

Centralized Network Utility Maximization with Accelerated Gradient Method

Ying Tian^{*}, Zhiliang Wang^{†‡}, Xia Yin^{*‡}, Xingang Shi[†], Jiahai Yang^{†‡}, and Han Zhang[†]

^{*}Department of Computer Science and Technology, BNRist, Tsinghua University, Beijing, China

[†]Institute for Network Sciences and Cyberspace, BNRist, Tsinghua University, Beijing, China

[‡]Zhongguancun Lab, Beijing, China

E-mails: y-tian18@mails.tsinghua.edu.cn, {wzl, shixg, yang}@cernet.edu.cn, {yxia, zhhan}@tsinghua.edu.cn

Abstract—Network utility maximization (NUM) is a well-studied problem for network traffic management and resource allocation. Because of the inherent decentralization and complexity of networks, most researches develop decentralized NUM algorithms. In recent years, the Software Defined Networking (SDN) architecture has been widely used, especially in cloud networks and inter-datacenter networks managed by large enterprises, promoting the design of centralized NUM algorithms. To cope with the large and increasing number of flows in such SDN networks, existing researches about centralized NUM focus on the scalability of the algorithm with respect to the number of flows, however the efficiency is ignored. In this paper, we focus on the SDN scenario, and derive a centralized, efficient and scalable algorithm for the NUM problem. By the designing of a smooth utility function and a smooth penalty function, we formulate the NUM problem with a smooth objective function, which enables the use of Nesterov's accelerated gradient method. We prove that the proposed method has $O(d/t^2)$ convergence rate, which is the fastest with respect to the number of iterations t , and our method is scalable with respect to the number of flows d in the network. Experiments show that our method obtains accurate solutions with less iterations, and achieves close-to-optimal network utility.

I. INTRODUCTION

As the number of network users and network traffic grow explosively, fair and efficient allocation of network resources becomes increasingly important. Regarding network resource allocation, a canonical and widely-studied problem is the *network utility maximization* (NUM) problem proposed by Kelly [1]. In the NUM problem, there are sources sending elastic traffic along predetermined paths. Each source has a continuously differentiable, non-decreasing and concave utility function related to its sending rate, e.g., the most commonly used α -fair function proposed by Mo and Walrand [2], [3]. The objective is to maximize the sum of utility functions subject to link capacity constraints. Because of the great generality and flexibility of the NUM problem, it has been combined with various network protocols and applied to different network scenarios like wireless network [4], multi-layer radio access network [5], network function virtualization [6] and so on. Due to the inherent decentralization and complexity of networks, most of the NUM researches focus on developing decentralized algorithms [7]–[9] based on dual decomposition [10]. They assume that information like link capacity and flow route are only locally known by network devices [11], and

the problem is iteratively solved in a decentralized manner, where network devices locally solve their subproblems and communicate about necessary information like intermediate results. Decentralized methods usually scale with the number of concurrent flows d , and the fastest convergence rate achieved by existing decentralized methods is $O(1/t)$ [12], [13] where t is the number of iterations.

In recent years, Software Defined Networking (SDN) [14] draws great attention because of its ability to decouple control plane from data plane. Unlike traditional decentralized networks, in the SDN architecture there is an SDN controller has a global view about network topology and network states, thus it can centrally computes forwarding paths, flow rates, etc., and then controls the behavior of switches in the data plane. The SDN architecture has been widely used, especially in cloud networks and inter-datacenter networks managed by large enterprises [15], [16], where a large number of flows are competing for scarce and expensive network resources like link bandwidth. To serve the urgent need of managing and optimizing resource allocation in such networks, it is essential to develop *centralized*, *efficient* and *scalable* algorithms for the NUM problem: the algorithm should be *centralized* to solve the problem on the SDN controller with global information, be *efficient* to achieve near-optimal results fast or even in real time, and be *scalable* to accommodate the large and even increasing number of network flows.

Existing researches about centralized NUM algorithm mainly focus on the scalability of the algorithm, but the efficiency is not ideal. Gupta et al. [17] achieve scalability by converting the original NUM problem into a simpler problem involving only aggregate flows, without guaranteeing the convergence rate and efficiency of the algorithm. The state-of-the-art work Exp-NUM [18] solves the penalty-based NUM formulation with exponentiated gradient descent. The algorithm is proven to be scalable with respect to the number of flows. But it shows poor efficiency because it converges quite slow with respect to the number of iterations t , and needs a lot of iterations to obtain an accurate solution.

In this paper, we follow the classical formulation of the NUM problem with d concurrent flows and the α -fair utility function, and derive a centralized *accelerated gradient method* (AGM) for the NUM problem, which is both efficient and scalable. While most researches only exploit the Lipschitz continuous property of the α -fair function, we go a step further

Corresponding authors are Zhiliang Wang and Xia Yin.

by revealing the smoothness of the α -fair function and design an appropriate smooth penalty function for the link capacity constraints, enabling the use of Nesterov's accelerated gradient method [19], which is believed to be the fastest iterative algorithm for smooth objective function. Our method is proven to have the fastest convergence rate with respect to the number of algorithm iterations t which shows great efficiency, and is scalable with respect to the number of flows d , showing its great potential for resource allocation in large-scale centralized controlled networks.

The main contributions of our paper are as follows:

- We are the first to formulate the NUM problem with a smooth objective function. First, we propose and prove that the α -fair utility function becomes smooth with only minor changes. Then, we design a smooth penalty function for the penalty-based formulation, which leads to the overall smoothness of the objective function in cooperation with the smooth α -fair utility function or any other smooth utility functions. The NUM formulation with smooth objective function benefits the use of existing algorithms, and will encourage the design of other fast algorithms.
- We propose to apply Nesterov's accelerated gradient method (AGM) to solve the centralized NUM problem. Thanks to the smoothness of the objective function, the algorithm has $O(d/t^2)$ convergence rate under the assumption that the number of network links is much smaller than d , which is the fastest with respect to t compared with all existing decentralized and centralized methods. The iteration complexity of our algorithm is proportional to $\sqrt{d}$, so the algorithm is scalable with respect to the number of flows and can maintain good performance as d increases. Besides, AGM is guaranteed to achieve more accurate solutions than the state-of-the-art method Exp-NUM with enough number of iterations. We also use two restart heuristics to boost the convergence of AGM and alleviate the bumps during the iteration.
- We conduct thorough experiments to evaluate the algorithms and demonstrate the efficiency and scalability of AGM. The experimental results are in good agreement with the theoretical results. AGM has the fastest convergence rate and achieves accurate solutions. Besides, our method obtains close-to-optimal utility function value, indicating the effectiveness of the smoothly penalized NUM formulation. And our method performs consistently well with different number of flows, showing its good scalability with respect to d .

The rest of the paper is structured as follows. Section II is the related work. Section III presents the formulation of the NUM problem. In Section IV, we introduce the state-of-the-art gradient based methods for centralized NUM problem. We formulated the NUM problem with smooth objective function in Sec V, and then present the accelerated gradient method in Sec VI. In section VII, we evaluate the proposed algorithm. In section VIII, we discuss the limitations of our method. Finally,

we make a conclusion in Section IX.

II. RELATED WORK

In the past two decades, there have been plenty of research in NUM. Most of them study concave utility functions, and some specifically use α -fair function in the problem formulation or as a representative example [5], [7], [8], [12], [13], [17], [18], [22]. NUM algorithms are usually iterative, whose performance is related to: i) *convergence rate*, i.e., the achieved error ε^1 at step t , or equivalently, *iteration complexity*, i.e., the number of iterations required to achieve an error; ii) *per-iteration cost*, measured by the computational complexity or number of mathematical operations per iteration.

Newton's method is believed to have the fastest convergence rate. Wei et al. [23] proposed a distributed Newton's method for the NUM problem. Zargham et al. [24], [25] accelerated dual decent with an approximate Newton-like algorithm. However, the convergence rate is not concisely derived. Besides, Newton's method requires the computation of Hessian matrix and its inverse, causing heavy per-iteration cost.

Therefore, first-order methods that only require the computation of the first derivative (gradient) are preferred. Kelly et al. [7] decomposed the NUM problem into a user subproblem and a network subproblem and solve the problem with gradient descent. Low et al. [8] proposed another decomposition framework and consider asynchronous situation. Both works leverage the dual-based gradient descent algorithm, but concise proof of the convergence rate is not given. In [20], authors proved that the dual gradient method with primal averaging has $O(1/\sqrt{t})$ convergence rate. In [12], authors assumed that constraint functions are Lipschitz continuous and convex, and proposed a virtual queue algorithm based on dual gradient method, achieving $O(1/t)$ convergence rate. Assuming strong concavity of the utility function, Beck et al. [13] solved the dual problem with accelerated gradient method. But their primal-dual method only achieves $O(1/t)$ convergence rate for the primal solution.

The works mentioned above all develop distributed algorithms for decentralized controlled networks through dual decomposition. As the network architecture evolves, NUM algorithms considering other network architectures emerge. Karakoç et al. [5] proposed a multi-layer decomposition for NUM problem in the radio access network solved by dual gradient descent method. Allybokus et al. [22] considered an SDN network with multiple distributed controllers, and developed a distributed algorithm based on alternating direction method of multipliers (ADMM) algorithm.

For the NUM problem in centralized SDN networks, a classic method is projected gradient ascent [21] (or or projected gradient descent for minimizing problems). For Lipschitz objective functions, it has $O(\sqrt{d}/\sqrt{t})$ convergence rate and $\Omega(d/\varepsilon^2)$ iteration complexity. For smooth objective functions, its performance guarantee is improved to $O(d/t)$ convergence rate and $\Omega(d/\varepsilon)$ iteration complexity. Because the iteration

¹The error ε is measured by the gap between the objective value of the current solution and the optimal value.

TABLE I
NETWORK TYPE, CONVERGENCE RATE, ITERATION COMPLEXITY, AND OBJECTIVE FUNCTION CLASS OF THE FIRST-ORDER METHODS

Network Type	Method	Convergence Rate	Iteration Complexity	Objective Function Class ^a
Decentralized controlled	Dual gradient [20]	$O(1/\sqrt{t})$	$\Omega(1/\varepsilon^2)$	concave
	Virtual queue [12]	$O(1/t)$	$\Omega(1/\varepsilon)$	concave (and Lipschitz continuous constraint)
	Primal-dual [13]	$O(1/t)$	$\Omega(1/\varepsilon)$	strongly concave
SDN network	Projected gradient ascent ^b [21]	$O(\sqrt{d}/\sqrt{t})$	$\Omega(d/\varepsilon^2)$	concave, Lipschitz continuous
		$O(d/t)$	$\Omega(d/\varepsilon)$	concave, smooth
	Exp-NUM [18]	$O(\sqrt{\ln d}/\sqrt{t})$	$\Omega(\ln d/\varepsilon^2)$	concave, Lipschitz continuous
	AGM (this paper)	$O(d/t^2)$	$\Omega(\sqrt{d}/\sqrt{\varepsilon})$	concave, smooth

^a In this table, we assume that the NUM problem is a maximizing problem so all objective functions should be concave.

^b Projected gradient ascent has different convergence rate and iteration complexity for Lipschitz continuous and smooth objective functions.

complexity of projected gradient ascent is proportional to d , it is not scalable with respect to the number of flows. To cope with the increasing number of flows, Gupta et al. [17] proposed a NUM decomposition in which the set of flows are grouped by source-destination pairs, and solved the problem with ADMM algorithm. But the convergence rate is not studied. Exp-NUM [18] is most relevant to our paper. Exp-NUM uses a penalty-based formulation of the NUM problem and additionally adds a global flow rate bound, which allows the problem to be solved with exponentiated gradient descent, i.e., mirror descent [26]. The advantage of Exp-NUM is that the iteration complexity is $\Omega(\ln d/\varepsilon^2)$, thus the algorithm is scalable with respect to the number of flows d . However, the convergence rate of Exp-NUM is $O(\sqrt{\ln d}/\sqrt{t})$, implying that it cannot converge to accurate solutions efficiently.

The convergence rate of our method AGM is $O(d/t^2)$, which is the fastest w.r.t. t compared with all decentralized and centralized methods mentioned above ². Besides, the iteration complexity is $\Omega(\sqrt{d}/\sqrt{\varepsilon})$ in the worst case, i.e., the number of iterations required for an ε -optimal solution is proportional to the square root of the number of flows. Thus, our method is both efficient and scalable. In Table I, we summarized the network type, convergence rate, iteration complexity and objective function class of the first-order methods.

III. NETWORK UTILITY MAXIMIZATION PROBLEM

In this section, we present the formulation of the NUM problem and introduce the commonly used utility function and penalty function. The notations used in this paper is summarized in Table II.

A. Problem Formulation

The system model of the NUM problem consists of the following aspects:

1) *Network*: The network is represented by a graph $G = (V, E)$, where V is the router set and E is the unidirectional link set. Each link $e \in E$ has a capacity c_e .

2) *Flows*: Let S represents the set of elastic flows whose sending rate can be controlled by the source or controller. $s \in S$ represents a specific flow, and x_s is the rate of s . We also use $\mathbf{x} = (x_s)_{s \in S}$ as the column vector of flow rates. Besides, let d be the number of flows in the network, i.e., $d = |S|$, $\mathbf{x} \in \mathbb{R}_+^d$.

²We believe AGM converges faster than Gupta et al. [17] in terms of t because $O(1/t^2)$ is recognized as the fastest convergence rate of all known first-order methods.

TABLE II
NOTATIONS

Symbol	Description
$G = (V, E)$	The network graph, where V is the node set and E is the unidirectional link set.
c_e	The capacity of link e .
S	The set of traffic flows.
d	The number of flows, i.e., $d = S $.
$\mathbf{x}$	The vector of flow rates where x_s is the rate of flow s .
$\mathbf{x}^{(t)}$	The obtained solution at iteration t .
$\mathbf{x}^*$	The optimal solution.
$\mathbf{A}$	The routing matrix where $\mathbf{A}_{e,s}$ represents whether flow s passes link e or not.
U_s, U	The flow utility function and the network utility function, where $U = \sum_{s \in S} U_s$.
P_e, P	The link penalty function and the network penalty function, where $P = \sum_{e \in E} P_e$.
V	The objective function.
V^*	The optimal objective function value.
$\mathbb{R}_+^d$	The space of d -dimensional nonnegative column vector.
$[\cdot]_+$	The projection on $\mathbb{R}_+^d$.
$\ \cdot\ $	The Euclidean norm.
$\odot$	The Hadamard product.

3) *Routing path*: The routing path is predetermined for each flow $s \in S$. We assume that there is one path for each flow and the $|E| \times |S|$ routing matrix $\mathbf{A}$ is defined as:

$$\mathbf{A}_{e,s} = \begin{cases} 1, & \text{if flow } s \text{ passes through link } e \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

This definition is suitable for simple paths in which a flow s fully passes each link. For paths in which a flow may split over links, e.g., Equal-cost multi-path (ECMP) or Segment Routing [28] paths, $\mathbf{A}_{e,s}$ can be defined as the fraction of flow s passes through link e and relaxed to $[0, 1]$. For convenience, we use the row vector $\mathbf{A}_e$ to denote the e -th row of matrix $\mathbf{A}$.

4) *Utility function*: Each flow s is associated with a *flow utility function* $U_s : \mathbb{R}_+ \rightarrow \mathbb{R}$, i.e., each flow s gains a utility $U_s(x_s)$ when it sends data at rate x_s . We call $U(\mathbf{x}) = \sum_{s \in S} U_s(x_s)$ the *network utility function*.

The NUM problem is formulated as:

$$\begin{aligned} \max \quad & U(\mathbf{x}) = \sum_{s \in S} U_s(x_s) \\ \text{s.t.,} \quad & \mathbf{A}_e \mathbf{x} \leq c_e, \forall e \in E \\ & x_s \geq 0, \forall s \in S \end{aligned} \quad (\text{NUM})$$

The objective is to maximize the network utility function. It is constrained that the traffic carried by a link should not exceed its capacity, and the flow rate should be nonnegative.

Usually, the utility function is defined by service provider in order to achieve a certain type of resource allocation. The α -fair function [3] is a well-known and most commonly used flow utility function, defined as Eq. (2). The α -fair function is flexible to capture different optimization goals, and different values of the parameter α yields different fairness criteria. For example, $\alpha = 0$ leads to the classic throughput maximization, $\alpha = 1$ yields proportional fairness [1], $\alpha = 2$ yields to potential delay fairness [29] and $\alpha \rightarrow \infty$ gives min-max fairness.

$$U_\alpha(x_s) = \begin{cases} \frac{x_s^{1-\alpha}}{1-\alpha}, & \alpha \geq 0, \alpha \neq 1 \\ \ln x_s, & \alpha = 1 \end{cases} \quad (2)$$

For the sake of our proposed method AGM and its associated theorems and proofs, we make the following assumptions:

Assumption 1. The network topology is sparse enough so that $|E| \ll |V|(|V| - 1)$. This assumption holds when network topologies are not full-mesh and are much sparser (e.g., the topologies in Topology Zoo dataset [27], or GÉANT and RF1221 used in VII Evaluation).

Assumption 2. The number of flows d is at least the same order of magnitude as the number of node pairs $|V|(|V| - 1)$. d could be greater than $|V|(|V| - 1)$ because we consider fine-grained flows sent by network users rather than aggregate flows between source-destination pairs.

Assumption 3. The utility function is concave and smooth. The α -fair function is well-known to be concave, and we will reveal its smoothness later in the paper.

B. Penalty-based Formulation

In problem (NUM), the link capacity constraints form a polyhedron and is hard to handle. For gradient based methods, projection onto the polyhedron can be time consuming. One way to deal with it is to derive the Lagrange dual of (NUM), which is suitable for decentralized algorithms. Another conventional way is to relax the problem by using a penalty-based formulation [11], and the objective function becomes:

$$V(\mathbf{x}) = \sum_{s \in S} U_s(x_s) - \mu \sum_{e \in E} P_e(\mathbf{A}_e \mathbf{x})$$

The function $P_e : \mathbb{R}_+ \rightarrow \mathbb{R}$ is a *link penalty function* that penalizes the arriving rate for overshooting the link capacity. We call $P(\mathbf{x}) = \sum_{e \in E} P_e(\mathbf{A}_e \mathbf{x})$ the *network penalty function*. $\mu > 0$ is a penalty parameter that balances the utility function and penalty function. To accommodate the convention of using gradient descent (instead of ascent) based algorithms for minimization (instead of maximization) problems, we negate the function $V(\mathbf{x})$, and then the problem is formulated as:

$$\begin{aligned} \min \quad & V_P(\mathbf{x}) = - \sum_{s \in S} U_s(x_s) + \mu \sum_{e \in E} P_e(\mathbf{A}_e \mathbf{x}) \\ \text{s.t.,} \quad & x_s \geq 0, \forall s \in S \end{aligned} \quad (\text{NUM-P})$$

The optimization objective now is to minimize the negative of the network utility functions plus the network penalty function. The advantage of the problem (NUM-P) is that

Algorithm 1 Projected Gradient Descent (PGD)

Input: initial point $\mathbf{x}^{(0)}$, step size γ , number of iterations T
Output: final solution $\hat{\mathbf{x}}^{(T)}$

- 1: **for** $t \leftarrow 0$ **to** $T - 1$ **do**
- 2: $\mathbf{x}^{(t+1)} \leftarrow [\mathbf{x}^{(t)} - \gamma \nabla V_P(\mathbf{x}^{(t)})]_+$
- 3: **end for**
- 4: **return** $\hat{\mathbf{x}}^{(T)} \leftarrow \frac{1}{T} \sum_{t=1}^T \mathbf{x}^{(t)}$

the link capacity constraints are removed so that the feasible region now becomes the nonnegative orthant $\mathbb{R}_+^d$ instead of a polyhedron, making the problem easier to deal with. Besides, by using an appropriate penalty function and parameter μ , the exact value of the optimal solution of (NUM) can be obtained through solving (NUM-P) [11], [18].

A typical choice for the link penalty function is to use the exact penalty function [30] $P_{e,+}(\mathbf{A}_e \mathbf{x})$ as shown in Eq. (3). The penalty value is equal to how much the link capacity is exceeded. However, this function is not continuously differentiable when $\mathbf{A}_e \mathbf{x} = c_e$, which may cause numerical issues and make the problem hard to handle.

$$P_{e,+}(\mathbf{A}_e \mathbf{x}) = \max\{\mathbf{A}_e \mathbf{x} - c_e, 0\} \quad (3)$$

IV. GRADIENT METHODS FOR NUM WITH THE EXACT PENALTY FUNCTION

In this section, we introduce the classic projected gradient descent (PGD) [21] and the state-of-the-art method Exp-NUM [18] for NUM problem with the α -fair utility function U_α (Eq. (2)) and exact penalty function $P_{e,+}$ (Eq. (3)). We describe the prerequisites, algorithm procedures, performance guarantees of these methods, and analyze their drawbacks.

The pseudocode of PGD is shown in Algorithm 1. At each iteration, PGD takes a step in the opposite direction of the gradient, and then projects the obtained point back onto $\mathbb{R}_+^d$. The final solution $\hat{\mathbf{x}}^{(T)}$ is the average over all intermediate points in the iteration. To give the convergence rate of PGD, an assumption is made:

Assumption 4. The partial derivatives of $V_P(\mathbf{x})$ are bounded, i.e., there exists $M \geq 0$ so that

$$\left| \frac{\partial V_P(\mathbf{x})}{\partial x_s} \right| \leq M, \quad \forall \mathbf{x} \in \mathbb{R}_+^d, s \in S,$$

implying that $V_P(\mathbf{x})$ is $(M\sqrt{d})$ -Lipschitz continuous³.

Then PGD achieves the following convergence guarantee:

Proposition 1 (Convergence rate of PGD [21]). If Algorithm 1 is run for T iterations with initial point $\mathbf{x}^{(0)}$, step size $\gamma = \frac{\|\mathbf{x}^* - \mathbf{x}^{(0)}\|}{M\sqrt{dt}}$, then the solution $\hat{\mathbf{x}}^{(T)}$ satisfies:

$$V_P(\hat{\mathbf{x}}^{(T)}) - V_P^* \leq \frac{M\sqrt{d} \|\mathbf{x}^* - \mathbf{x}^{(0)}\|}{\sqrt{T}},$$

i.e., the error is $\varepsilon = O(\sqrt{d}/\sqrt{t})$ at step t , or equivalently, PGD achieves an ε -optimal solution with $\Omega(d/\varepsilon^2)$ iterations.

According to Proposition 1, PGD is neither efficient nor scalable: numerous iterations are required to reach an accurate

³See Definition 1 and Lemma 1 in Section V-A.

Algorithm 2 Exponentiated Gradient Descent (Exp-NUM)

Input: initial point $\mathbf{x}^{(0)}$, step size γ , number of iterations T , global flow rate bound C

Output: final solution $\hat{\mathbf{x}}^{(T)}$

```
1: for  $t \leftarrow 0$  to  $T - 1$  do
2:    $\mathbf{v}^{(t)} \leftarrow \nabla V_P(\mathbf{x}^{(t)})$ 
3:    $\mathbf{x}^{(t+1)} \leftarrow C \frac{\mathbf{x}^{(t)} \odot \exp(-\gamma \mathbf{v}^{(t)})}{C + \sum_{s \in S} x_s^{(t)} [\exp(-\gamma v^{(t)}) - 1]}$ 
4: end for
5: return  $\hat{\mathbf{x}}^{(T)} \leftarrow \frac{1}{T} \sum_{t=1}^T \mathbf{x}^{(t)}$ 
```

solution because of the slow convergence rate with respect to t , and the iterative complexity grows proportionally to the number of network flows d .

Exp-NUM [18] is the state-of-the-art gradient method for the NUM problem. Considering problem (NUM-P) with the α -fair utility function U_α and exact penalty function $P_{e,+}$, authors of Exp-NUM propose an exponentiated gradient descent method whose iteration complexity is nearly independent of the number of flows. In Exp-NUM, besides Assumption 4, another assumption is made:

Assumption 5. The sum of flow rates is bounded by a parameter C , thus the feasible region of (NUM-P) becomes $\mathcal{X} = \{\mathbf{x} \in \mathbb{R}_+^d : x_s \geq 0 \text{ and } \sum_{s \in S} x_s \leq C\}$, where $C \geq 0$.

The pseudocode of Exp-NUM are shown in Algorithm 2. At each iteration, it updates $\mathbf{x}^{(t+1)}$ with an exponential function of the gradient. The final solution is still the averaged point $\hat{\mathbf{x}}^{(T)}$. Exp-NUM has the following convergence rate guarantee:

Proposition 2 (Convergence rate of Exp-NUM [18]). If Algorithm 2 is run for T iterations with initial point $\mathbf{x}^{(0)}$ and step size $\gamma = M^{-1} \sqrt{2D(\mathbf{x}^*, \mathbf{x}^{(0)})/t}$, then the solution $\hat{\mathbf{x}}^{(T)}$ satisfies:

$$V_P(\hat{\mathbf{x}}^{(T)}) - V_P^* \leq \frac{M \sqrt{2CD(\mathbf{x}^*, \mathbf{x}^{(0)})}}{\sqrt{T}}, \quad (4)$$

where $D(\mathbf{x}^*, \mathbf{x}^{(0)})$ is the Bregman divergence [18] between $\mathbf{x}^*$ and $\mathbf{x}^{(0)}$. If $\mathbf{x}^{(0)}$ is the barycenter of $\mathcal{X}$, then $D(\mathbf{x}^*, \mathbf{x}^{(0)})$ is less than $C \ln(d+1)$, and ExpNUM achieves $\varepsilon = O(\sqrt{\ln d / \sqrt{t}})$ error at step t , or equivalently, Exp-NUM achieves an ε -optimal solution with $\Omega(\ln d / \varepsilon^2)$ iterations.

There are a few things to note about PGD and Exp-NUM:

1) Both PGD and Exp-NUM are not efficient and need a lot of iterations to converge. The advantage of Exp-NUM compared with PGD is that the iteration complexity is in logarithmic time relative to d , thus the algorithm is scalable with respect to the number of flows. However, its convergence rate is still proportional to $1/\sqrt{t}$.

2) PGD and Exp-NUM uses the average point $\hat{\mathbf{x}}^{(T)}$ as the final solution, instead of the point at the last iteration $\mathbf{x}^{(T)}$. The reason is that the intermediate solutions at each iteration may oscillate around the optimal solution and every-step progress cannot be guaranteed, while using the average point yields a provable convergence theorem.

3) Assumption 4 and Assumption 5 are not valid, so the convergence analyses of PGD and Exp-NUM, i.e., Proposi-

tion 1 and Proposition 2, are not solid. Assumption 4 does not hold for $V_P(\mathbf{x})$ with the α -fair utility function U_α and exact penalty function $P_{e,+}$. First, $P_{e,+}(\mathbf{A}_e \mathbf{x})$ is non-differential when $\mathbf{A}_e \mathbf{x} = \mathbf{c}_e$, so $\frac{\partial V_P(\mathbf{x})}{\partial x_s}$ cannot be calculated. Second, even if it is artificially specified that $\frac{\partial P_{e,+}(\mathbf{A}_e \mathbf{x})}{\partial x_s} = 0$ when $\mathbf{A}_e \mathbf{x} = \mathbf{c}_e$, still there is no upper bound M on $\left| \frac{\partial V_P(\mathbf{x})}{\partial x_s} \right|$. By Eq. (2), $U'_s(x_s) = x_s^{-\alpha}$. So when $x_s = 0$, $U'_s(x_s) = \infty$, and $\left| \frac{\partial V_P(\mathbf{x})}{\partial x_s} \right| = \infty$. Thus the partial derivatives of $V_P(\mathbf{x})$ over $\mathbf{x} \in \mathbb{R}_+^d$ (or $\mathbf{x} \in \mathcal{X}$) are not bounded. Exp-NUM chooses to add a small number to the argument x_s of $U_\alpha(x_s)$ to handle this problem in the experiments, but no theoretical analysis is given. (See the next section for detail.) Assumption 5 alters the feasible region of (NUM-P). The optimal solution of (NUM-P) is not changed only if $C \geq \sum_{e \in E} c_e$.

V. NUM PROBLEM WITH SMOOTH OBJECTIVE FUNCTION

In this section, we propose the NUM problem formulation with smooth objective function. We start with some preliminary definitions and lemmas. And then we design a smooth utility function and a smooth penalty function respectively to construct the smooth objective function. The obtained NUM problem benefits the use of PGD and Exp-NUM algorithms. More importantly, it enables the use of the accelerated gradient method presented in the next section.

A. Preliminaries

We define the concept *Lipschitz continuous* and *smooth* with respect to the Euclidean norm $\|\cdot\|$, and introduce some relevant properties. The proof of the lemmas is trivial and straightforward, so it is omitted.

Definition 1 (Lipschitz continuous). A function f is *L-Lipschitz continuous* (or *L-Lipschitz* for short) on $Q \in \mathbb{R}^n$ with *Lipschitz constant* $L \geq 0$, if for all $\mathbf{x}, \mathbf{y} \in Q$, we have

$$\|f(\mathbf{x}) - f(\mathbf{y})\| \leq L \|\mathbf{x} - \mathbf{y}\|$$

Lemma 1. A function f is *L-Lipschitz* on $Q \in \mathbb{R}^n$ if and only if there exists $L \geq 0$ so that $\|\nabla f(\mathbf{x})\| \leq L$ for all $\mathbf{x} \in Q$.

Lemma 2. The composition of Lipschitz functions is Lipschitz. Specifically, if a function $f : X \rightarrow Y$ is L_f -Lipschitz on X , and a function $g : Y \rightarrow Z$ is L_g -Lipschitz on Y , then the function $g \circ f : X \rightarrow Z$ is $(L_f L_g)$ -Lipschitz on X .

Definition 2 (Smooth). A function f is β -smooth on $Q \in \mathbb{R}^n$ with *smooth constant* $\beta \geq 0$, if f has β -Lipschitz continuous gradient, i.e., for all $\mathbf{x}, \mathbf{y} \in Q$, we have

$$\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \leq \beta \|\mathbf{x} - \mathbf{y}\| \quad (5)$$

Lemma 3. The sum of smooth functions is smooth. Specifically, if a function $f : X \rightarrow Y$ is β_f -smooth on X , and a function $g : X \rightarrow Y$ is β_g -smooth on X , then the function $(f + g) : X \rightarrow Z$ is $(\beta_f + \beta_g)$ -smooth on X .

B. Smooth Utility Function

According to the analysis at the end of Section IV and Lemma 1, the α -fair function is not Lipschitz continuous. Its corresponding network utility function $U(\mathbf{x}) =$

$\sum_{s \in S} U_\alpha(x_s)$ is not smooth, either. $\|\nabla U(\mathbf{x}) - \nabla U(\mathbf{y})\| = \sqrt{\sum_{s \in S} (x_s^{-\alpha} - y_s^{-\alpha})^2}$. If $x_s = 0$ for some $s \in S$, then $x_s^{-\alpha} = \infty$ and $\|\nabla U(\mathbf{x}) - \nabla U(\mathbf{y})\| = \infty$. Thus, Eq. (5) does not hold. Clearly, the “infinity” problem occurs when some x_s approaches 0. So we define the (α, ξ) -fair function as:

$$U_{\alpha, \xi}(x_s) = \begin{cases} \frac{(x_s + \xi)^{1-\alpha}}{1-\alpha}, & \alpha \geq 0, \alpha \neq 1 \\ \ln(x_s + \xi), & \alpha = 1 \end{cases}, \quad (6)$$

where ξ is a small number and $\xi > 0$. Because the formulation of the (α, ξ) -fair function is similar to the α -fair function, so we think it can be used as a link utility function. Actually, it is conventional to add a small number to the argument x_s of the α -fair function to avoid numerical issues when $x_s \rightarrow 0$. For example, Exp-NUM [18] and Primal-dual [13] did so in their experiments. However, existing works view the function $U_{\alpha, \xi}(x_s)$ as a “mathematical trick” instead of using it in problem formulation, and there is no theoretical analysis of the mathematical property of $U_{\alpha, \xi}(x_s)$. Here we show that it yields a smooth network utility function.

Lemma 4. The function $U(\mathbf{x}) = \sum_{s \in S} U_{\alpha, \xi}(x_s)$ is $\frac{\alpha}{\xi^{\alpha+1}}$ -smooth on $\mathbb{R}_+^d$.

Proof. First we define function $\varphi : \mathbb{R}_+ \rightarrow \mathbb{R}_+$ as:

$$\varphi(x) = (x + \xi)^{-\alpha},$$

where $\alpha \geq 0, \xi > 0$. Then we have:

$$|\varphi'(x)| = \left| \frac{\alpha}{(x + \xi)^{\alpha+1}} \right| \leq \frac{\alpha}{\xi^{\alpha+1}}, \quad \forall x \geq 0.$$

By Lemma 1, $\varphi(x)$ is $\frac{\alpha}{\xi^{\alpha+1}}$ -Lipschitz continuous. Then by Definition 1,

$$|\varphi(x) - \varphi(y)| \leq \frac{\alpha}{\xi^{\alpha+1}} |x - y|, \quad \forall x \geq 0, y \geq 0. \quad (7)$$

By Eq. (6),

$$\nabla U(\mathbf{x}) = \nabla \left[\sum_{s \in S} U_{\alpha, \xi}(x_s) \right] = ((x_s + \xi)^{-\alpha})_{s \in S}. \quad (8)$$

Then for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}_+^d$, we have:

$$\begin{aligned} & \|\nabla U(\mathbf{x}) - \nabla U(\mathbf{y})\| \\ &= \sqrt{\sum_{s \in S} [(x_s + \xi)^{-\alpha} - (y_s + \xi)^{-\alpha}]^2} \quad (\text{by Eq. (8)}) \\ &\leq \sqrt{\sum_{s \in S} \left[\frac{\alpha}{\xi^{\alpha+1}} |x_s - y_s| \right]^2} \quad (\text{by Eq. (7)}) \\ &= \frac{\alpha}{\xi^{\alpha+1}} \|\mathbf{x} - \mathbf{y}\|. \end{aligned}$$

By Definition 2, $U(\mathbf{x}) = \sum_{s \in S} U_{\alpha, \xi}(x_s)$ is $\frac{\alpha}{\xi^{\alpha+1}}$ -smooth. $\square$

Remark. A special case is when $\alpha = 0$. In this case, the function $U(\mathbf{x}) = \sum_{s \in S} U_0(x_s) = \sum_{s \in S} x_s$ is already 0-smooth, so there is no need to use $U_{0, \xi}(x_s)$. In addition, $\sum_{s \in S} U_{0, \xi}(x_s)$ is also 0-smooth for any $\xi \in \mathbb{R}$. Thus, Lemma 4 still holds for $\alpha = 0$. The proof is omitted due to the space limitation.

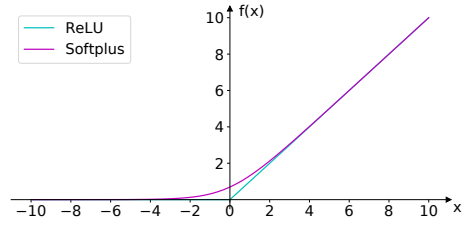


Fig. 1. Graphs of ReLU function and Softplus function.

C. Smooth Penalty Function

The exact penalty function $P_{e,+}$ is not continuously differential and not smooth. Neither does the corresponding network penalty function. There have been works about approximating $P_{e,+}$ with functions with more tractable mathematical properties, see [31]–[33] for example. Usually piecewise functions are used which are difficult to handle, and the functions are continuously differential but the smoothness is not proved.

We notice that the exact penalty function is actually the so called *Rectified Linear Unit (ReLU)* activation function widely used in deep learning. Inspired by it, we propose to use the *Softplus* function defined as

$$f(x) = \ln(1 + \exp(x)), \quad \forall x \in \mathbb{R},$$

to construct the penalty function. Fig. 1 shows the graphs of ReLU function and Softplus function. The two functions behavior similar except near the origin, where the Softplus function is differential and seems to be smooth. Then, for $e \in E$ we define the soft penalty function $P_{e,S}$ as:

$$P_{e,S}(\mathbf{A}_e \mathbf{x}) = \ln(1 + \exp(\mathbf{A}_e \mathbf{x} - c_e)).$$

And we then prove that its corresponding network penalty function is smooth.

Lemma 5. The function $P(\mathbf{x}) = \sum_{e \in E} P_{e,S}(\mathbf{A}_e \mathbf{x})$ is $(\frac{1}{4} \sum_{e \in E} \|\mathbf{A}_e\|^2)$ -smooth on $\mathbb{R}_+^d$.

Proof. For all $e \in E$, we define function $f_e : \mathbb{R}_+ \rightarrow \mathbb{R}_+^d$ as

$$f_e(x) = x \mathbf{A}_e,$$

which is $\|\mathbf{A}_e\|$ -Lipschitz on $\mathbb{R}_+$ by Definition 1, because for all $x, y \in \mathbb{R}_+$, we have

$$\|f_e(x) - f_e(y)\| = \|(x - y) \mathbf{A}_e\| = \|\mathbf{A}_e\| |x - y|.$$

We define function $g : \mathbb{R} \rightarrow \mathbb{R}^+$ as

$$g(x) = f'(x) = \frac{1}{1 + e^{-x}},$$

which is $\frac{1}{4}$ -Lipschitz by Lemma 1, because its derivative satisfies

$$|g'(x)| = \left| \frac{e^{-x}}{(1 + e^{-x})^2} \right| \leq \frac{1}{4}, \quad \forall x \in \mathbb{R}.$$

For all $e \in E$, we define function $h_e : \mathbb{R}_+^d \rightarrow \mathbb{R}$ as

$$h_e(\mathbf{x}) = \mathbf{A}_e \mathbf{x} - c_e,$$

which is $\|\mathbf{A}_e\|$ -Lipschitz on $\mathbb{R}_+^d$ by Definition 1, because for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}_+^d$, we have

$$\|h_e(\mathbf{x}) - h_e(\mathbf{y})\| = \|\mathbf{A}_e \mathbf{x} - \mathbf{A}_e \mathbf{y}\| \leq \|\mathbf{A}_e\| \|\mathbf{x} - \mathbf{y}\|.$$

For all $e \in E$, we define function $\tilde{P}_{e,S}(\mathbf{x}) = P_{e,S}(\mathbf{A}_e \mathbf{x})$, whose gradient is

$$\nabla \tilde{P}_{e,S}(\mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{A}_e \mathbf{x} + c_e)} \mathbf{A}_e = f_e \circ g \circ h_e(\mathbf{x}).$$

By Lemma 2, $\nabla \tilde{P}_{e,S}(\mathbf{x})$ is $(\frac{1}{4} \|\mathbf{A}_e\|^2)$ -Lipschitz on $\mathbb{R}_+^d$, i.e., $\tilde{P}_{e,S}(\mathbf{x})$ is $(\frac{1}{4} \|\mathbf{A}_e\|^2)$ -smooth on $\mathbb{R}_+^d$. Then by Lemma 3, the function $P(\mathbf{x}) = \sum_{e \in E} P_{e,S}(\mathbf{A}_e \mathbf{x})$ is $(\frac{1}{4} \sum_{e \in E} \|\mathbf{A}_e\|^2)$ -smooth on $\mathbb{R}_+^d$. $\square$

D. NUM with Smooth Objective Function

With the smooth utility function and smooth penalty function, the NUM problem is now defined as:

$$\begin{aligned} \min \quad & V_S(\mathbf{x}) = - \sum_{s \in S} U_{\alpha, \xi}(x_s) + \mu \sum_{e \in E} P_{e,S}(\mathbf{A}_e \mathbf{x}) \\ \text{s.t.,} \quad & x_s \geq 0, \forall s \in S \end{aligned} \quad (\text{NUM-S})$$

Theorem 1. The function V_S is β_V -smooth on $\mathbb{R}_+^d$, where $\beta_V = \frac{\alpha}{\xi^{\alpha+1}} + \frac{\mu}{4} \sum_{e \in E} \|\mathbf{A}_e\|^2 = O(d)$.

Proof. We define $U(\mathbf{x}) = \sum_{s \in S} U_{\alpha, \xi}(x_s)$ and $P(\mathbf{x}) = \sum_{e \in E} P_{e,S}(\mathbf{A}_e \mathbf{x})$, then

$$\nabla V_S(\mathbf{x}) = -\nabla U(\mathbf{x}) + \mu \nabla P(\mathbf{x}).$$

Then for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}_+^d$, we have

$$\begin{aligned} & \|\nabla V_S(\mathbf{x}) - \nabla V_S(\mathbf{y})\| \\ &= \|\nabla U(\mathbf{y}) - \nabla U(\mathbf{x}) + \mu \nabla P(\mathbf{x}) - \mu \nabla P(\mathbf{y})\| \\ &\leq \|\nabla U(\mathbf{y}) - \nabla U(\mathbf{x})\| + \mu \|\nabla P(\mathbf{x}) - \nabla P(\mathbf{y})\| \\ &\leq (\frac{\alpha}{\xi^{\alpha+1}} + \frac{\mu}{4} \sum_{e \in E} \|\mathbf{A}_e\|^2) \|\mathbf{x} - \mathbf{y}\| \quad (\text{by Lemma 4, 5}) \end{aligned}$$

By Definition 2, the function V_S is $(\frac{\alpha}{\xi^{\alpha+1}} + \frac{\mu}{4} \sum_{e \in E} \|\mathbf{A}_e\|^2)$ -smooth on $\mathbb{R}_+^d$. Besides, by definition of the routing matrix $\mathbf{A}$ (Eq. (1)), we have:

$$\beta_V = \frac{\alpha}{\xi^{\alpha+1}} + \frac{\mu}{4} \sum_{e \in E} \|\mathbf{A}_e\|^2 \leq \frac{\alpha}{\xi^{\alpha+1}} + \frac{\mu}{4} |E| d. \quad (9)$$

By Assumption 1 and Assumption 2, $|E| \ll d$. Thus, by Eq. (9), $\beta_V = O(d)$. $\square$

One benefit of (NUM-S) is that Assumption 4 holds for its objective function V_S . For all $\mathbf{x} \in \mathbb{R}_+^d, s \in S$, $|\frac{\partial V_S(\mathbf{x})}{\partial x_s}| \leq M = \max\{\xi^{-\alpha}, \mu \sum_{e \in E} \mathbf{A}_{es} \forall s \in S\}$. Thus, PGD and Exp-NUM can actually be used to solve (NUM-S) and the convergence analyses become solid. Furthermore, because the objective function is now smooth, the convergence guarantee of PGD can be improved as follows:

Proposition 3 (Convergence rate of PGD for (NUM-S) [21]). If Algorithm 1 is run for T iterations with an initial point $\mathbf{x}^{(0)}$, step size $\gamma = \frac{1}{\beta_V}$, then the solution $\mathbf{x}^{(T)}$ satisfies:

$$V_S(\mathbf{x}^{(T)}) - V_S^* \leq \frac{2\beta_V \|\mathbf{x}^* - \mathbf{x}^{(0)}\|^2}{T}. \quad (10)$$

By Theorem 1, $\beta_V = O(d)$. Thus, the error is $\varepsilon = O(d/t)$ at step t , or equivalently, PGD achieves an ε -optimal solution with $\Omega(d/\varepsilon)$ iterations.

For (NUM-S), the convergence rate of PGD is now proportional to $1/t$ instead of $1/\sqrt{t}$, but the iteration complexity is

Algorithm 3 Accelerated Gradient Method (AGM)

Input: initial point $\mathbf{x}^{(0)}$, step size γ , number of iterations T

Output: final solution $\mathbf{x}^{(T)}$

```

1:  $\mathbf{y}^{(0)} \leftarrow \mathbf{x}^{(0)}, a_0 \leftarrow 1$ 
2: for  $t \leftarrow 0$  to  $T-1$  do
3:    $\mathbf{x}^{(t+1)} \leftarrow [\mathbf{y}^{(t)} - \gamma \nabla V_S(\mathbf{y}^{(t)})]_+$ 
4:    $a_{t+1} \leftarrow \frac{1 + \sqrt{4a_t^2 + 1}}{2}$ 
5:    $\mathbf{y}^{(t+1)} \leftarrow \mathbf{x}^{(t+1)} + \frac{a_t - 1}{a_{t+1}} (\mathbf{x}^{(t+1)} - \mathbf{x}^{(t)})$ 
6: end for
7: return  $\mathbf{x}^{(T)}$ 
```

still proportional to d . Note that now the final solution is not the averaged point, but the point at the last iteration. However, Exp-NUM is only applicable to minimizing Lipschitz functions, so its convergence rate remains $O(\sqrt{\ln d}/\sqrt{t})$ for (NUM-S).

VI. ACCELERATED GRADIENT METHOD FOR NUM WITH SMOOTH OBJECTION FUNCTION

In this section, we present the accelerated gradient method (AGM), which is a tailor-made algorithm for smooth objective function.

A. Vanilla Accelerated Gradient Method

We have shown that the proposed (NUM-S) problem benefits the use of PGD and Exp-NUM methods. Furthermore, it also enables the use of the *accelerated gradient method* (AGM) proposed by Nesterov [19], which achieves a superior $O(1/t^2)$ convergence rate for smooth objective functions.

The pseudocode of AGM is shown in Algorithm 3. At each iteration, $\mathbf{x}^{(t+1)}$ is moved towards the opposite gradient direction at the point $\mathbf{y}^{(t)}$ instead of $\mathbf{x}^{(t)}$. Then the point $\mathbf{y}^{(t+1)}$ is updated based on the obtained point $\mathbf{x}^{(t+1)}$ at this iteration and $\mathbf{x}^{(t)}$ at previous iteration, balanced by the parameters a_t and a_{t+1} . AGM can be viewed as a momentum method [34], because it takes steps based on information at not only the current iteration but also previous iterations. It has the following convergence guarantee:

Theorem 2 (Convergence rate of AGM [19]). If Algorithm 3 is run for T iterations with an initial point $\mathbf{x}^{(0)}$, step size $\gamma = \frac{1}{\beta_V}$, then the final solution $\mathbf{x}^{(T)}$ satisfies:

$$V_S(\mathbf{x}^{(T)}) - V_S^* \leq \frac{2\beta_V \|\mathbf{x}^* - \mathbf{x}^{(0)}\|^2}{(T+1)^2}. \quad (11)$$

By Theorem 1, $\beta_V = O(d)$. Thus, the error is $\varepsilon = O(d/t^2)$ at step t , or equivalently, AGM achieves an ε -optimal solution with $\Omega(\sqrt{d}/\sqrt{\varepsilon})$ iterations.

Like PGD and Exp-NUM, AGM is also a first-order method with low per-iteration cost. In addition, AGM has two main advantages over PGD and Exp-NUM:

1) AGM has the fastest convergence rate with respect to t , which is proportional to $1/t^2$. Compared with PGD (convergence rate proportional to $1/t$) and Exp-NUM (convergence

rate proportional to $1/\sqrt{t}$, AGM is much more efficient, and it converges to an accurate solution within a lot less iterations.

2) The iteration complexity of AGM is proportional to $\sqrt{d}$. Although AGM is not as scalable as Exp-NUM with respect to the number of flows, it still can maintain good performance as the number of flows grows.

Remark. The convergence guarantees of PGD, Exp-NUM and AGM hold under different assumptions: Assumption 4 or 3 for PGD (Proposition 1 or 3), Assumption 4-5 for Exp-NUM (Proposition 2), and Assumption 1-3 for AGM (Theorem 1, 2). When Assumption 1-5 are all true, Proposition 2, 3 and Theorem 2 all hold, and it is provable that AGM achieves better solution compared with PGD throughout the whole iteration process, and AGM guarantees to achieve more accurate solution compared with Exp-NUM after enough number of iterations.

Theorem 3. The solution achieved by AGM is better than that achieved by PGD for any number of iterations T .

Proof. By Eq. (10) and Eq. (11), for AGM to achieve better solution, there should be:

$$\begin{aligned} \frac{2\beta_V \|\mathbf{x}^* - \mathbf{x}^{(0)}\|^2}{(T+1)^2} &\leq \frac{2\beta_V \|\mathbf{x}^* - \mathbf{x}^{(0)}\|^2}{T}, \\ \Rightarrow \frac{(T+1)^2}{T} &\geq 1. \end{aligned} \quad (12)$$

Obviously, Eq. (12) holds for any $T \geq 1$. $\square$

Theorem 4. The solution achieved by AGM is better than that achieved by Exp-NUM when the number of iterations is larger than $\left(\frac{2\beta_V \|\mathbf{x}^* - \mathbf{x}^{(0)}\|^4}{M^2 CD(\mathbf{x}^*, \mathbf{x}^{(0)})}\right)^{\frac{1}{3}} - 1$.

Proof. By Eq. (4) and Eq. (11), for AGM to achieve better solution, there should be:

$$\begin{aligned} \frac{2\beta_V \|\mathbf{x}^* - \mathbf{x}^{(0)}\|^2}{(T+1)^2} &\leq \frac{M \sqrt{2CD(\mathbf{x}^*, \mathbf{x}^{(0)})}}{\sqrt{T}}, \\ \Rightarrow \frac{(T+1)^4}{T} &\geq \frac{2\beta_V^2 \|\mathbf{x}^* - \mathbf{x}^{(0)}\|^4}{M^2 CD(\mathbf{x}^*, \mathbf{x}^{(0)})}. \end{aligned} \quad (13)$$

As $\frac{(T+1)^4}{T} \geq \frac{(T+1)^4}{T+1}$, for Eq. (13) to hold, it is sufficient that the following inequality holds:

$$\frac{(T+1)^4}{T+1} \geq \frac{2\beta_V^2 \|\mathbf{x}^* - \mathbf{x}^{(0)}\|^4}{M^2 CD(\mathbf{x}^*, \mathbf{x}^{(0)})}. \quad (14)$$

By Eq. (14), $T \geq \left(\frac{2\beta_V \|\mathbf{x}^* - \mathbf{x}^{(0)}\|^4}{M^2 CD(\mathbf{x}^*, \mathbf{x}^{(0)})}\right)^{\frac{1}{3}} - 1$. $\square$

B. Accelerated Gradient Method with Restart

Although by Theorem 2 AGM converges fast, it does not mean that the obtained objective value sequence is monotone. When running the AGM algorithm, it is common that there are bumps in the objective value trajectory, which empirically slows down the convergence. To deal with this problem, we leverage the restart heuristic proposed by O'donoghue et al

Algorithm 4 Accelerated Gradient Method with Restart

Input: initial point $\mathbf{x}^{(0)}$, step size γ , number of iterations T

Output: final solution $\mathbf{x}^{(T)}$

```

1:  $\mathbf{y}^{(0)} \leftarrow \mathbf{x}^{(0)}, a_0 \leftarrow 1$ 
2: for  $t \leftarrow 0$  to  $T-1$  do
3:    $\mathbf{x}^{(t+1)} \leftarrow [\mathbf{y}^{(t)} - \gamma \nabla V_S(\mathbf{y}^{(t)})]_+$ 
4:    $a_{t+1} \leftarrow \frac{1 + \sqrt{4a_t^2 + 1}}{2}$ 
5:    $\mathbf{y}^{(t+1)} \leftarrow \mathbf{x}^{(t+1)} + \frac{a_t - 1}{a_{t+1}}(\mathbf{x}^{(t+1)} - \mathbf{x}^{(t)})$ 
6:   if Restart Condition then
7:      $\mathbf{y}^{(t+1)} \leftarrow \mathbf{x}^{(t+1)}, a_{t+1} \leftarrow 0$ 
8:   end if
9: end for
10: return  $\mathbf{x}^{(T)}$ 

```

[35]. They propose to “restart” the algorithm, i.e., taking the current point as a new initial point, when certain restart condition holds. In this paper, we consider two restart conditions:

- 1) function restart: $V_S(\mathbf{x}^{(k+1)}) > V_S(\mathbf{x}^{(k)})$;
- 2) gradient restart: $[\nabla V_S(\mathbf{y}^{(k)})]^\top (\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}) > 0$.

The pseudocode of AGM with restart is shown in Algorithm 4. At each iteration, the algorithm checks whether the restart condition holds. If so, $\mathbf{y}^{(t+1)}$ is set to $\mathbf{x}^{(t+1)}$ and the parameter a_{t+1} is reset to 1, so that the algorithm is restarted by the current point. Although the restart method boosts convergence empirically, there is no theoretical guarantee on the convergence rate.

VII. EVALUATION

In this section, we evaluate the smooth penalty-based NUM formulation and the proposed algorithm.

A. Setup

1) *Topology:* We use two topologies GÉANT (23 nodes, 74 links) [36] and RF1221 (104 nodes, 302 links) [37]. The shortest path is used as the predetermined path for each flow, so elements in the routing matrix are integer. To better display the result, the link capacities are scaled down so that $c_e \leq 100, \forall e \in E$. Because the flow rates x_s are decision variables in the NUM problem, the traffic matrix is not needed in the experiments.

2) *Optimization Objective:* We use two different utility functions. One is $\alpha = 0, \xi = 0$, so that the network utility function is $U(\mathbf{x}) = \sum_{s \in S} x_s$, corresponding to throughput maximization. The other is $\alpha = 1, \xi = 0.5$, so that network utility function is $U(\mathbf{x}) = \sum_{s \in S} \ln(x_s + 0.5)$, corresponding to proportional fairness. μ is set to 2 empirically.

3) *Algorithms:* We solve the problem (NUM-S) with PGD, Exp-NUM (C is set to $\sum_{e \in E} c_e$), AGM, AGM with function restart (i.e., AGM-fr) and AGM with gradient restart (i.e., AGM-gr).

All algorithms are implemented with Python. Computations are performed on a server with 2.40GHz Intel CPU and 512G memory.

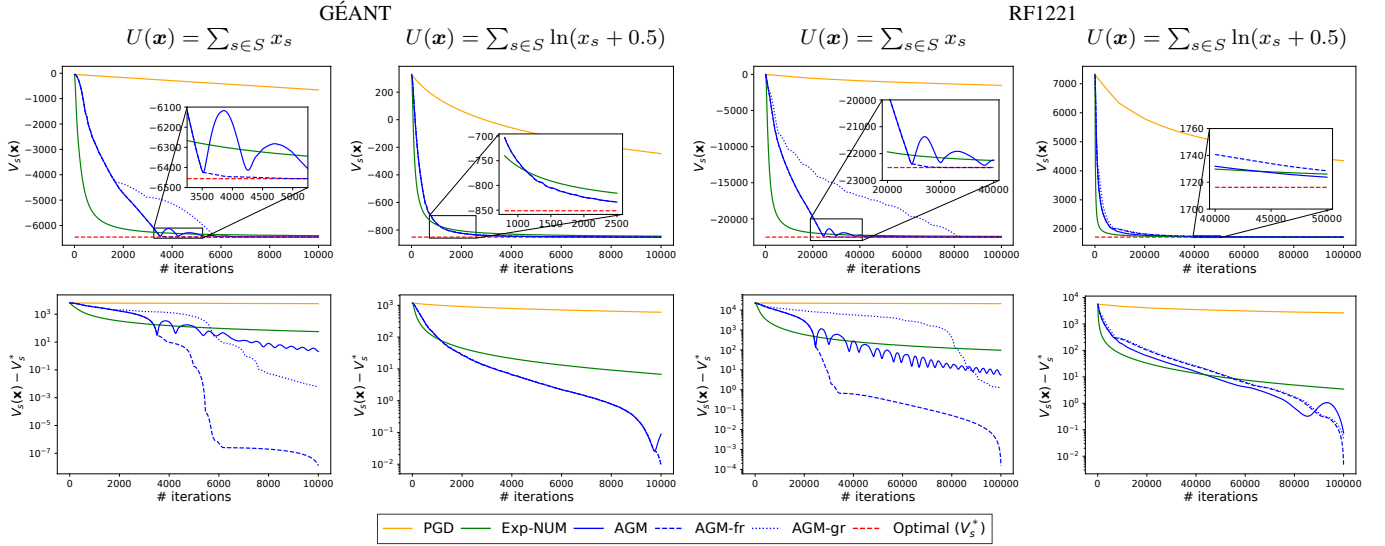


Fig. 2. The objective value and error at each iteration in GÉANT and RF1221.

B. Convergence Rate

We evaluate the convergence rate of the algorithms. We assume that there is one flow between each source-destination pairs, and run the algorithms with the aforementioned objective. In Fig. 2, we plot the objective value (at top) and corresponding error (at bottom in log scale) in GÉANT and RF1221 at the first 10000 and 100000 iterations respectively.

First, experiments show that AGM has the fastest convergence rate and outperforms PGD and Exp-NUM, which agrees with the theoretical results. Given a tolerable error, our method will solve the NUM problem much quicker, which benefits the network resource allocation. As proved in Theorem 3, PGD converges the slowest in experiment, mainly because its iteration complexity is $\Omega(d/\varepsilon)$, which is proportion to d . When comparing AGM and Exp-NUM, the experiment results is in consistent with Theorem 4. Exp-NUM has $O(\sqrt{\ln d}/\sqrt{t})$ convergence rate, which is nearly independent on the number of flows d , thus it converges fast in the beginning. However, the convergence slows down later and Exp-NUM can only achieve $10 \sim 10^2$ error at end. In contrast, AGM converges a little slower at first, but it performs better and obtains more accurate solutions as the number of iterations grows, because of its $O(d/t^2)$ convergence rate, which is the fastest with respect to t .

Second, the restart method boosts the convergence of AGM and alleviates the bumps. Although AGM converges a little faster at first, AGM-fr and AGM-gr outperform it as the iterations continue. In particular, AGM-fr achieves 10^{-2} -optimal solution for all objective functions and topologies, and it even achieves 10^{-7} -optimal solution for $U_s = \sum_{s \in S} x_s$ in GÉANT. Besides, there are less bumps in the curves of AGM-fr and AGM-gr.

C. Network Utility Optimization

The objective of (NUM-S) is the smooth penalized function $V_s(x)$, instead of the network utility function $U(x)$. In order to evaluate the performance of the algorithms in terms of network

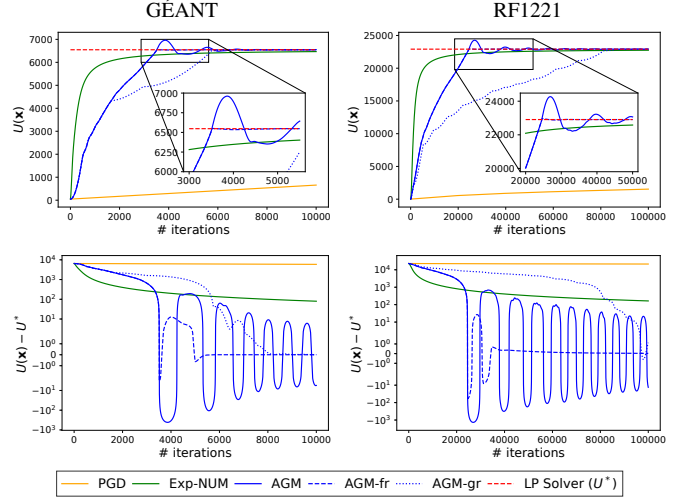


Fig. 3. The network utility value and error at each iteration when $U_s = \sum_{s \in S} x_s$ in GÉANT and RF1221.

utility maximization, we take $U(x) = \sum_{s \in S} x_s$ for example so that the problem (NUM) is a linear programming (LP) problem aiming at maximizing throughput and can be solved by commercial LP solvers. We denote its optimal value as U^* . And we record the $U(x)$ (at top) and error with respect to U^* (at bottom in symmetric log scale) during iterations when solving (NUM-S). The results are shown in Fig. 3.

First, the rate at which $U(x)$ approaches U^* is consistent with the convergence rate of the algorithms, i.e., Exp-NUM performs well at first but AGM surpasses it later and PGD hardly converges. Hence, the convergence rate implies the efficiency of the algorithms in solving NUM problem.

Second, AGM causes obvious bumps in $U(x)$. Although it converges faster than Exp-NUM, it tends to overshoot the solution and oscillates around U^* . In contrast, AGM-fr obtains stable results while maintaining the fast convergence rate.

Third, AGM-fr achieves close to optimal network utility value, which means that the smooth penalty-based formulation

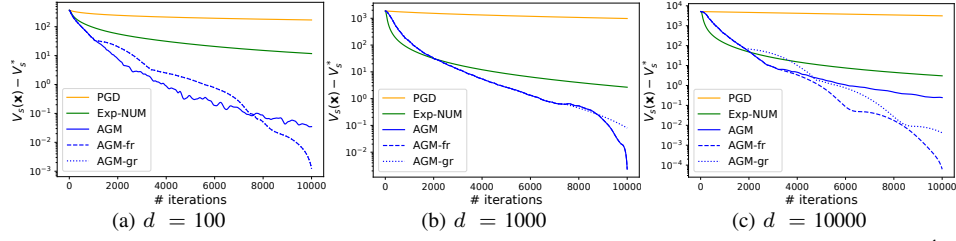


Fig. 4. The error at each iteration with different number of flows d when $U_s = \sum_{s \in S} \ln(x_s + 0.5)$ in GÉANT.

TABLE III

THE ACHIEVED NETWORK UTILITY VALUE AND ERROR (IN PARENTHESES) WHEN $U_s = \sum_{s \in S} x_s$ IN GÉANT AND RF1221.

Algorithm	GÉANT	RF1221
PGD	657.09 (5890.11)	1533.36 (21374.64)
Exp-NUM	6467.39 (79.81)	22744.75 (163.26)
AGM	6552.78 (-5.58)	22921.28 (-13.28)
AGM-fr	6547.199 (0.001)	22907.97 (0.03)
AGM-gr	6547.19 (0.01)	22906.87 (1.13)
LP Solver	6547.20	22908.00

TABLE IV

AVERAGE COMPUTATION TIME PER ITERATION WITH DIFFERENT NUMBER OF FLOWS d WHEN $U_s = \sum_{s \in S} \ln(x_s + 0.5)$ IN GÉANT.

Algorithm	$d=100$	$d=1000$	$d=10000$
PGD	$7.2 \times 10^{-5}s$	$1.7 \times 10^{-4}s$	$1.1 \times 10^{-3}s$
Exp-NUM	$9.3 \times 10^{-5}s$	$2.7 \times 10^{-4}s$	$1.3 \times 10^{-3}s$
AGM	$8.0 \times 10^{-5}s$	$2.0 \times 10^{-4}s$	$1.2 \times 10^{-3}s$
AGM-fr	$1.5 \times 10^{-4}s$	$3.4 \times 10^{-4}s$	$1.8 \times 10^{-3}s$
AGM-gr	$8.4 \times 10^{-5}s$	$2.3 \times 10^{-4}s$	$1.3 \times 10^{-3}s$

and our algorithm can help solve the original NUM problem effectively. In Table III, we record the $U(x)$ and corresponding error at the last iteration. Still, PGD does not converge, and the error achieved by Exp-NUM is unsatisfactory. AGM overshoots the result while the restart methods perform well. AGM-fr achieves quite accurate $U(x)$ in both topologies.

D. Performance with Different d

We evaluate the impact of the number of flows d on algorithm performance. Due to the space limitation, we take GÉANT with $U_s = \sum_{s \in S} \ln(x_s + 0.5)$ for example, and run the algorithms when there are 100, 1000, 10000 flows in the network respectively. The results are similar for RF1221 and $U_s = \sum_{s \in S} x_s$.

Fig. 4 shows the error at each iteration. We've proven that the iteration complexity of AGM is $\Omega(\sqrt{d}/\varepsilon)$, so that AGM can keep good performance as d increases. And the experiment results are consistent with the theoretical results. As show in Fig. 4, different d has no obvious effect on the convergence of AGM, it performs consistently well. Besides, although the iteration complexity of Exp-NUM is said to be nearly independent on d , our method AGM, AGM-fr and AGM-gr outperforms Exp-NUM with all d settings.

In Table IV we record the average computation time per iteration of different algorithms. As d increases, the computa-

tion time per iteration also increases. With the same d , the time per iteration of PGD, Exp-NUM, and AGM are similar while Exp-NUM takes a little longer time for the computation of the exponential function, because they are all first-order methods. Thus, the difference in their convergence rates implies their different computation time to solve the problem. AGM-fr takes longer time because of the additional objective value computation when judging restart condition. Considering its fast and stable convergence, we think it's worth it.

VIII. LIMITATIONS

Our method AGM aims at solving NUM problem with smooth utility functions. Although the most commonly studied α -fair utility function is proven to be smooth, in general, however, utility functions are not necessarily smooth or even not concave, which prevents the use of our method. Besides, one concern about the Softplus function is that the penalty value is although small, but still greater than zero when $\mathbf{A}_e \mathbf{x} \leq c_e$. Such unwanted penalty can be mitigated by further moving the Softplus function towards the right of the horizontal axis or fine-tuning the parameter μ . The convergence guarantee of AGM holds only when the number of links are much smaller than the number of flows in the network.

IX. CONCLUSION

In this paper, we propose a centralized, efficient and scalable accelerated gradient method (AGM) for the centralized NUM problem. We discover the smoothness of the α -fair utility function, and formulate the NUM problem with smooth objective function, which not only benefits the existing method, but also enables the use of Nesterov's accelerated gradient method. We prove that AGM has the fastest $O(d/t^2)$ convergence rate with respect to t , and has $\Omega(\sqrt{d}/\sqrt{\varepsilon})$ iteration complexity which is scalable with respect to the number of flows d . Experiment results are in good agreement with the theoretical results and demonstrate the great potential of AGM for resource allocation in large-scale centralized controlled networks. In the future, we will evaluate AGM with other smooth utility functions and penalty functions, and investigate its performance in real-world use cases.

ACKNOWLEDGMENT

This work is supported in part by the National Key Research and Development Program of China under Grant No. 2018YFB1800400 and National Natural Science Foundation of China under Grant No. 62002009. We thank the anonymous reviewers for their valuable comments.

REFERENCES

- [1] F. Kelly, "Charging and rate control for elastic traffic," *European transactions on Telecommunications*, vol. 8, no. 1, pp. 33–37, 1997.
- [2] J. Mo and J. Walrand, "Fair end-to-end window-based congestion control," in *Performance and Control of Network Systems II*, vol. 3530. SPIE, 1998, pp. 55 – 63.
- [3] —, "Fair end-to-end window-based congestion control," *IEEE/ACM Transactions on Networking*, vol. 8, no. 5, pp. 556–567, 2000.
- [4] A. Eryilmaz and R. Srikant, "Joint congestion control, routing, and mac for stability and fairness in wireless networks," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 8, pp. 1514–1524, 2006.
- [5] N. Karakoç, A. Scaglione, A. Nedić, and M. Reisslein, "Multi-layer decomposition of network utility maximization problems," *IEEE/ACM Transactions on Networking*, vol. 28, no. 5, pp. 2077–2091, 2020.
- [6] L. Gu, D. Zeng, S. Tao, S. Guo, H. Jin, A. Y. Zomaya, and W. Zhuang, "Fairness-aware dynamic rate control and flow scheduling for network utility maximization in network service chain," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 5, pp. 1059–1071, 2019.
- [7] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan, "Rate control for communication networks: shadow prices, proportional fairness and stability," *Journal of the Operational Research Society*, vol. 49, no. 3, pp. 237–252, 1998.
- [8] S. Low and D. Lapsley, "Optimization flow control. I. basic algorithm and convergence," *IEEE/ACM Transactions on Networking*, vol. 7, no. 6, pp. 861–874, 1999.
- [9] R. La and V. Anantharam, "Utility-based rate control in the internet for elastic traffic," *IEEE/ACM Transactions on Networking*, vol. 10, no. 2, pp. 272–286, 2002.
- [10] D. Palomar and M. Chiang, "A tutorial on decomposition methods for network utility maximization," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 8, pp. 1439–1451, 2006.
- [11] S. Shakkottai and R. Srikant, "Network optimization and control," *Found. Trends Netw.*, vol. 2, no. 3, pp. 271–379, 2007.
- [12] H. Yu and M. J. Neely, "A simple parallel algorithm with an $O(1/t)$ convergence rate for general convex programs," *SIAM Journal on Optimization*, vol. 27, no. 2, pp. 759–783, 2017.
- [13] A. Beck, A. Nedić, A. Ozdaglar, and M. Teboulle, "An $O(1/k)$ gradient method for network resource allocation problems," *IEEE Transactions on Control of Network Systems*, vol. 1, no. 1, pp. 64–73, 2014.
- [14] O. N. Foundation, "Software-defined networking: The new norm for networks," *ONF White Paper*, 2012.
- [15] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, "B4: Experience with a globally-deployed software defined wan," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 3–14.
- [16] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven wan," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 15–26.
- [17] R. Gupta, L. Vandenbergh, and M. Gerla, "Centralized network utility maximization over aggregate flows," in *2016 14th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, 2016, pp. 1–8.
- [18] L. Vigneri, G. Paschos, and P. Mertikopoulos, "Large-scale network utility maximization: Countering exponential growth with exponentiated gradients," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp. 1630–1638.
- [19] Y. E. Nesterov, "A method for solving the convex programming problem with convergence rate $O(1/k^2)$," in *Dokl. akad. nauk Sssr*, vol. 269, 1983, pp. 543–547.
- [20] Y. Nesterov and V. Shikhman, "Dual subgradient method with averaging for optimal resource allocation," *European Journal of Operational Research*, vol. 270, no. 3, pp. 907–916, 2018.
- [21] S. Bubeck, "Convex optimization: Algorithms and complexity," *Foundations and Trends® in Machine Learning*, vol. 8, no. 3-4, pp. 231–357, 2015.
- [22] Z. Allybokus, K. Avrachenkov, J. Leguay, and L. Maggi, "Real-time fair resource allocation in distributed software defined networks," in *2017 29th International Teletraffic Congress (ITC 29)*, vol. 1, 2017, pp. 19–27.
- [23] E. Wei, A. Ozdaglar, and A. Jadbabaie, "A distributed newton method for network utility maximization," in *49th IEEE Conference on Decision and Control (CDC)*, 2010, pp. 1816–1821.
- [24] M. Zargham, A. Ribeiro, and A. Jadbabaie, "Accelerated backpressure algorithm," in *2013 IEEE Global Communications Conference (GLOBECOM)*, 2013, pp. 2269–2275.
- [25] M. Zargham, A. Ribeiro, A. Ozdaglar, and A. Jadbabaie, "Accelerated dual descent for network flow optimization," *IEEE Transactions on Automatic Control*, vol. 59, no. 4, pp. 905–920, 2014.
- [26] A. S. Nemirovskij and D. B. Yudin, *Problem complexity and method efficiency in optimization*. Wiley-Interscience, 1983.
- [27] S. Knight, H. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *Selected Areas in Communications, IEEE Journal on*, vol. 29, no. 9, pp. 1765 –1775, october 2011.
- [28] C. Filsfils, N. K. Nainar, C. Pignataro, J. C. Cardona, and P. Francois, "The segment routing architecture," in *2015 IEEE Global Communications Conference (GLOBECOM)*, 2015, pp. 1–6.
- [29] L. Massoulié and J. Roberts, "Bandwidth sharing: objectives and algorithms," in *IEEE INFOCOM '99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No.99CH36320)*, vol. 3, 1999, pp. 1395–1403 vol.3.
- [30] W. I. Zangwill, "Non-linear programming via penalty functions," *Management Science*, vol. 13, no. 5, pp. 344–358, 1967.
- [31] M. Ç. Pinar and S. A. Zenios, "On smoothing exact penalty functions for convex constrained optimization," *SIAM Journal on Optimization*, vol. 4, no. 3, pp. 486–511, 1994.
- [32] X. Xu, Z. Meng, J. Sun, L. Huang, and R. Shen, "A second-order smooth penalty function algorithm for constrained optimization problems," *Computational optimization and applications*, vol. 55, no. 1, pp. 155–172, 2013.
- [33] S. A. Zenios, M. Ç. Pinar, and R. S. Dembo, "A smooth penalty function algorithm for network-structured problems," *European Journal of Operational Research*, vol. 83, no. 1, pp. 220–236, 1995. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/037722179590601A>
- [34] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *International conference on machine learning*. PMLR, 2013, pp. 1139–1147.
- [35] B. O'donoghue and E. Candes, "Adaptive restart for accelerated gradient schemes," *Foundations of computational mathematics*, vol. 15, no. 3, pp. 715–732, 2015.
- [36] S. Uhlig, B. Quoitin, J. Lepropre, and S. Balon, "Providing public intradomain traffic matrices to the research community," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 1, pp. 83–86, 2006.
- [37] R. Hartert, S. Vissicchio, P. Schaus, O. Bonaventure, C. Filsfils, T. Telkamp, and P. Francois, "A declarative and expressive approach to control forwarding paths in carrier-grade networks," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 15–28, 2015.