifically, for NASNet we set the arrival rates to 100×10^3 (ours) and 30×10^3 (Random Configuration) and run for 1×10^5 time slots; for Llama-NAS we set them to 130 (ours) and 90 (Random Configuration) and run for 3×10^5 time slots. To capture steady-state behavior, we compute the average utility over the final 2×10^4 time slots. The time average utilities achieved by our algorithm (compared to the Random Configuration) are 4.02×10^5 (compared to 1.62×10^5) for NASNet; and 6.38×10^2 (compared to 4.64×10^2) for Llama-NAS.

VII. Conclusion

Elastic services have flexible processing configurations and can be adaptive to changing network conditions. We develop a simple yet representative model for elastic services, and formulate the maximum elastic service flow problem. We characterize the network capacity region for elastic services using linear program formulations, and develop online algorithms that jointly optimize routing, processing, scheduling, and configuring elastic services to maximize network throughput and utility under stochastic arrivals. Joint optimization of network resources and service configurations substantially improves the network's ability to support elastic services, as demonstrated in applications such as distributed machine learning.

REFERENCES

[1] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *Proc. International Conference on Learning Representations (ICLR)*, 2017.

[2] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "MnasNet: Platform-aware neural architecture search for mobile," in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 2820–2828.

[3] H. Xie, Z. Qin, G. Y. Li, and B. Juang, "Deep learning enabled semantic communication systems," *IEEE Transactions on Signal Processing*, vol. 69, pp. 2663–2675, 2021.

[4] X. Wang, Z. Yang, J. Wu, Y. Zhao, and Z. Zhou, "EdgeDuet: Tiling small object detection for edge-assisted autonomous mobile vision," in *Proc. IEEE INFOCOM*, 2021, pp. 1–10.

[5] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "On orchestrating virtual network functions in NFV," in *Proc. CNSM*, 2015, pp. 50–56.

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

[7] Z. Cao, S. S. Panwar, M. Kodialam, and T. V. Lakshman, "Enhancing mobile networks with software defined networking and cloud computing," *IEEE/ACM Transactions on Networking*, vol. 25, no. 3, pp. 1431–1444, 2017.

[8] L. Guo, J. Z. F. Pang, and A. Walid, "Joint placement and routing of network function chains in data centers," in *Proc. IEEE INFOCOM*, 2018, pp. 612–620.

[9] H. Feng, J. Llorca, A. M. Tulino, D. Raz, and A. F. Molisch, "Approximation algorithms for the NFV service distribution problem," in *Proc. IEEE INFOCOM*, 2017, pp. 1–9.

[10] T. Ouyang, Z. Zhou, and X. Chen, "Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 10, pp. 2333–2345, 2018.

[11] 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.

[12] J. Zhang, A. Sinha, J. Llorca, A. M. Tulino, and E. Modiano, "Optimal control of distributed computing networks with mixed-cast traffic flows," *IEEE/ACM Transactions on Networking*, vol. 29, no. 4, pp. 1760–1773, 2021.

[13] A. Sinha and E. Modiano, "Optimal control for generalized network-flow problems," *IEEE/ACM Transactions on Networking*, vol. 26, no. 1, pp. 506–519, 2018.

[14] X. Chen, C. Xu, M. Wang, Z. Wu, S. Yang, L. Zhong, and G.-M. Muntean, "A universal transcoding and transmission method for livecast with networked multi-agent reinforcement learning," in *Proc. IEEE INFOCOM*, 2021.

[15] V. Sohn, S. Kim, and H.-W. Lee, "Joint frame drop and object detection task offloading for mobile devices via RL with Lyapunov optimization," *IEEE Transactions on Mobile Computing*, 2025.

[16] G. Even, M. Medina, and B. Patt-Shamir, "On-line path computation and function placement in SDNs," in *Stabilization, Safety, and Security of Distributed Systems*. Springer International Publishing, 2016, pp. 131–147.

[17] G. Even, M. Rost, and S. Schmid, "An approximation algorithm for path computation and function placement in sdns," in *Structural Information and Communication Complexity*. Springer International Publishing, 2016, pp. 374–390.

[18] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[19] M. Fredman and R. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," in *Proc. 25th Annual Symposium on Foundations of Computer Science (FOCS)*, 1984, pp. 338–346.

[20] D. Gamarnik, "Stability of adaptive and nonadaptive packet routing policies in adversarial queueing networks," *SIAM Journal on Computing*, vol. 32, no. 2, pp. 371–385, 2003.

[21] L. Georgiadis, M. J. Neely, and L. Tassiulas, "Resource allocation and cross-layer control in wireless networks," *Foundations and Trends® in Networking*, vol. 1, no. 1, pp. 1–144, 2006.

[22] A. Sinha and E. Modiano, "Network utility maximization with heterogeneous traffic flows," in *Proc. 16th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, 2018, pp. 1–8.

[23] M. J. Neely, *Stochastic Network Optimization with Application to Communication and Queueing Systems*. Morgan and Claypool Publishers, 2010.

[24] D. Bertsekas and R. Gallager, *Data networks (2nd ed.)*. USA: Prentice-Hall, Inc., 1992.

[25] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697–8710.

[26] A. Sarah, S. Nittur Sridhar, M. Szankin, and S. Sundaresan, "Llama-nas: Efficient neural architecture search for large language models," in *European Conference on Computer Vision*. Springer, 2024, pp. 67–74.