Fairness-aware Dynamic Rate Control and Flow Scheduling for Network Utility Maximization in Network Service Chain

Lin Gu, Member, IEEE, Deze Zeng, Member, IEEE, Sheng Tao, Song Guo, Senior Member, IEEE, Hai Jin, Fellow, IEEE, Albert Zomaya, Fellow, IEEE, and Weihua Zhuang, Fellow, IEEE

Abstract—Network Function Virtualization (NFV) decouples the traditional network functions from specific or proprietary hardware such that virtualized network functions (VNFs) can run in software form. By exploring NFV, a consecutive set of VNFs can constitute a service function chain (SFC) to provide network service. From the perspective of network service providers, how to maximize the network utility is always one of the major concerns. To this end, there are two main issues need to be considered at runtime: 1) how to handle the unpredictable network traffic burst? 2) how to fairly allocate resources among various flows to satisfy different traffic demands? In this paper, we investigate a fairness-aware flow scheduling problem for network utility maximization, with joint consideration of resource allocation and rate control. Based on a discrete time queuing model, we propose a low-complexity online distributed algorithm using Lyapunov optimization framework, which can achieve arbitrary optimal utility with different fairness levels by tuning the fairness bias parameter. We theoretically analyze the optimality of the algorithm and evaluate its efficiency by both simulation and testbed based experiments.

Index Terms—NFV, Flow Scheduling, Rate Control, Fairness, Network Utility Maximization.

I. INTRODUCTION

RECENTLY, Network Function Virtualization (NFV) emerges as a promising technology to provide various network functions (e.g., firewall, deep packet inspection, instruction detection system and network optimizers) on standard commodity devices such as x86 servers, rather than relying on traditional dedicated hardware. By exploring the advantages of NFV, network operators or network service providers can deploy various network functions as virtual network function (VNF) instances in a fast and flexible way, and adaptively change VNF configurations and balance network traffic according to the network requirements and performance degradation. To avoid network overburdening, one potential approach is to take rate control [8] into consideration. Note that different network services are usually represented by various SFCs, requiring different types and quantities of network resources. Therefore, it is essential to study how to allocate the resources among different network flows and how to control flow rates with guaranteed quality-of-service (QoS). As the network resources are usually shared by many different flows, besides the QoS of each flow, the fairness among these flows cannot be ignored. For example, consider a server with a resource capacity of $C$, as shown in Fig. 2, with sufficient link bandwidth. There are two flows, $flow_1$ and $flow_2$, to be processed by VNF instances $nf_1$ and $nf_2$.
respective. Both \( n_f 1 \) and \( n_f 2 \) are deployed in the same server and share its resources. VNFs \( n_f 1 \) and \( n_f 2 \) consume \( C \) and \( 2C \) resource units to process one packet, respectively. The maximum throughput 1 can be achieved when admitting only packets of \( f_{low} 1 \) and aggressively rejecting all packets of \( f_{low} 2 \) as shown in resource allocation strategy 1. Obviously, this allocation is unfair and unacceptable. To pursue service fairness, another solution is to admit packets of both \( f_{low} 1 \) and \( f_{low} 2 \) with the resource allocation of \( C/3 \) and \( 2C/3 \), achieving better fairness at the cost of a lower total throughput. It is clear that there exists a tradeoff between throughput and fairness, which should be carefully balanced via flow rate control and resource allocation. When there exist multiple VNFs for one network function in different servers, balancing fairness and throughput is challenging.

Moreover, due to the semantics diversity of SFCs and time-varying traffic demands at runtime, it is non-trivial to dynamically allocate the network resources and control the flow rate in large-scale networks. In essence, we need a runtime scheduler that can answer these questions: 1) For each network flow, how many network packets can be admitted in each server? 2) For each server processing multiple flows, how to allocate resources to each flow? 3) For each flow packet processed by current VNF, which servers with the next VNF should be selected? To answer the above questions, we study the problem of rate control and flow scheduling problem in NFV networks with flow fairness consideration. The main contributions of this paper are as follows:

- To the best of our knowledge, this is the first study to investigate the online network flow scheduling and resource allocation problem with the consideration of fairness in NFV service chain. We propose a utility function parameterized with a fairness bias to balance the tradeoff between throughput and fairness. Based on this utility function, we establish a comprehensive analytical framework by taking advantage of Lyapunov optimization.
- We propose an online distributed fairness-aware flow scheduling algorithm, which makes all control decisions distributedly according to the real-time traffic demands and network status. We formally prove that this algorithm achieves \( [O(1/V), O(V)] \) tradeoff between network utility and network congestion.
- We implement a testbed to evaluate the achievable performance of our proposed solution. Both simulation and testbed based experiments are conducted to validate the correctness and efficiency of our algorithm. We specially analyze the tradeoff between throughput and fairness based on our performance evaluation results.

The reminder of this paper is structured as follows. Our system model is introduced in Section II and the network utility maximization problem is formulated in Section III. We then propose an online distributed algorithm based on the formulation in Section IV. The performance evaluation results from both simulations and testbed are presented and discussed in Section V. Section VI summaries some recent related studies and finally Section VII concludes our work.

II. SYSTEM MODEL

In this section, we introduce our system model in this paper. The major notations are listed in Table I.

A. Network and Flow Model

An NFV network can be described as an indirected graph \( G = (N, E) \), including a set of servers, \( N \), and a set of network edges, \( E \). The server set \( N \) can be further categorized into three disjoint subsets, i.e., \( N = S \cup P \cup D \), with \( S \) as the source server set, \( P \) as processing server set and \( D \) as destination server set. A server, \( n \in N \), is equipped with \( C_n \) units of resource (e.g., total available CPU frequency in GHz) and an edge, \( e_{ij} \in E \), has communication capacity \( L_{ij} \) (e.g., total available bandwidth in Gbps). By taking advantage of the NFV technology, service providers are able to change VNF configurations to deal with various network dynamics, e.g., increasing or decreasing VNF resources when the network flow varies with time. Let \( V \) be the VNF set of all network services and \( V(n) \) be the VNF set in server \( n \in P \). Multiple VNF instances of different network function types may be placed in one single server, hence its resource will be shared by these instances.

Moreover, the resource requirements of different VNFs vary with their network function types. For example, an
by a finite upper bound rate as $A_{n}^{\text{max}}$. Therefore, the maximum total network flow rate is $A_{f}^{\text{max}} = \sum_{n} A_{n}^{\text{max}}$.

### B. Queuing Model and Control Decisions

To capture the processing procedure of all packets, we denote $Q_{n}^{f,k}(t)$ as the queue backlogs of flow $f$, requiring the $k$th network function in server $n$ at time slot $t$. Specifically, the queue backlogs at destination servers should be 0, i.e., $Q_{n}^{f,k}(t) \equiv 0, \forall n = d_{f}$. Without loss of generality, we assume that all queue buffers are finite and all queues are initially empty at time slot $t = 0$.

At each time slot, $t$, we should make three control decisions to schedule flow packets between queues, as follows:

1) Admission Control Decisions: To guarantee the system stability with limited network resources, the number of flow packets admitted to the network should be controlled. For network flow $f$, we need to decide the flow rate $R_{n}^{f}(t)$ admitted in server $n$ at any time slot $t$, without overburdening the resource capacity;

2) Load Balancing Decisions: We should determine how many packets of each queue shall be offloaded to the other servers hosting the same VNF. We define the packet rate of the $k$th network function of flow $f$ offloaded from server $i$ to $j$ as $A_{i,j,k}^{f}(t)$ at time slot $t$;

3) Resource Allocation Decisions: For each server, we shall decide how many resources (e.g., CPU cycles) shall be allocated to a VNF to enable certain desired processing rate for a flow. Let $\mu_{i,j,k}^{f}(t)$ be the packet processing rate of the $k$th network function in flow $f$ in server $n$ at time slot $t$.

### III. Problem Formulation

In this section, we provide a formal description of our problem objective for network utility maximization, with the consideration of queuing dynamics, resource capacity constraints, throughput requirement constraints.

#### A. Queuing Dynamics

At each time slot $t$, for the $k$th network function of flow $f$, its queue backlog vector can be described as $Q(t) = (Q_{n}^{f,k}(t))_{n \in \mathbb{N}, f \in \mathbb{F}, k \in [1, l(f)])$. After applying the admission control, load balancing and resource allocation decisions, the queue backlog at the next time slot $t + 1$ is updated as follows:

$$
Q_{n}^{f,k}(t + 1) = \\
\left[Q_{n}^{f,k}(t) - \mu_{i,j,k}^{f}(t) - \sum_{j \in (n)} A_{i,j,k}^{f}(t)\right]^{+} \\
\quad + \theta^{f,k-1} \mu_{f,k-1}(t) + \sum_{i \in (n)} A_{i,j,k}^{f}(t) + \lambda_{|k=1} R_{n}^{f}(t),
$$

where $\mathbb{N}$ and $\mathbb{O}$ are the input and output server set of $n$, respectively. In (1), the present queue size is calculated as $Q_{n}^{f,k}(t) - \mu_{i,j,k}^{f}(t) - \sum_{j \in (n)} A_{i,j,k}^{f}(t)^{+}$, where $|x|^{+} = \max(0, x)$. Note that each network function can have multiple VNFs located in different servers. The packets of flow $f$ can be processed by the VNF of the $k$th network function type in server $n$ at a rate of $\mu_{i,j,k}^{f}(t)$ or offloaded to the same type of VNFs in other servers at a rate of $\sum_{j \in (n)} A_{i,j,k}^{f}(t)$.

---

### TABLE I

**NOTATIONS**

| $\mathbb{N}$ | The server set |
| $\mathbb{E}$ | The network edge set |
| $\mathbb{F}$ | The network flow set |
| $\mathbb{V}$ | The VNF set in all servers |
| $\mathbb{O}(n)$ | The output server set of $n$ |
| $\mathbb{I}(n)$ | The input server set of $n$ |
| $\mathbb{B}$ | The network edge set |
| $l(f)$ | The chain length of flow $f$ (destination included) |
| $v(f,k)$ | The type of the $k$th network function of flow $f$ |
| $\nu(n)$ | The set of network functions in server $n$ |
| $L_{ij}$ | The link resource capacity of edge $(i,j) \in \mathbb{E}$ |
| $\beta_{i,j,k}$ | The required resource for $k$th network function of flow $f$ |
| $\theta_{i,j,k}$ | The scaling factor of for $k$th network function of flow $f$ |
| $R_{n}^{f}(t)$ | The resource capacity of server $n$ |
| $A_{n}^{f,k}(t)$ | The newly arrived packets of flow $f$ in server $n$ at time slot $t$ |
| $Q_{n}^{f,k}(t)$ | The queue backlog for $k$th network function of flow $f$ in server $n$ at time slot $t$ |
| $\mu_{n}^{f,k}(t)$ | The number of packet for $k$th network function of flow $f$ processed in server $n$ at time slot $t$ |
| $A_{i,j,k}^{f}(t)$ | The packets for $k$th network function of flow $f$ from server $i$ to server $j$ at time slot $t$ |
| $R_{n}^{f}(t)$ | The rate of flow $f$ admitted in server $n$ at time slot $t$ |
| $r_{f}^{t}$ | The total average throughput of flow $f$ |
Similarly, let $\theta^{f,k-1}$, $\lambda_n^{f,k-1}(t)$ and $\sum_{i \in \mathbb{N}} \lambda_n^{f,k}(t)$ be the number of packets processed by VNF of the $(k-1)$th network function in server $n$ and input packets from VNFs of the $(k-1)$th network function in other servers, respectively. As mentioned, $P^{(k)}(t)$ controls the number of admitted newly arrived packets. Packets of flow $f$ must be processed by all ordered network functions in the SFC, and newly arrived packets must be accepted from the first network function, i.e., $P^{(k)}(t) \equiv 1$ when $k = 1$, and 0 otherwise. The queue lengths of all flows are all 0 at time slot 0, i.e., $Q_n^{f,k}(0) \equiv 0, \forall n \in \mathbb{N}, f \in \mathbb{F}, k \in [1, l(f)]$. The destination servers sink all packets as soon as they arrive, and hence their queue sizes always be 0, i.e., $Q_n^{f,k}(t) \equiv 0, \forall n \in \mathbb{D}$. Nevertheless, to ensure the network stability, we must have

$$\lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T-1} \sum_{n,f,k} \mathbb{E}(Q_n^{f,k}(t)) < \infty. \quad (2)$$

**B. Resource Capacity Constraints**

As mentioned in Section II, the VNF instances are placed in different servers. For any server $n$, the flow packets can only be processed if there exists a corresponding VNF instance for the $k$th network function. Otherwise, no packet can be processed, i.e.,

$$\mu_n^{f,k}(t) = 0, \text{if } v(f, k) \not\in \mathbb{V}(n). \quad (3)$$

After processing, the flow packets are processed by the next VNF in the same server or transmitted to the next VNF in other servers. The total number of flow packets transmitted over an edge at time slot $t$ is constrained by its link capacity, i.e.,

$$\sum_{f \in \mathbb{F}} \sum_{k \in [1, l(f)]} (A_{ij}^{f,k}(t) + \lambda_{ji}^{f,k}(t)) \leq L_{ij}, (i,j) \in \mathbb{E}. \quad (4)$$

Similarly, the total number of flow packets processed in server $n$ is limited by its resource capacity $C_n$, as

$$\sum_{f \in \mathbb{F}} \sum_{k \in [1, l(f)]} \beta_n^{f,k} \mu_n^{f,k}(t) \leq C_n, \forall n \in \mathbb{N}. \quad (5)$$

**C. Throughput Constraints**

For each server $n$, regardless of the allocated resources, the admitted packets of flow $f$ at the first VNF cannot exceed the number of newly arrived packets, i.e.,

$$0 \leq R_n^{f}(t) \leq A_n^{f}, \forall n \in \mathbb{N}, f \in \mathbb{F}. \quad (6)$$

Let $r_f(t) = \sum_{n \in \mathbb{N}} R_n^{f}(t)$ and $r_f$ be the mean throughput of flow $f$, i.e.,

$$r_f = \lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T-1} \sum_{n \in \mathbb{N}} \mathbb{E}(R_n^{f}(t)), \forall f \in \mathbb{F}. \quad (7)$$

Then, the throughput of each flow is constrained by the minimum and the maximum throughput requirements to ensure the SLA. That is,

$$r_f^{\min} \leq r_f \leq r_f^{\max}, \forall f \in \mathbb{F}. \quad (8)$$

**D. Problem Objective**

Following [9], we define the utility function of flow $f \in \mathbb{F}$ as

$$U_f(r_f, \alpha_f) = \begin{cases} (1 - \alpha_f)^{-1} r_f^{-1}, & \text{if } \alpha_f \neq 1 \\ \log(r_f), & \text{if } \alpha_f = 1 \end{cases}$$

where $\alpha_f (\geq 0)$ is a fairness tuning parameter. It can be observed that the utility function is strictly non-decreasing and concave. A larger $\alpha_f$ means we prefer more fairness than throughput. When $\alpha_f = 0$, the problem is degenerated to a throughput maximization problem. By summarizing all above, our optimization problem can be formulated as follows:

NUM-Online:

$$\begin{aligned} \text{max} : & \sum_{f \in \mathbb{F}} U_f(r_f, \alpha_f), \\ \text{s.t.} : & (2), (3), (4), (5), (6) \text{ and } (8). \end{aligned} \quad (9)$$

**IV. ONLINE ALGORITHM DESIGN AND ANALYSIS**

Note that the number of newly arrived packets at the source server is unknown and unpredictable. To tackle this problem, we design an online distributed algorithm by taking the advantages of Lyapunov optimization framework, which has been widely used in online distributed network control, e.g., [14]–[17], to make close-to-optimum decisions while guaranteeing system stability.

**A. Problem Transformation using Lyapunov Optimization**

The utility function $U_f(r_f, \alpha_f)$ is non-linear. We define auxiliary variables $\gamma_f = \lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}(y_f(t))$ and let $y_f \leq r_f, \forall f \in \mathbb{F}$. (10)

For each flow $f \in \mathbb{F}$, we can transform the original problem in (9) into the following:

$$\begin{aligned} \text{max} : & \lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T-1} \sum_{f \in \mathbb{F}} \mathbb{E}(U_f(y_f(t), \alpha_f)), \\ \text{s.t.} : & (2), (3), (4), (5), (6), (8) \text{ and } (10). \end{aligned} \quad (11)$$

Applying the Jensen’s inequality, the optimal solutions of problems (9) and (11) are equal since our utility function in (9) is non-decreasing and concave [10]. To solve the new problem (11), virtual queues $H_f(t)$ is introduced to transform the inequality constraints in (10) into a queue stability problem as:

$$H_f(t+1) = [H_f(t) - R_f(t)]^+ + \gamma_f(t) \tag{12}$$

with $0 \leq \gamma_f(t) \leq A_f^{\max}$ and $H_f(0) \equiv 0, \forall f \in \mathbb{F}$. From (12), we can observe that $H_f(t+1) \geq H_f(t) - R_f(t) + \gamma_f(t)$. Summing up all inequalities over time slots $t \in \{0, 1, \ldots, T-1\}$ and then dividing the sum by $T$, we obtain $H_f(T) - H_f(0) + \frac{1}{T} \sum_{t=0}^{T-1} R_f(t) \geq \gamma_f(t)$. Taking expectations of both sides and letting $t \to \infty$, we get $\lim_{t \to \infty} \frac{\mathbb{E}(H_f(T))}{T} + r_f \geq \gamma_f$. If the virtual queues $H_f(T)$ are stable, we have $\lim_{t \to \infty} \frac{\mathbb{E}(H_f(T))}{T} = 0$ and constraints (10) are satisfied. Therefore, when the stability of $H_f(t)$ is guaranteed, the inequality constraints (10) are always satisfied.
Similarly, the inequality constraints (8) can also be transformed into the following queue stability problem as

\[ Y_f(t + 1) = (Y_f(t) - R_f(t))^+ + r_f^\min, \quad (13) \]

and

\[ Z_f(t + 1) = (Z_f(t) - r_f^\max)^+ + R_f(t). \quad (14) \]

Let \( H(t) = (H_f, f \in F) \), \( Y(t) = (Y_f, f \in F) \) and \( Z(t) = (Z_f, f \in F) \) be the virtual queue vectors and let \( \Theta(t) = [Q(t), H(t), Y(t), Z(t)] \) be the combined queue vector, including the actual queues and the virtual queues. We can measure current NFV network congestion condition as follows:

\[
L(\Theta(t)) = \frac{1}{2} \sum_{n \in \Omega} \sum_{f \in F} \sum_{k \in [1, 2] \{f\}} Q_n^{f,k}(t)^2 + \frac{1}{2} \sum_{f \in F} H_f(t)^2 \nonumber + \sum_{f \in F} Y_f(t)^2 + \sum_{f \in F} Z_f(t)^2. \quad (15)
\]

When \( L(\Theta(t)) \) in (15) is small, all actual and virtual queue backlogs are small and the NFV network is stable. Yet, persistently and directly keeping \( L(\Theta(t)) \) small is computationally prohibitive. To address this issue, a time slot conditional Lyapunov drift can be further defined as:

\[
\Delta(\Theta(t)) = \mathbb{E}[L(\Theta(t + 1)) - L(\Theta(t))|\Theta(t)]. \quad (16)
\]

Then, the objective of problem (11) can transformed into a drift-minimizing bound minimization problem at each time slot \( t \) as

\[
\Delta(\Theta(t)) = \text{VE}(\sum_{f \in F} U_f(\gamma_f(t), \alpha_f)|\Theta(t)) \quad (17)
\]

with \( \text{VE} \geq 0 \) is a congestion control parameter to balance the stability and utility tradeoff. That is, a larger \( \text{VE} \) implies a higher utility but also results in higher network congestion at the same time. It is hard to directly minimize the drift-minus-utility. Fortunately, we notice that there always exists an upper bound on the drift as follows.

**Lemma 1.** Regardless of the control decision, drift \( \Delta(\Theta(t)) \) at any time slot \( t \) always satisfies

\[
\Delta(\Theta(t)) \leq B - \sum_{n \in \Lambda} \sum_{f \in F} \sum_{k \in [1, 2] \{f\}} Q_n^{f,k}(t) \mathbb{E}[(\mu_n^{\max} + \lambda_n^{\max,\text{out}})^2 + (\mu_n^{\max} + \lambda_n^{\max,\text{in}} + \theta_n^{\max})^2] \nonumber + \sum_{f \in F} H_f(t)^2 \nonumber + \sum_{f \in F} Y_f(t)^2 + \sum_{f \in F} Z_f(t)^2 \nonumber - \sum_{f \in F} Y_f(t) R_f(t) - r_f^{\min} \Theta(\Theta(t)) \nonumber \]

\[
- \sum_{f \in F} Z_f(t) r_f^{\max} - R_f(t) \Theta(\Theta(t)) \nonumber \]

where \( B \) is a finite constant depending on different network settings.

**Proof:** It is already known that \([a - b]^+ + c^2 \leq a^2 + b^2 + c^2 - 2a(b - c), \theta_n^{f,k} \leq \theta_n^{\max}, \mu_n^{f,k}(t) \leq \frac{C_n}{\theta_n^{f,k}} \leq \mu_n^{\max}, \lambda_n^{f,k}(t) \leq \lambda_n^{\max,\text{out}}, \lambda_n^{f,k}(t) \leq \lambda_n^{\max,\text{in}}, R_f(t) \leq A_f^{\max} \) and \( R_f(t), \gamma_f(t) \leq A_f^{\max} \). From (1), (12), (13) and (14), we have

\[
Q_n^{f,k}(t + 1)^2 \leq Q_n^{f,k}(t)^2 + (\mu_n^{\max} + \lambda_n^{\max,\text{out}})^2 \nonumber + (\theta_n^{\max} + \lambda_n^{\max,\text{in}} + \theta_n^{(k=1)\lambda_n^{\max}})^2 \nonumber - 2Q_n^{f,k}(t) (\mu_n^{f,k}(t) + \sum_{j \neq f} \lambda_n^{f,j}(t)) \nonumber - \theta_n^{f,k-1} \lambda_n^{f,k}(t) - \sum_{j \neq f} \lambda_n^{f,j}(t) \nonumber - \theta_n^{(k=1)\lambda_n^{f}}(t), \quad (19)
\]

\[
H_f(t + 1)^2 \leq H_f(t)^2 + 2A_f^{\max} + 2H_f(t) (R_f(t) - r_f^{\min}), \quad (20)
\]

\[
Y_f(t + 1)^2 \leq Y_f(t)^2 + (\lambda_f^{\max})^2 + (r_f^{\min})^2 - 2Y_f(t) (R_f(t) - r_f^{\min}), \quad (21)
\]

\[
Z_f(t + 1)^2 \leq Z_f(t)^2 + (\lambda_f^{\max})^2 + (r_f^{\max})^2 - 2Z_f(t) (r_f^{\max} - R_f(t)). \quad (22)
\]

Thus, we have

\[
\Delta(\Theta(t)) \leq \frac{1}{2} \sum_{n \in \Lambda, f \in F} \mathbb{E}[(\mu_n^{\max} + \lambda_n^{\max,\text{out}})^2 + (\mu_n^{\max} + \lambda_n^{\max,\text{in}} + \theta_n^{(k=1)\lambda_n^{\max}})^2] \nonumber + \frac{1}{2} \sum_{f \in F} \mathbb{E}[4 \sum_{n \in \Lambda} (\lambda_n^{\max})^2 + (r_f^{\min})^2 + (r_f^{\max})^2] \Theta(\Theta(t)) \nonumber \]

\[
- \sum_{f \in F} Q_n^{f,k}(t) \mathbb{E}[\mu_n^{f,k}(t) + \sum_{j \neq f} \lambda_n^{f,j}(t)] \nonumber - \theta_n^{f,k-1} \lambda_n^{f,k}(t) - \sum_{j \neq f} \lambda_n^{f,j}(t) - \theta_n^{(k=1)\lambda_n^{f}}(t), \quad (23)
\]

\[
- \sum_{f \in F} H_f(t) \mathbb{E}[R_f(t) - \gamma_f(t)] \Theta(\Theta(t)) \nonumber \]

\[
- \sum_{f \in F} Y_f(t) \mathbb{E}[R_f(t) - r_f^{\min}] \Theta(\Theta(t)) \nonumber \]

\[
- \sum_{f \in F} Z_f(t) \mathbb{E}[r_f^{max} - R_f(t)] \Theta(\Theta(t)) \nonumber \]

By defining \( B \) as \( \frac{1}{2} \sum_{n \in \Lambda, f \in F} (\mu_n^{\max} + \lambda_n^{\max,\text{out}})^2 + (\theta_n^{\max} + \lambda_n^{\max,\text{in}} + \theta_n^{(k=1)\lambda_n^{\max}})^2 + \frac{1}{2} \sum_{f \in F} 4 \sum_{n \in \Lambda} (\lambda_n^{\max})^2 + (r_f^{\min})^2 + (r_f^{\max})^2 \), we have (18).

Subtracting the utility function from both sides of (18) and combining similar terms, the drift-minus-utility is bounded by

\[
\Delta(\Theta(t)) = \text{VE}(\sum_{f \in F} U_f(\gamma_f(t), \alpha_f)|\Theta(t)) \nonumber \]

\[
\leq B + \sum_{f \in F} Y_f(t) r_f^{\min} - \sum_{f \in F} Z_f(t) r_f^{\max} - \Gamma \quad (24)
\]

where

\[
\Gamma = \sum_{f \in F} \mathbb{E}[VU_f(\gamma_f(t), \alpha_f) - H_f(t) \gamma_f(t) | \Theta(t)] \quad (25)
\]

\[
+ \sum_{f \in F} \mathbb{E}[(H_f(t) + Y_f(t) - Z_f(t)) R_f(t) - \sum_{n \in \Lambda} Q_n^{f,k}(t) R_f(t)] \Theta(\Theta(t)) \nonumber \]

\[
+ \sum_{n \in \Lambda, f \in F} Q_n^{f,k}(t) \mathbb{E}[\sum_{j \neq f} \lambda_n^{f,j}(t) - \sum_{j \neq f} \lambda_n^{f,j}(t)] \Theta(\Theta(t)) \nonumber \]

\[
+ \sum_{n \in \Lambda, f \in F} Q_n^{f,k}(t) \mathbb{E}[\mu_n^{f,k}(t) - \theta_n^{f,k-1} \lambda_n^{f,k-1}(t)] \Theta(\Theta(t)). \quad (25)
\]
As a result, we instead try to minimize the upper bound of the drift-minus-utility, i.e., the right side of (24), to achieve our goal of utility maximization. Such method has been proved as feasible and efficient in several online stochastic resource management studies, e.g., [8], [10], [27].

B. Online Distributed Algorithm

To achieve the goal of utility maximization is to equivalent to minimizing the drift-minus-utility bound in (24) by maximizing \( \Gamma \) in (25), from which we observe that \( \Gamma \) can be maximized by maximizing the four summation items individually. The four items actually refer to auxiliary variable derivation, admission control, load balancing and resource allocation, respectively. Following such philosophy, we can design an online algorithm to be implemented and distributively executed on each VNF for the above decisions as follows.

1) Auxiliary Variables Derivation: Since the auxiliary variables \( \gamma_f(t) \) are independent of the network flow, to maximize (25) is equivalent to

\[
\max : VU_f(\gamma_f(t), \alpha_f) - H_f(t)\gamma_f(t), \forall f \in F
\]  

with the constraint of \( 0 \leq \gamma_f(t) \leq A_f^{\max} \). As the utility function is differential, problem (26) can be solved by differentiating with \( \gamma_f(t) \). By such means, it is easy to derive that, when \( \alpha_f = 0 \),

\[
\gamma_f(t) = \begin{cases} A_f^{\max}, & \text{if } H_f(t) < V, \\ 0, & \text{otherwise.} \end{cases}
\]

In the case of \( \alpha_f \neq 0 \), we can make the following decisions:

\[
\gamma_f(t) = \begin{cases} A_f^{\max}, & \text{if } H_f(t) < \frac{V}{(A_f^{\max})^\alpha_f}, \\ \frac{V}{H_f(t)^\alpha_f}, & \text{otherwise.} \end{cases}
\]

2) Admission Control: The first VNF of each SFC shall be responsible for the admission control to avoid over-saturation of the whole chain, so as to make sure of the chain stability. Similar to the auxiliary variable derivation, we can maximize (25) by maximizing

\[
\sum_n (H_f(t) + Y_f(t) - Z_f(t) - Q_n^{f,1}(t)) R_n^f(t).
\]

We replace \( R_f(t) \) with \( \sum_n R_n^f(t) \) with the consideration of constraint (6) for each network flow \( f \in F \). The admission control decision \( R_n^f(t) \) can be made as

\[
R_n^f(t) = \begin{cases} A_n^f, & \text{if } Q_n^{f,-1}(t) < H_f(t) + Y_f(t) - Z_f(t), \\ 0, & \text{otherwise.} \end{cases}
\]

Obviously, the admission control decision is a queue length threshold-based strategy. When current queue length is shorter than the threshold, i.e., \( H_f(t) + Y_f(t) - Z_f(t) \), the newly arrived packets can be admitted without violating system stability. If the queue length exceeds the threshold, all packets will be blocked to guarantee the network stability.

3) Load Balancing: With the consideration of the dependency relationship in an SFC, admission control solely cannot guarantee the stability of the whole chain. Once a number of packets got allowed to enter the chain, all the VNFs to-be-visited thereafter shall cooperatively maintain the system stability. As a VNF may have multiple instances, the packet processing workloads can be balanced among them. The load balancing decision also shall be carefully made. We adopt classical backpressure routing algorithm [11] to guide the load balancing. According to the backpressure routing algorithm, (25) can be maximized under constraints (4) via setting the flow rates \( A_{ij}^{f,k}(t) \) to

\[
A_{ij}^{f,k}(t) = \begin{cases} L_{ij}, & \text{if } (f, k) = (f^*, t), Q_{i}^{f,k}(t) > Q_{j}^{f,k}(t), \\ 0, & \text{otherwise,} \end{cases}
\]

with

\[
(f^*, t) = \arg \max (|Q_{i}^{f,k}(t) - Q_{j}^{f,k}(t)|) \text{if } v(f, k) \in \mathbb{V}(n).
\]

Based on the preceding analysis, the queue length difference between neighbor servers is used to make the load balancing decision. From this point, the difference can be viewed as the network flow weight. The network flow with higher weight is offloaded with higher priority to the other servers hosting the same VNF.

4) Resource Allocation: Upon receiving certain packets from various flows, a VNF instance shall be allocated with corresponding resource to catch up with the desired packet processing rate. We allocate the resource on a server to the kth VNF for flow f to enable processing rate \( \mu_{n}^{f,k}(t) \). According to our analysis, maximizing (25) is equivalent to maximizing

\[
\sum_{f \in F} \sum_{k \in [1, l(f) - 1]} Q_n^{f,k}(t)(\mu_{n}^{f,k}(t) - \theta_{n}^{f,k-1} \mu_{n}^{f,k-1}(t))
\]

under constraints (5).

Notice that (29) can be rewritten into

\[
\sum_{f \in F} \sum_{k \in [1, l(f) - 1]} \mu_{n}^{f,k}(t)(Q_n^{f,k}(t) - \theta_{n}^{f,k} Q_n^{f,k+1}(t)).
\]

Denoting

\[
W_n^{f,k} = \left[ \frac{1}{\beta_{f,k}^{n}} (Q_n^{f,k}(t) - \theta_{n}^{f,k} Q_n^{f,k+1}(t)) \right]^{+}
\]

and

\[
(f^*(t), k^*(t)) = \arg \max (W_n^{f,k})
\]

when \( v(f, k) \in \mathbb{V}(n) \). The resource allocation \( \mu_{n}^{f,k}(t) \) can be derived as:

\[
\mu_{n}^{f,k}(t) = \begin{cases} C_n, & \text{if } (f, k) = (f^*(t), k^*(t)), W_n^{f,k} \neq 0, \\ 0, & \text{otherwise.} \end{cases}
\]

Therefore, upon resource allocation, both queue length and resource consumption of different VNF instances are considered as the flow weight. The flow with longer length and less strict resource requirement has a higher priority to be processed first.

5) Queue Updates: Combining all control decisions derived above, we can update the queue information, e.g., \( Q_n^{f,k}(t) \), \( H_f(t), Y_f(t) \) and \( Z_f(t) \), according to (1), (12), (13) and (14) for various control decisions in the next time slot. Such procedures proceed until the end of the SFC life.
It is observed that NUM-Offline is a convex problem with a concave utility function as the objective and all constraints are linear. As a result, it can be solved by using any method introduced in [12] or via any mathematical solver when the problem size is small. Here, we use the commercial mathematical optimization software MOSEK [13] to solve it. Different from the proposed online algorithm, NUM-Offline requires a priori knowledge of all flow rates over time slots \( t \leq T \). Instead of making the admission control, load balancing and resource allocation decisions every time slot, the offline decisions are made based on the average flow rate \( A_f = \frac{\sum_{t=1}^{T} A_f(t)}{T} \). Although the solution of NUM-Offline is impractical, in the next section, we will treat it as the benchmark to evaluate the efficiency of our algorithm.

V. PERFORMANCE EVALUATION

The performance of our online distributed algorithm is evaluated through both extensive simulations and testbed experiments under different network status. We report our performance evaluation results in this section.

A. Simulation-based Performance Evaluation

1) Simulation Settings: We simulate a network based on the topology of Cogent’s network with 43 connected servers\(^1\). All the links between any two servers are set with the same bandwidth capacity of 300. There are a number of different types of SFCs consisting of different VNFs hosted on different servers. A VNF may have a number of instances on different servers. Two different service chains may share the same type of VNF. We generate a number of flows (25 in default) requesting different service chains with average arrival rate \( a_i = 100 \) and the maximum packet arrival rate \( A^{\text{max}}_i = 2a_i \). The number of newly arrived packets at each time slot \( t \) is uniformly and randomly distributed within the range of \([0, A^{\text{max}}_i]\). In default, we set the scaling factor \( \theta = 1 \), indicating that the flow rate does not change after passing through a VNF. For each group of experiment, we run the simulation for 100,000 time slots to get the average throughput of each flow, based on which the fairness index value is calculated. The queue backlog on each VNF is also analyzed. We use the well known Jain’s index to evaluate the fairness of flows. We default set \( \alpha = 1 \) and \( V = 10,000 \). To verify the optimality of our online scheduling algorithm (Online), we also derive the offline optimal network utility by solving NUM-Offline via MOSEK. The optimal network utility under offline settings is computed using time average arrival rates as flow rate.

2) Simulation based Evaluation Results: Firstly, we investigate how \( V \) affects the performance of different network flows as well as the whole system. The performance evaluation results are reported in Fig. 3. Figs. 3(a) and 3(b) plot the time averaged utility and queue backlog under different values of \( V \), when \( \alpha = 0 \) and 1, respectively. We can observe that the time averaged utility first increases significantly with the increasing of \( V \), and then gradually converges to the offline optimum when the value of \( V \) is large. This quantitatively

\(^1\)http://cogentco.com/en/network/network-map
verifies the correctness of (32) that our algorithm can approach the optimal profit with a diminishing gap $1/V$. However, with the increasing of $V$, the time averaged queue backlogs also increase, implying more severe network congestion. This is because we allow more packets to be admitted into the network under a larger value of $V$. Figs. 3(a) and 3(b) reflect the tradeoff between network stability and utility maximization discussed in Section IV-C. Furthermore, a larger $V$ is needed for a larger $\alpha$, so as to achieve close-to-optimum utility.

Next, we investigate the tradeoff between throughput and fairness under different values of $\alpha$, as shown in Fig. 4. When $\alpha = 0$, we can achieve the maximum throughput, albeit with a bad fairness in return. This is because, when $\alpha = 0$, we bias on the network throughput and totally ignore the fairness among flows, i.e., admitting more packets which consume less resources and blocking resource-intensive flow. If we set a larger value of $\alpha$, fairness will get better, while the overall network throughput will degrade. When the value of $\alpha$ is large enough, e.g., $\alpha = 3$, we can see that the fairness among flows is close to 1, indicating that each flow has a similar throughput when converged.

We investigate the effect of resource competition among different flows by specially considering a small scale network where there are only 4 flows. The results are shown in Figs. 5 and 6, from which we notice that the average throughput of each flow increases with $V$ firstly when $V$ is small. However, when the value of $V$ becomes larger than a certain value, the throughputs of both flow 2 and flow 4 begin to decrease, may even approach 0 (e.g., flow 4) when $V$ is very large. The reason behind this phenomenon is that we set flow 2 and flow 4 as computation-intensive. In this case, our algorithm targeting at throughput maximization may sacrifice them so as to allow more packets from the other flows requiring less resources. Obviously, this is not acceptable and we shall take the fairness into consideration. The results are plotted in Fig. 6, from which we can see that flow 2 and flow 4 can achieve throughput higher than 30, while flow 1 and flow 3 cannot get throughput higher than 60, no matter what the value of $V$ is. Therefore, better fairness among flows can be achieved after the introduction of throughput requirement constraints at the cost of degraded overall network throughput. In other words, the flow throughput requirements have a significant influence on the fairness, and our algorithm can well address the tradeoff between fairness and throughput.

Next, we enlarge the network scales by increasing the number of flows from 2 to 200 to check how our algorithm performs in different network scales. Specially, we enforce a portion of flows to share the same VNFs to investigate the competition among flows. We also compare our $\alpha$-based approach ($\alpha = 0$ and 1, respectively) with the hard maxmin one described in [26]. The performance evaluation results are shown in Fig. 7. When $\alpha = 0$, with the increasing of flow number, the throughput first increases. It is because more flows brings more newly arrival packets. With good online scheduling, we can allocate the resource efficiently and achieve the goal of larger throughput with a relatively low fairness (0.84). However, when the number of flows surpasses 15, the total throughput start to decrease as more flows also imply more resource requirements. With a limited resource capacity, the throughput gradually decreases. On the other hand, when $\alpha = 1$, the fairness almost keeps the same (above 0.95) while the throughput shows the same trend as $\alpha = 0$. The hard maxmin approach can achieve a perfectly fair result (1.0 all the time), but sacrifices the total throughput in return. This further verifies the efficiency of our algorithm. One interesting finding
is that the value of fairness finally converges with ($\alpha = 1$) or without fairness control ($\alpha = 0$). The reason is that, when the number of flows is large, the randomly generated workload, network function locations and resource requirement of each flow is naturally well balanced with a good fairness even without scheduling. Hence, the fairness keeps stable as 1.0 after the number of flows reaches 50.

Finally, we evaluate how different values of traffic scaling factor $\theta$ affects the average throughput. Fig. 8 plots the results under different flow arrival rates when $\alpha = 0$. It can be observed that the total network utility shows as a linearly increasing function of the arrival rate when the value of $\theta$ is small, regardless the values of $V$. Such phenomenon is due to that all the flows can be admitted into the network when the traffic cannot saturate the network. However, with the increasing of $\theta$, the network resource capacity fail to satisfy the total resource requirements. In this case, our admission control can prevent excessive packets coming into the network and the network is still stable. Therefore, we can see that the average throughput finally converges as it is constrained by the network capacity.

B. Testbed based Performance Evaluation

Besides simulation based studies, we implement a testbed to verify the efficiency of our algorithm practically.

1) Testbed Implementation: Fig. 9 shows the architecture of our testbed system. Each network function can be viewed as one queue with limited queuing buffer. The queue manager is responsible for monitoring the queue backlog status and manage the queueing packets of each flow requesting for certain service chains. When a new packet arrives, the rx thread reads and classifies it to a specific flow based on the five-tuple information in packet headers and enqueues the packet to a queue. For the packets processed by all...
required network functions, the $tx$ thread uses a round-robin algorithm to dequeue them from the queues and send them out. Revisiting our computation control decision, at each time slot, the schedule thread runs our computation control decision and choose one queue to serve based on the ratio of the queue backlog difference and resource cost. For the other queues, we need to calculate the number of packets to be processed based on the knowledge of available resources. Then, the queue manager updates the packets waiting in each queue accordingly. In practice, we cannot obtain the exact resource amount. We simplify the algorithm and just let all cores to serve the selected queue. All the threads process packets from the selected queue with the network function corresponding to that queue. We also implement a flow generator which can generate flows with different rates. We implement the architecture based on open-source DPDK framework\textsuperscript{2} and bind each thread on one core to avoid context switch.

We implement basic Run-to-Completion (RTC) and Pipeline execution modes to compare with our fairness-aware scheduling. The RTC mode \cite{29} runs all network functions in one thread. While the Pipeline mode \cite{30} runs each network function in one thread and uses intermediate queues to store packets for communication with the other threads. We set up four kinds of network functions $n f \ 1$, $n f \ 2$, $n f \ 3$ and $n f \ 4$, which cost $10K$, $20K$, $10K$ and $5K$ CPU cycles per packet, respectively. In RTC mode and our mode, we use three threads to process packets. In the Pipeline mode, we use three threads to run three network functions, respectively. We set up three flows corresponding to service chains $n f \ 1 \rightarrow n f \ 2 \rightarrow n f \ 3$, $n f \ 2 \rightarrow n f \ 3$, and $n f \ 2 \rightarrow n f \ 4$, respectively. The arrival rates are set as $1Gbps$, $1.5Gbps$ and $0.5Gbps$, respectively, with identical packet length. We measure the traffic sent out as the flow throughput. All the experiments are carried out on a server equipped with two 2.6GHz 48-Core Intel Xeon CPU E5-2670 2.60Ghz processors.

Two groups of experiments are conducted. In the first group of experiments, we exclude the fairness issue as both RTC and Pipeline do not consider fairness. We use this group to evaluate how our Online algorithm performs on the throughput in comparison with the two competitors. Next, in the second group of experiments, we take the fairness issue into consideration and evaluate how our Online algorithm reacts to various fairness requirements, and how the throughput is affected by the fairness.

\textsuperscript{2}https://dpdk.org/
2) Testbed based Evaluation Results: Firstly, we consider a throughput maximization scenario without fairness consideration, i.e., $\alpha = 0$, by varying the number of flows from 3 to 15. Fig. 10 shows the result of our experiments. We specially detail the throughput of all 3 flows when there are 3 flows in Fig. 10(a) and give the overall throughput in Fig. 10(b). We notice that the Pipeline mode performs the worst among these three modes. This is because, in the Pipeline mode, the resources (cores) are fairly allocated to all four network functions, regardless of the workloads. The network functions with heavy workloads may become the bottleneck and limit the throughput of the related flows. On the other hand, RTC mode shares the resource among network functions and each core can be fully utilized. However, RTC mode processes each flow without considering the packet processing cost, i.e., packet processing cycle. The output flow rate is strictly related to the arrival rate, i.e., a larger arrival rate imposing a larger output rate. It can be observed from Fig. 10(a) that the output rates of flow 1, flow 2 and flow 3 are highly related with their arrival rates, i.e., 1Gbps, 1.5Gbps and 0.5Gbps. Note that flow 1 is with a high processing cost of 40K cycles per packet and requires more resources to produce the same output rate, leading to sub-optimal network throughput. Our algorithm properly tackles these problems by maximizing the total throughput with joint consideration of rate control and resource utility, achieving close-to-optimum network throughput. In this experiment, we can see from Fig. 10 that flow 2 and flow 3 get fully served, considering of their low-cost network functions, while most packets of flow 1 are blocked due to its high-processing-cost. By allocating more resources to the low-cost flows, the total throughput of our algorithm is significantly increased, as shown in both Fig. 10(a) and Fig. 10(b).

Although we aim at a maximal throughput, we cannot totally ignore the fairness. It is not acceptable to absolutely reject flow 1 as in the first group of experiments. Next, we take the fairness into consideration and study the tradeoff between the throughput and fairness by varying the value of $\alpha$ from 0 to 1.8. The results are plotted in Fig. 11. Similarly, we detail the throughput in Fig. 11(a) and give the overall throughput in Fig. 11(b). We can see that the largest throughput can be achieved with the worst fairness when $\alpha = 0$, i.e., the case in Fig. 10. The throughputs of flow 2 and flow 3 are close to their arrival rates, while the throughput of flow 1 is only around 1% of its arrival rate due to its high processing cost. As the value of $\alpha$ increases, the total throughput decreases gradually while the fairness gets better, consistent with our simulation experiment results. With the increasing of $\alpha$, the throughput of flow 1 increases while the throughputs of flow 2 and flow 3 decrease. In order to achieve better fairness, more resources are allocated to flow 1 to improve its throughput. When the value of $\alpha$ is large enough, the throughputs of the three flows are the same as the one under RTC mode and the throughputs are approximately proportional of the flow arrival rates, achieving the best fairness. Such phenomenon can be also observed in Fig. 11(b) where the fairness finally converge when $\alpha \geq 1.2$.

VI. RELATED WORK

A. NFV Optimization

The promising advantages of NFV have attracted much interest from both academia and industry. Much effort has been made to improve its performance by exploring the scalability and flexibility of NFV [20]. Recent studies mainly focus on the VNF deployment with the consideration of QoS [5], [6]. For example, Fei et al. [21] aim at minimizing the maximum load to achieve the load balancing of both computation and bandwidth resources and Zhang et al. [22] jointly consider the resource cost and link delay cost in NFV service chain deployment. Wang et al. [19] further investigate the dynamic VNF placement problem and design an $O(1)$-competitive algorithm to minimize the total cost. Most existing algorithms are centralized and highly rely on network traffic statistics, or even require a global network knowledge. As a result, they cannot be applied into large-scale networks due to the lack of scalability and flexibility, and also fail to cope with the dynamically changing traffic demands. To address this issue, dynamic and de-centralized algorithms to schedule network resource allocation and to minimize the network cost are proposed, e.g., in [7]. However, the resource capacity and fairness among network flows are not considered. In particular, we focus on balancing the tradeoff between flow fairness and network performance. To our best knowledge, this is the first study to address this tradeoff in the literature.

B. Network Fairness Optimization

Network fairness optimization problems, aiming at achieving fairness among different flows, has been widely investigated in traditional wireless and wireline networks [23]. Kavitha et al. [24] propose a robust $\alpha$-fair scheduling algorithm derived from game theory in presence of non-cooperation in cellular networks. Schwarz et al. [25] propose an adjustable fairness scheduler based on $\alpha$-fair utility function, where Jain’s fairness index is adopted. Ronasi et al. [26] further use hard min-max fairness instead of $\alpha$-fair utility function to provide fairness between multiple flows, while maximizing the minimum throughput of flows at the same time. To optimize the network fairness at runtime, Neely et al. [27] use Lyapunov optimization technology to dynamically solve network utility maximization problem for achieving fairness among different flows in a heterogeneous network. The fairness optimization methods proposed for traditional wireless or wireline networks cannot be directly applied in NFV based networks, due to the ignorance of resource sharing of VNF instances and resource capacity constraints.

VII. CONCLUSION

In this paper, we investigate a network utility maximization problem, with the consideration of throughput and fairness tradeoff among multiple network flows in NFV service chain. To balance the tradeoff, we define a network utility function with a fairness bias, and adopt a discrete time-slotted queuing model to describe the online flow scheduling and rate control problem. Specially, we take into the consideration of SFC
semantics, queue stability, network resource provision, and requirement conditions. Based on the formulation, we propose an online dynamic distributed algorithm by taking advantages of Lyapunov optimization techniques to tackle the network dynamics. We provide theoretical analysis to show that our algorithm can achieve close-to-optimum average throughput arbitrarily. To validate the correctness of our analysis and the high efficiency of our algorithm, extensive simulation-based and testbed-based experiment results are provided to show that our algorithm achieves performance close to the optimal solution, with fairness consideration.

REFERENCES