Hermes: Utility-aware Network Update in Software-defined WANs

Jiaqi Zheng[†], Qiufang Ma[†], Chen Tian[†], Bo Li[†], Haipeng Dai[†], Hong Xu[‡], Guihai Chen[†][¶], Qiang Ni[§]
[†]*State Key Laboratory for Novel Software Technology, Nanjing University, China*
[‡]*City University of Hong Kong, Hong Kong, China*
[¶]*Shanghai Jiao Tong University, Shanghai, China*
[§]*Lancaster University, Lancaster, UK*

***Abstract*—State-of-the-art inter-datacenter WANs rely on software defined networking (SDN) to orchestrate their data transmission. Optimization requires frequent network update operations to switch forwarding tables. When scheduling inter-datacenter WANs, the utility of services should be respected. Yet, existing network update approaches do not respect network utility and could result in performance degradation during the network update procedure. Further, the update causes not only performance degradation, but also the degradation period is unnecessarily prolonged. In this paper we propose Hermes, a utility-aware network update system. We aim to find a rate limiting scheme for update which maximizes the sum of service utility, while ensuring the congestion-free property during the update. We propose an optimization framework for the maximum utility network update problem (MUP). MUP is NP-hard and a series of algorithms are developed to solve it. Extensive simulation and testbed experiments with a prototype demonstrate that Hermes can increase the total utility by 80% compared to state-of-the-art. At the same time, it reduces the total update time and control overhead by 40% and 55%, respectively.**

I. Introduction

Motivation: State-of-the-art inter-datacenter WANs rely on software defined networking (SDN) to orchestrate their data transmission. For example, Google [1] and Microsoft [2] use SDN to interconnect their geo-distributed datacenters. It is well-known that an inter-datacenter WAN is a highly expensive network infrastructure. Via centralized traffic engineering (TE) every several minutes, the link utilization of a SDN-based WAN can approach 100% [1], while traditional networks have only 30% to 40% average utilization. Nevertheless, such optimization requires frequent network update operations to switch forwarding tables.

When scheduling inter-datacenter WANs, the utility of services should be respected. For a specific service, a utility function characterizes the relationship between the allocated bandwidth and the quality-of-service [3]. It reflects the service-level objective, *e.g.*, minimizing the flow completion time, minimizing the deadline missing ratio, *etc.* It enables network operators to evaluate how bandwidth should be allocated to increase the revenue. Initially, SWAN [2] and B4 [1] only use the weighted max-min fairness principle to allocate bandwidth among services. Calendaring [4] and Amoeba [5] then improve performance for deadline-aware services by respecting their objectives. Usually the service-level bandwidth resource is divided equally among its flows especially for the video streaming applications [6]. The service-level and flow-level utility optimization are equivalent in inter-datacenter WANs since the ingress switch aggregates the flows belong to the same service [2].

Existing network update approaches [2], [7], [8], [9] do not respect network utility and could result in performance degradation during the network update procedure. Prior work strives to find a congestion-free multi-stage update plan [7], [2]. To guarantee a update plan always exists, a portion (10%–50% [2]) of the network capacity has to be reserved. If we simply decrease (*e.g.*, proportionally) the allocated rate for each flow, the sum of flow utility could be significantly reduced. Furthermore, calculating a multi-stage update plan requires solving a series of linear programming problems (LPs), one for each stage. This procedure could be very slow especially when the number of stages is large [10]. In addition, for multi-stage update, the controller waits for all the switches to complete their update operations before entering into the next stage [11]. As a result, the update causes not only performance degradation, but also a prolonged degradation period. To the best of our knowledge, our work proposes the first update approach that takes network utility into account.

Our contributions: In this paper we propose Hermes, a utility-aware network update system. We aim to find a rate limiting scheme for update which maximizes the sum of service utilities, while ensuring the congestion-free property during the update procedure. When update is done, all services resume to its original rate. Compared with multi-stage update [2], [7], our approach only requires one update stage and can significantly speed up the update procedure. In addition, we save the computation overhead of solving a series of LPs. Furthermore, we do not need any bandwidth headroom and the network still runs at near full utilization.

Our first contribution is that we propose an optimization framework for the maximum utility network update problem (MUP). Generally speaking, given service-level utility functions, the optimization program aims to determine the reduced rate for each service before update, such that the congestion-free condition is ensured during the asynchronous update (Sec. III).

978-1-5386-6043-0/18/$31.00 ©2018 IEEE
DOI 10.1109/ICNP.2018.00032

Our second contribution is that we develop a series of algorithms to solve MUP. We prove that MUP is NP-hard. We first propose an iterative improvement algorithm to calculate the reduced rate for each service. By using convex relaxation techniques, we next determine a concave envelop for each non-concave utility function and obtain an initial solution using multi-block ADMM techniques. Further we adjust the allocated rate which improves upon the initial solution by greedily increasing the total utility in the network (Sec. IV).

Our third contribution is a concrete implementation and evaluation of Hermes. We develop a prototype of Hermes using Linux `tc` to dynamically adjust the allocated rate and evaluate it on a small-scale testbed. We use OFSoftSwitch and Dpctl [12] as Openflow switches and the controller. Extensive simulation and testbed experiments demonstrate that Hermes can increase the total utility by 80% compared to state-of-the-art. At the same time, it reduces the total update time and control overhead by 40% and 55%, respectively (Sec. V).

II. Background and Motivation

A. Related work

Network utility maximization: There is a rich literature on network utilization maximization (NUM). Dolev et al. [13] use Newton-method-based update steps to speed up the algorithm convergence procedure for solving NUM. Later, Wei et al. [14] take advantage of matrix splitting techniques to further improve the converging time, where the proposed approaches can be computed in a decentralized manner. From the optimization perspective, NUM is used to analyze existing TCP protocols [15] by specifying some special utility functions. Recently NUM is widely used in cloud data centers, pFabric [16] aims to minimize the flow completion time and can provide near optimal rate control schemes. Centralized Fastpass [17] relies on fine-grained packet scheduling to flexibly allocate network resource. NUMFabric [3] considers per-flow bandwidth allocation and optimization, where the allocation policy can be well captured by a NUM problem.

Network update in SDNs: SWAN [2] and zUpdate [7] try to find congestion-free update plans in inter-datacenter and intra-datacenter, respectively. SWAN shows that if each link has certain slack capacity, there always exists a congestion-free update sequence. As the update speed of the dynamic scheduling is different, Dionysus [8] employs dependency graphs to determine a congestion-free update plan according to different runtime conditions of switches. To reduce the dependency complexity, Cupid [9] divides the global update dependencies among switches into local restrictions to avoid high overhead when generating a update plan. Ludwig et al. [18] aim to minimizing the number of sequential controller interactions when transitioning from the initial to the final update stage. The authors prove that finding a shortest node ordering sequence that avoids forwarding loops is NP-hard. Another work by Ludwig et al. [19] considers secure network updates in the presence of middleboxes such as firewalls and NAT. Instead of congestion-free update, Zheng et al. [10] advocate to find an update plan that minimizes the transient

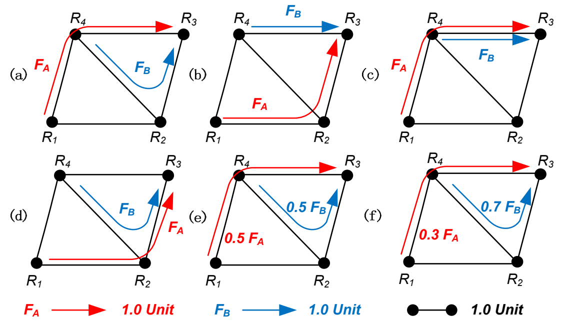


Fig. 1. A motivating example.

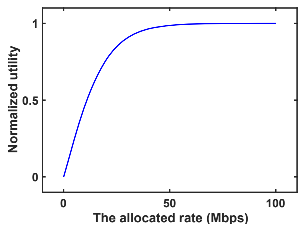
congestion, while it cannot be applied to bandwidth-sensitive data transfers.

Though the idea of NUM and network update have surfaced in the literature, the novelty of our work lies in a comprehensive exploration of integrating NUM into network update, which to our knowledge has not been done before.

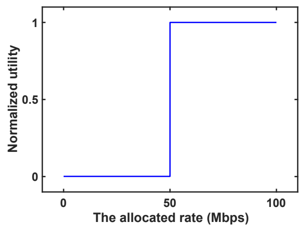
B. Utility function

The traffic on the WAN is a mix of the diverse services [2], [20]. Fig. 2 illustrates the utility functions for four typical services [21]: elastic service, hard real-time service, delay-adaptive service and rate-adaptive service. An example of elastic service is the data backup operation, which periodically transfers the data from one data center to another data center. The delivery usually does not have hard deadline and can tolerate the network delay [22]. The hard real-time services such as the web search, online chatting and gaming are triggered by the the end users. Taking web search for example, one user searched something in the internet and the user requests were routed to the closed data center. When the search results were finished, the search engine wanted to rank the pages based on user's preference (e.g. from user's past social behaviors). Unluckily, the user's preference was located in another data center. At this point, the data center which is processing the user requests need to communicate with the data center which stores the user data. Reducing the allocated bandwidth for this kind of service could result in significant performance degradation. Other applications such as the audio and video delivery, which can tolerate a small extend of packet loss and delay variations, provide the delay-adaptive and rate-adaptive services.

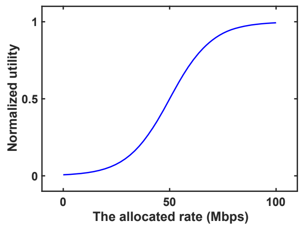
We wish to find a rate-limiting scheme before update which maximizes the total utility, while ensuring congestion-freedom during asynchronous update. Each flow is associated with a utility function that captures how the utility varies with the allocated bandwidth. This can be naturally formulated by an optimization program. Moreover, the problem of finding an update plan that maximizes the total utility is more general than the prior work [7], [2], [8]. Existing work can be viewed as a special case that each utility function is identical.



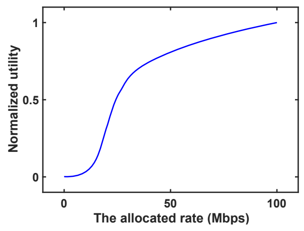
(a) Utility function of flows for a elastic service. The utility begins to decrease when the allocated rate is less than 100 Mbps.



(b) Utility function of flows for a hard real-time service. The utility drops to zero when the allocated rate is less than 50 Mbps.

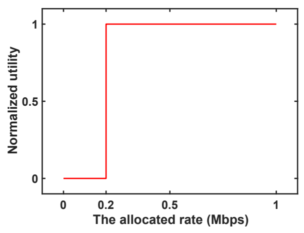


(c) Utility function of flows for a delay-adaptive service. The utility begins to decrease when the allocated rate is less than 100 Mbps.

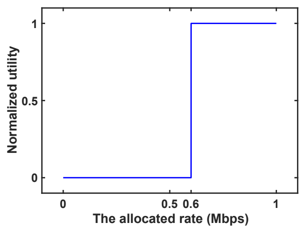


(d) Utility function of flows for a rate-adaptive service. The utility begins to decrease when the allocated rate is less than 100 Mbps.

Fig. 2. Utility functions for four typical services.



(a) Normalized utility for flow A



(b) Normalized utility for flow B

Fig. 3. The different utility functions of flows for two hard real-time services shown in Fig. 1.

C. A Motivating Example

In a software defined data center network, whenever the topology or traffic matrix changes, the controller needs to recalculate routing in order to optimize performance. Consider the example in Fig. 1, in which there are four switches $R_1, \ldots, R_4$, and the link capacity is one unit. F_A and F_B are two flows from R_1 to R_3 and R_4 to R_3, respectively, whose demand are both one unit. The initial routing is illustrated in Fig. 1(a). At this point, suppose a new flow appears from R_4 to R_2 with a demand of one unit. The controller then wants to change routing to Fig. 1(b). Due to the different update order, the two flows may be routed temporarily as in Fig. 1(c) (or Fig. 1(d)) during the transition. In this case congestion occurs at the link from R_4 to R_3 (or the link from R_2 to R_3), which is overloaded with twice its capacity, and results in severe packet loss.

Reducing the flow rate before update can avoid transient congestion [8] during the asynchronous update. If each flow halves its original rate as shown in Fig. 1(e), the demand of each flow becomes 0.5 unit. The maximum link load in state (c) or (d) is only one unit, which cannot beyond the link capacity, and the whole process is congestion-free. However, the utility of rate-reduced flows is ignored and may result in poor performance especially for the hard real-time services. We assume the utility functions of two flows are illustrated as in the Fig. 3(a) and Fig. 3(b), which mathematically formulates the relation between utility and the allocated flow rate. If we simply halve the rate of each flow to avoid the transient congestion during update, the total utility is $f_A(0.5) + f_B(0.5) = 1.0 + 0 = 1.0$. Note that if we reduce the rate of flow A and flow B to 0.3 and 0.7, respectively, as shown in Fig. 1(f), the total utility is $f_A(0.3) + f_B(0.7) = 1.0 + 1.0 = 2.0$, which increases the utility by 100% and the whole process is congestion-free as well.

III. An Optimization Framework

A. Network Model

TABLE I
Key notations in this paper.

Input	F_l	The set of flows for the elastic services
	F_h	The set of flows for the hard real-time services
	F_d	The set of flows for the delay-adaptive services
	F_r	The set of flows for the rate-adaptive services
	F	The set of all flows i. $F = F_l \cup F_h \cup F_d \cup F_r$
	V	The set of switches v
	E	The set of links e
	G	The directed network graph $G = (V, E)$
	C_e	The capacity of link e
	$P(i)$	The set of initial and final path for flow i
	d_i	The demand of flow i
	$l_i(\cdot)$	The utility function of flows for the elastic services
	$h_i(\cdot)$	The utility function of flows for the hard real-time services
	$g_i(\cdot)$	The utility function of flows for the delay-adaptive services
	$f_i(\cdot)$	The utility function of flows for the rate-adaptive services
	$\hat{h}_i(\cdot)$	The concave envelop of the function $h_i(\cdot)$
	$\hat{g}_i(\cdot)$	The concave envelop of the function $g_i(\cdot)$
	$\hat{f}_i(\cdot)$	The concave envelop of the function $f_i(\cdot)$
	$y^1_{i,p}$	The indicator variable of initial route that equals 1 if flow i uses path p and 0 otherwise
	$y^2_{i,p}$	The indicator variable of final route that equals 1 if flow i uses path p and 0 otherwise
Output	x_i	The allocated rate (bandwidth) for flow i

Before presenting the problem definitions, we first discuss our network model. A network is a directed graph $G = (V, E)$, where V is the set of switches and E the set of links with capacities C_e for each link $e \in E$. F represents the set of flows in the network and each flow $i \in F$ is associated with a demand d_i. The set of flows F includes four types of flows F_l, F_h, F_d and F_r, and corresponding utility function is $l_i(\cdot)$, $h_i(\cdot)$, $g_i(\cdot)$ and $f_i(\cdot)$, respectively. Each flow i is routed through a initial path, and will be moved to a final path. The initial and final routing configurations for flow i are denoted by $y^1_{i,p}$ and $y^2_{i,p}$. Before update, we calculate the rate x_i for each flow i, aiming to maximize the sum of utility, such that the congestion-free condition is ensured during the asynchronous network update. It should be noted that the flow in our model is in fact an aggregate of all flows belong to the same service between the ingress-egress switch pair. This does not lose generality of the model, and is in line with previous work such as SWAN [2] and B4 [1]. For convenience, we summarize important notations in Table I.

B. Problem Formulation

Based on the above network model, we formulate maximizing utility network update problem (MUP) as an optimization program (1). Given the initial routing configuration $\{y^1_{i,p}\}$ and final routing configuration $\{y^2_{i,p}\}$, we wish to find an optimal rate-limiting scheme before update which maximizes the total utility, while ensuring congestion-freedom condition during update.

$$\text{maximize} \sum_{i\in F_l} l_i(x_i) + \sum_{i\in F_h} h_i(x_i) + \sum_{i\in F_d} g_i(x_i) + \sum_{i\in F_r} f_i(x_i) \tag{1}$$

$$\text{subject to} \quad \sum_{i\in F} x_i \sum_{p\in P(i):e\in p} \max(y^1_{i,p}, y^2_{i,p}) \le C_e, \quad \forall e \in E, \tag{1a}$$

$$0 \le x_i \le d_i, \quad \forall i \in F. \tag{1b}$$

The objective of program (1) is to maximize the sum of utility for each flow i. The optimization variables x_i indicate the allocated bandwidth for flow i before update. Constraint (1a) characterizes transient congestion for individual links e during transition. For example, as illustrated in Fig. 1, during the transition from Fig. 1(a) to Fig. 1(b), *i.e.*, from initial route to final route, the maximum load on the link $\langle R_4, R_3\rangle$ is $x_{f_A} \times \max(1,0) + x_{f_B} \times \max(0,1) = x_{f_A} + x_{f_B}$, which describes the case shown in Fig. 1(c). Similarly, the maximum load on the link $\langle R_2, R_3\rangle$ is $x_{f_A} \times \max(0,1) + x_{f_B} \times \max(1,0) = x_{f_A} + x_{f_B}$, which describes the case shown in Fig. 1(d). Constraint (1b) is the flow demand conservation constraint, which represents the reduced rate x_i cannot beyond the original flow demand d_i. Here we slightly abuse the notation and use the same index variable i to refer to the set of flows F and each type of flows F_l, F_h, F_d and F_r for convenience. The meaning of i is clear given the context in the formulation.

C. Complexity Analysis

We first establish the hardness of our problem MUP below.

Theorem III.1. *MUP is NP-hard, even for a network consisting of two switches and two parallel links.*

Proof. Here we only give an intuition. Consider a special case of MUP as shown in Fig. 4. The capacity of link e_0 and e_1 is C. There are $\frac{n}{2}$ flows initially routed from link e_0, and will be moved into their final link e_1. The flow demand and the utility function is d_i and $h_i(\cdot)$ respectively, where $\sum_i d_i = C$, $h_i(\cdot)$ represents the utility function of flows for the hard real-time services, $i \in \{1, 2, \cdots, n/2\}$. Similarly, there are $\frac{n}{2}$ flows initially routed from link e_1, and will be moved into their final link e_0. The flow demand and the utility function is d_i and $h_i(\cdot)$ as well, where $i \in \{n/2+1, n/2+2, \cdots, n\}$. The utility function of each hard real-time flow i in our instance has one critical point r_i and is defined as follows.

$$h_i(x_i) = \begin{cases} 1 & x_i > r_i \\ 0 & 0 \le x_i \le r_i \end{cases}$$

where $r_i + \epsilon = d_i$, $i \in \{1, 2, \cdots, n\}$, ϵ is an arbitrary small number. The function $h_i(\cdot)$ indicates that the utility for flow i drops to zero if we reduce the flow rate (we assume that the reduced flow rate is larger than ϵ). Otherwise, the utility is one.

We construct a polynomial reduction from the set partition problem [23] to it. Consider a partition instance consisting of n items, each with a value a_i. Each item i corresponds to one of the flows in the example of Fig. 4, where $a_i = d_i$, $i \in \{1, 2, \cdots, n\}$. Therefore, any feasible partition of the items corresponds to the set of flows with and without rate reduction, and vice versa. The set of rate-reduced flows forms one set of the partition, and that keeping the original rate forms the other. $\square$

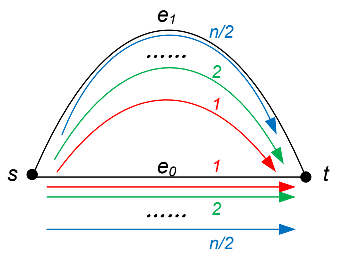


Fig. 4. Reduction from Partition to MUP.

Now we analyze the complexity of program (1). Although the max function is used in constraint (1a), this is still linear since $y^1_{i,p}$ and $y^2_{i,p}$ are both known constant. The constraint (1a) and (1b) with $|E| + |F|$ linear constraints construct a convex polytope. The function $l_i(\cdot)$ is concave, while the functions $h_i(\cdot)$, $g_i(\cdot)$ and $f_i(\cdot)$ are non-concave as there exist two critical points $x_{i,1}$ and $x_{i,2}$ in the x-axis such that for $\forall \alpha \in (0, 1)$, the following condition holds [24],

$$\mathscr{A}_i(\alpha \cdot x_{i,0} + (1-\alpha)\cdot x_{i,1}) < \alpha \cdot \mathscr{A}_i(x_{i,0}) + (1-\alpha)\cdot \mathscr{A}_i(x_{i,1})$$

where $\mathscr{A}_i$ represents the set of functions, $\mathscr{A}_i \in \{h_i(\cdot), g_i(\cdot), f_i(\cdot)\}$. Hence, the sum of the objective function is non-concave as well and we have the theorem below.

Theorem III.2. *The program (1) is non-concave.*

IV. Algorithms

In this section we develop a set of algorithms to tackle MUP. We first propose an iterative improvement algorithm to calculate the reduced flow rate before update. However, the performance of this approach heavily relies on the input step length. In addition, we use convex relaxation techniques to determine a concave envelop for each non-concave utility function and obtain an initial solution using standard methods for nonlinear convex optimization. Further we adjust the allocated rate which improves upon the initial solution by greedily increasing the total utility in the network.

A. An iterative improvement algorithm

The main intuition of our iterative improvement algorithm is explained as follows. At the beginning, each flow's rate is its original rate d_i. In order to avoid transient congestion during update, we first reduce the rate of flows passing through the most congested links. We iteratively decrease the flow rate with the least utility reduction relative to an input parameter Δ, until all the links are congestion-free.

Algorithm 1: Iterative improvement

```
Input: The directed acyclic network $G$; the initial route $y^1_{i,p}$ and
    the final route $y^2_{i,p}$ for each flow $i \in F$; the utility functions
    $l_i(\cdot)$, $h_i(\cdot)$, $g_i(\cdot)$ and $f_i(\cdot)$ for the elastic service, the hard
    real-time service, delay-adaptive service and rate-adaptive
    service; the step length $\Delta$.
Output: The allocated rate $\{x_i\}$.
1: $L_e = \sum_{i\in F} d_i \sum_{p\in P(i):e\in p} \max(y^1_{i,p}, y^2_{i,p}), \forall e \in E$
2: $\delta_e = L_e - C_e, \forall e \in E$
3: $x_i = d_i, \forall i \in F$
4: while there exists a link $e$ such that $\delta_e > 0$ do
5:     $e^* = \arg\max_{e\in E} \delta_e$
6:     $\delta = \delta_{e^*}$
7:     Construct the flow set $F^*$ that is routed through link $e^*$
8:     while $\delta > 0$ do
9:         $i' = \arg\min_{i\in F^*} \frac{\mathscr{A}_i(x_i)-\mathscr{A}_i(x_i-\Delta)}{\Delta}$, where
           $\mathscr{A}_i \in \{l_i(\cdot), h_i(\cdot), g_i(\cdot), f_i(\cdot)\}$
10:        $x_{i'} = x_{i'} - \min(\delta, \Delta)$
11:        $\delta = \delta - \min(\delta, \Delta)$
12:    Update $L_e$ and $\delta_e$
```

The procedure of determining the rate of each flow is shown in Algorithm 1. We first calculate the possible maximum link load L_e during asynchronous update (line 1). Let δ_e be the difference between the link load L_e and link capacity C_e (line 2). The inequation $\delta_e > 0$ indicates that the congestion happens as the link load is beyond its link capacity. We iteratively reduce the flow rate in the most congested links e^* until the condition $\delta_e \leq 0$ is satisfied (lines 4-12). In each iteration, we firstly calculate δ_e for each link in order to determine the most congested link e^*. And then we construct the set of flows F^* in which the flows are routed through the link e^* (lines 5-7). We calculate the variation of utility function relative to Δ and pick flow i' with the least utility reduction (line 9). Accordingly we decrease the rate of flow i' by Δ and the parameter Δ should be subtracted from δ as well (lines 10-11). The $\min(\delta, \Delta)$ used here is to ensure that δ cannot be less than zero after rate reduction. Once δ equals zero, we update L_e and δ_e and the algorithm enters into the next loop (line 12).

B. A rate adjustment algorithm

As the iterative improvement algorithm relies on the input step length Δ, we focus on using convex relaxation techniques to tackle our problem. We explain the high level working of our idea. Since the utility function in the objective is separable, we can replace each non-concave function with a concave function and then solve this new program using standard methods for nonlinear convex optimization. Based on this initial solution, we first reduce the rate for hard real-time services as its utility keeps unchanged once the allocated rate is beyond a critical point r_i. Next we increase the rate for the flows with maximum utility increment and the least bandwidth consumption. Finally we update the initial solution and complete the rate adjustment procedure.

Before illustrating our rate adjustment algorithm, we first introduce the related definition first.

Definition IV.1. Concave envelop [24]: *For each utility function $\mathscr{A}_i(x_i)$, the concave envelop $\hat{\mathscr{A}_i}(\cdot)$ is a concave function defined as following.*

$$\hat{\mathscr{A}_i}(\cdot) = \inf\{\mathscr{B}_i(\cdot)|\mathscr{B}_i(\cdot) \textit{ is concave and } \mathscr{B}_i(x_i) \geq \mathscr{A}_i(x_i)\}$$

where $\mathscr{A}_i(\cdot) \in \{h_i(\cdot), g_i(\cdot), f_i(\cdot)\}$ in our problem.

As the functions $h_i(x_i)$, $g_i(x_i)$ and $f_i(x_i)$ in the objective are non-concave, we plan to relax the program (2) by replacing $h_i(x_i)$, $g_i(x_i)$ and $f_i(x_i)$ with their concave envelops $\hat{h}_i(x_i)$, $\hat{g}_i(x_i)$ and $\hat{f}_i(x_i)$. Note that calculating the concave envelop for arbitrary functions could be hard. Here we only consider the special case that is hard real-time service, delay-adaptive and rate-adaptive service respectively, whose concave envelops are shown in Fig. 5. When the concave envelop for each non-concave function is determined, the program (2) can be reformulated as the following program by convex relaxation techniques.

$$\text{maximize} \sum_{i\in F_l} l_i(x_i) + \sum_{i\in F_h} \hat{h}_i(x_i) + \sum_{i\in F_d} \hat{g}_i(x_i) + \sum_{i\in F_r} \hat{f}_i(x_i) \tag{2}$$

$$\text{subject to} \qquad (1a), (1b).$$

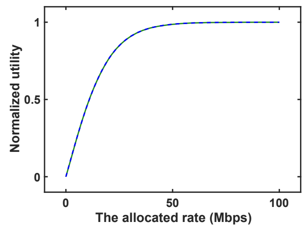
Let $\mathscr{L}(x, \lambda, u, w)$ be the Lagrangian of program (2) with dual variables λ, u and w.

$$\mathscr{L}(x, \lambda, u, w) = \sum_{i\in F_l} l_i(x_i) + \sum_{i\in F_h} \hat{h}_i(x_i) + \sum_{i\in F_d} \hat{g}_i(x_i) + \sum_{i\in F_r} \hat{f}_i(x_i) + \lambda^T \left(C_e - \sum_{i\in F} x_i \sum_{p\in P(i):e\in p} \max(y^1_{i,p}, y^2_{i,p}) \right) + u^T \cdot (d_i - x_i) + w^T \cdot x_i$$

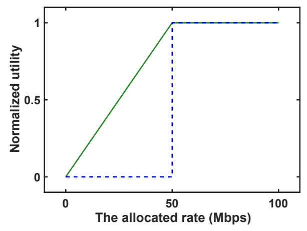
The Lagrange dual program of primal program (2) is defined as the following.

$$\text{minimize} \quad \sup_x \mathscr{L}(x, \lambda, u, w) \tag{3}$$

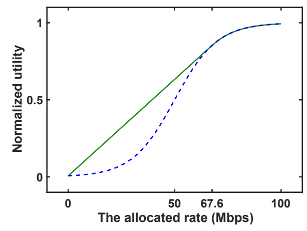
$$\text{subject to} \quad \lambda \geq 0, \quad u \geq 0, \quad w \geq 0. \tag{3a}$$



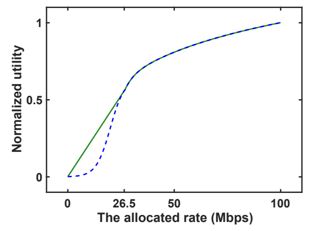
(a) The concave envelop of utility function for the elastic service.



(b) The concave envelop of utility function for the hard real-time service.



(c) The concave envelop of utility function for the delay-adaptive service.



(d) The concave envelop of utility function for the rate-adaptive service.

Fig. 5. The concave envelops (colored green) of utility functions for four typical services.

From the Theorem 2 in [25], we can derive that Corollary IV.1 can be established, which indicates the solution of the concave relaxation program (2) is bounded by the results below.

Corollary IV.1. *Let $w \in \mathscr{R}^{|F|}$ be a random variable with uniform distribution on the unit sphere. We construct the following auxiliary optimization program,*

$$\textit{minimize} \quad w^T \cdot x \tag{4}$$

$$\textit{subject to} \quad \text{(1a)},$$

$$\sum_{i\in F_h}\left(\hat{h}_i(x_i)-\hat{h}_i(x_i^*)\right)+\sum_{i\in F_d}\left(\hat{g}_i(x_i)-\hat{g}_i(x_i^*)\right)$$
$$+\sum_{i\in F_r}\left(\hat{f}_i(x_i)-\hat{f}_i(x_i^*)\right) \quad \geq$$
$$\lambda^{*T}\left(\sum_{i\in F}(-x_i+x_i^*)\sum_{p\in P(i):e\in p}\max(y_{i,p}^1,y_{i,p}^2)\right)$$
$$+u^{*T}\cdot(-x_i+x_i^*)+w^{*T}\cdot(x_i-x_i^*). \tag{4a}$$

We assume $(x^*, \lambda^*, u^*, w^*)$ is an optimal primal-dual tuple of program (2) and (3). Then with probability 1, $\mathscr{R}$ has a unique solution $\tilde{x}$ that satisfies the following condition.

$$OPT-\left(\sum_{i\in F_l}l_i(\tilde{x}_i)+\sum_{i\in F_h}\hat{h}_i(\tilde{x}_i)+\sum_{i\in F_d}\hat{g}_i(\tilde{x}_i)+\sum_{i\in F_r}\hat{f}_i(\tilde{x}_i)\right)$$
$$\leq \sum_{i=1}^{\min\{|F|,|E|+|F|\}} \rho_i$$

where OPT represents the optimal solution of program (1) and the definition of ρ_i is as following.

Definition IV.2. Nonconcavity of a function [25]: *We define the nonconcavity ρ_i as the equation below.*

$$\rho_i = \sup_{x_i}\{\hat{\mathscr{A}}_i(\cdot)-\mathscr{A}_i(\cdot)\}$$

where $\rho_1 \geq \rho_2 \geq \cdots \geq \rho_{\min\{|F|,|E|+|F|\}}$.

Based on the above discussion, we propose our rate adjustment algorithm. The complete procedure is shown in Algorithm 3. By determining the concave envelop for each non-concave function, we first obtain the initial solution $\{x_i\}$ using standard methods for nonlinear convex optimization. However, solving this program is time-consuming especially in large-scale production networks with thousands of switches and flows. Thus we set out to design a parallel algorithm to solve this program instead.

```
Algorithm 2: A Proximal Jacobian ADMM Algorithm
Input: The directed acyclic network $G$; the initial route $y_{i,p}^1$ and
    the final route $y_{i,p}^2$ for each flow $i \in F$; the utility functions
    and its concave envelops $l_i(\cdot)$, $\hat{h}_i(\cdot)$, $\hat{g}_i(\cdot)$ and $\hat{f}_i(\cdot)$ for the
    elastic service, the hard real-time service, delay-adaptive
    service and rate-adaptive service.
Output: A solution $\{x_i\}$.
1: Transform the program (2) to program (5).
2: Initialize the variables $\boldsymbol{\delta}$, $\boldsymbol{\gamma}$, $\boldsymbol{x}$ and multipliers $\boldsymbol{\theta}$, $\boldsymbol{\varphi}$ to zero.
3: for $t = 1, 2, \cdots$ do
4:     Update $\boldsymbol{\delta}$, $\boldsymbol{\gamma}$, $\boldsymbol{x}$ from programs (6), (7), (8) in parallel.
5:     Update $\boldsymbol{\theta}$, $\boldsymbol{\varphi}$ from equations (9), (10).
```

Inspired by the framework of multiple-block ADMM [26], we develop a proximal Jacobian ADMM algorithm that can converge to an optimal solution at the rate of $o(\frac{1}{t})$ in Algorithm 2, where t is the number of iteration times. As the constraints in program (2) are inequalities, we require to transform program (2) to program (5) in order to apply 3-block ADMM (line 1). Furthermore, we initialize the variables $\boldsymbol{\delta}$, $\boldsymbol{\gamma}$ $\boldsymbol{x}$, and multipliers $\boldsymbol{\theta}$, $\boldsymbol{\varphi}$ to zero (line 2) and solve each subprogram in parallel (lines 3-5).

Now we explain the detailed transformation procedure. For convenience, we denote Δ and Φ as following.

$$\Delta = \sum_{p\in P(i):e\in p}\max(y_{i,p}^1,y_{i,p}^2)$$

$$\Phi = \sum_{i\in F_l}l_i(x_i)+\sum_{i\in F_h}\hat{h}_i(x_i)+\sum_{i\in F_d}\hat{g}_i(x_i)+\sum_{i\in F_r}\hat{f}_i(x_i)$$

We reformulate the program (2) in order to apply ADMM. We first introduce slack variables δ_e and γ_i to transform the inequality constraints (1a) and (1b) to equality constraints (5a) and (5b) required by ADMM. Second, all variables in the constraints are separable for each group and comply with the condition of ADMM. Towards this end, we add the original constraint $x_i \geq 0$ and reformulate the program (2) to program (5) as follows.

$$\text{maximize} \quad \Phi \tag{5}$$
$$\text{subject to} \quad C_e - \sum_{i \in F} x_i \cdot \Delta - \delta_e = 0, \quad \forall e \in E, \tag{5a}$$
$$d_i - x_i - \gamma_i = 0, \quad \forall i \in F, \tag{5b}$$
$$x_i \geq 0, \quad \forall i \in F. \tag{5c}$$

Let $\mathscr{L}_\rho$ be the augmented Lagrangian of program (5) with dual variables θ and φ. i.e., introducing an extra $\mathscr{L}$-2 norm term into the objective:

$$\mathscr{L}_\rho = \Phi + \sum_{e \in E} \theta_e \left(C_e - \sum_{i \in F} x_i \cdot \Delta - \delta_e \right) + \sum_{i \in F} \varphi_i (d_i - x_i - \gamma_i) + \frac{\rho}{2} \sum_{e \in E} \left(C_e - \sum_{i \in F} x_i \cdot \Delta - \delta_e \right)^2 + \frac{\rho}{2} \sum_{i \in F} \left(d_i - x_i - \gamma_i \right)^2$$

where $\rho > 0$ is the penalty parameter. The reason that we introduce the penalty term is to speed up the convergence rate [27].

Distributed 3-block ADMM. We initialize the variables $\boldsymbol{\delta}$, $\boldsymbol{\gamma}$, $\boldsymbol{x}$ and multipliers $\boldsymbol{\theta}$, $\boldsymbol{\varphi}$ to zero. For $t = 1, 2, \cdots$, repeat the following steps.

1. δ-update. Each link e solves the following subproblem for obtaining δ_e^{t+1}:

$$\text{maximize} \quad \theta_e^t \cdot \delta_e + \frac{\rho}{2} \left(C_e - \sum_{i \in F} x_i^t \cdot \Delta - \delta_e \right)^2 + \frac{w}{2} \left(\delta_e - \delta_e^t \right)^2 \tag{6}$$
$$\text{subject to} \quad \delta_e \geq 0, \quad \forall e \in E. \tag{6a}$$

This per-link subproblem is a small-scale quadratic program and can be solved efficiently.

2. γ-update. Each flow i solves the following subproblem for obtaining γ_i^{t+1}:

$$\text{maximize} \quad \varphi_i^t \cdot \gamma_i + \frac{\rho}{2} \left(d_i - x_i^t - \gamma_i \right)^2 + \frac{w}{2} \left(\gamma_i - \gamma_i^t \right)^2 \tag{7}$$
$$\text{subject to} \quad \gamma_i \geq 0, \quad \forall i \in F. \tag{7a}$$

This per-flow subproblem can be solved by the standard solvers for quadratic program.

3. x-update. Each flow i solves the following subproblem for obtaining x_i^{t+1}:

$$\text{maximize} \quad \Phi - \sum_{e \in E} \theta_e^t \cdot x_i \cdot \Delta + \frac{\rho}{2} \left(d_i - x_i - \gamma_i^t \right)^2 - \varphi_i^t \cdot x_i + \frac{\rho}{2} \sum_{e \in E} \left(C_e - \sum_{i \in F} x_i \cdot \Delta - \delta_e^t \right)^2 + \frac{w}{2} \left(x_i - x_i^t \right)^2 \tag{8}$$
$$\text{subject to} \quad x_i \geq 0, \quad \forall i \in F. \tag{8a}$$

4. Dual updates. Each link e updates θ for the constraint (5a):

$$\theta_e^{t+1} = \theta_e^t + \iota \cdot \rho \cdot \left(C_e - \sum_{i \in F} x_i^{t+1} \cdot \Delta - \delta_e^{t+1} \right) \tag{9}$$

Each flow i updates φ for the constraint (5b):

$$\varphi_i^{t+1} = \varphi_i^t + \iota \cdot \rho \cdot \left(d_i - x_i^{t+1} - \gamma_i^{t+1} \right) \tag{10}$$

where $\iota \cdot \rho$ is the step size for the dual update.

Algorithm 3: Rate adjustment

```
Input: The directed acyclic network $G$; the initial route $y_{i,p}^1$ and
    the final route $y_{i,p}^2$ for each flow $i \in F$; the utility functions
    $l_i(\cdot)$, $h_i(\cdot)$, $g_i(\cdot)$ and $f_i(\cdot)$ for the elastic service, the hard
    real-time service, delay-adaptive service and rate-adaptive
    service.
Output: The allocated rate $\{\hat{x}_i\}$.
 1: for each $i \in F \setminus F_l$ do
 2:     Determining the concave envelop $\hat{h}_i(\cdot)$, $\hat{g}_i(\cdot)$ or $\hat{f}_i(\cdot)$
        according to Definition IV.1
 3: Obtain $\{x_i\}$ using Algorithm 2.
 4: for each $i \in F_h$ do
 5:     if $l_i(x_i) = 1$ then
 6:         $x_i = r_i + \epsilon$
 7:         $\hat{x}_i = x_i$
 8: for each $i \in F$ do
 9:     if the rate of the flow $i$ can be increased at most $\xi_i$ without
        link capacity violation then
10:         $F^* = F^* \cup \{i\}$
11: while $F^* \neq \emptyset$ do
12:     $i' = \arg\max_{i \in F^*} \frac{\mathscr{A}_i(x_i+\xi) - \mathscr{A}_i(x_i)}{\xi_i}$, where
        $\mathscr{A}_i \in \{l_i(\cdot), h_i(\cdot), g_i(\cdot), f_i(\cdot)\}$
13:     $\hat{x}_{i'} = x_{i'} + \xi_{i'}$
14:     Re-construct the flow set $F^*$
```

Note that Algorithm 2 can converge to an optimal solution at the rate of $o(\frac{1}{t})$, which can significantly speed up the calculation procedure. After obtaining the solution $\{x_i\}$ using Algorithm 2, we adjust the rate of each flow which improves upon the initial solution by greedily increasing the total utility (lines 4-13). Specifically, for each hard real-time flows i, we determine the critical point r_i of its utility function, where $\hat{x}_i = r_i + \epsilon$, which indicates that the rate of flow i can be decreased by $x_i - \hat{x}_i$ without utility reduction (lines 5-7). After that, we construct the flow set F^* in which the rate of each flow i can be independently increased at most ξ_i without the capacity violation (lines 8-10). Next we pick the flow i' with the maximum utility increment relative to ξ_i (lines 12). Accordingly, we increase the flow rate by $\xi_{i'}$ and re-construct the flow set F^* (lines 13-14). Note that the initial solution is bounded by the results from Corollary IV.1. Algorithm 3 improves upon the initial solution whenever possible, and thus its performance is at least as good as that of initial solution.

V. Experimental Evaluation

We evaluate our algorithms using both prototype implementation and large-scale simulation.

Benchmark Schemes: We compare the following schemes with our algorithms.

TABLE II
THE USED UTILITY FUNCTIONS IN OUR EVALUATION.

Service type	Utility function	Parameter settings
The elastic service	$l_i(x_i) = \frac{2}{1+e^{-\theta\cdot(x_i-\beta)}} - 1$	$\theta \in [0.1, 0.2], \beta \in [-10, 0]$
The hard real-time service	$h_i(x_i) = \begin{cases} 1 & r_i < x_i \leq 100 \\ 0 & 0 \leq x_i \leq r_i \end{cases}$	$r_i \in (0, 100)$
The delay-adaptive service	$g_i(x_i) = \frac{1}{1+e^{-\theta\cdot(x_i-\beta)}}$	$\theta \in [0.1, 0.5], \beta \in [40, 60]$
The rate-adaptive service	$f_i(x_i) = \begin{cases} \frac{1}{1+e^{-\theta\cdot(x_i-\beta)}} & 0 \leq x_i \leq r_i \\ \frac{1}{1+e^{-\theta\cdot(x_i-\beta)}} + \log_{10}(\frac{x_i}{r_i}) & r_i < x_i \leq 100 \end{cases}$	$\theta \in [0.2, 0.4], \beta = r_i - 10, r_i \in [20, 80]$

- **One Shot**: Each flow's rate is evenly decreased on the congested links during the transition directly from the initial to the final routing configurations.
- **I**^{2}: Our heuristic iterative improvement algorithm shown in Algorithm 1. Unless stated otherwise, we configure the parameter Δ to 1.0 in Algorithm 1.
- **SWAN**: State-of-the-art update algorithm [2]. As discussed in Sec. I, this algorithm cannot take utility into consideration, and we only include it for comparing the control overhead and update time. We leave 10% link capacity vacant to guarantee the congestion-free update plan always exists.
- **Hermes**: Our rate adjustment algorithm shown in Algorithm 3. Unless stated otherwise, we configure the parameter ϵ to 1.0 in Algorithm 3.

The traffic used in our evaluation is generated in [28], and we change the flow demand to simulate traffic variations. Each flow is associated with a utility function shown in Tab. II Given the demand, we calculate the initial and final routing to maximize the network utility. Unless stated otherwise, we configure ρ, w and ι to be 0.1, 0.02 and 1.0 respectively in Algorithm 2 as suggested in [26].

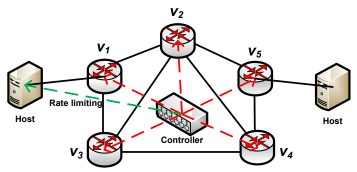


Fig. 6. Network topology used in our testbed.

A. Implementation and Testbed Emulations

Implementation: We develop a prototype of our algorithms using OFSoftSwitch and Dpctl [12] as Openflow switches and the controller. Now we describe how to perform network update in Hermes. As illustrated in Fig. 7, the initial time is t_0. We first obtain a solution to MUP using Algorithm 3 at t_1 (the time for generating update plan is $t_1 - t_0$). Based on this solution, the controller generates a specific rate-limiting policy for each host. The hosts configure Linux `tc class` and `filter` property to reduce the sending rate once they receive their own rate-limiting policy from the controller. When the flow rate is successfully reduced at t_2, we start to update the forwarding rules (the rate-limiting time is $t_2 - t_1$). Hermes relies on two-phase update protocols [29] to perform network update. We use `VLAN ID` in packet headers to index stages. In the first phase, new rules—whose matching fields use the new `VLAN ID` that corresponds to the second stage—are added. During this phase, flows are still forwarded according to existing rules as packets are still stamped with the `VLAN ID` of the first stage. Once the update is done for all switches, the protocol enters the second phase, when we stamp every incoming packet with the new `VLAN ID`. At this point the new rules become functional, and old rules are removed by the controller (the update time is $t_3 - t_2$).

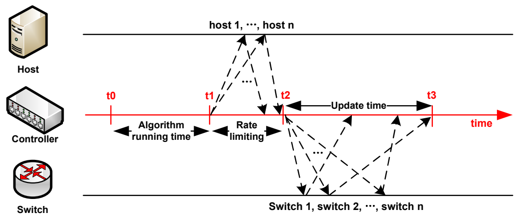


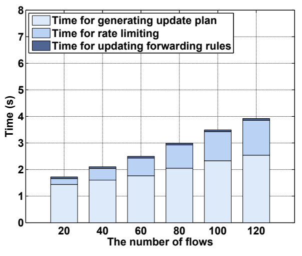
Fig. 7. The whole network update procedure in Hermes.

For completeness we now explain the implementation of SWAN now. SWAN solves a series of LPs to find a congestion-free multi-stage update plan. The update plan consists of discrete stages, each of which moves a portion of flows to a temporary path to avoid transient congestion. At each stage, SWAN requires to re-configure the splitting weights at ingress switch based on the calculation results of LP. Accordingly, the packets at ingress switch are splitted to a set of pre-defined output ports proportional to the corresponding weights. We use Openflow `group table` with `select` type [30] to implement this function. The Openflow messages used in our experiments are shown in Table III.

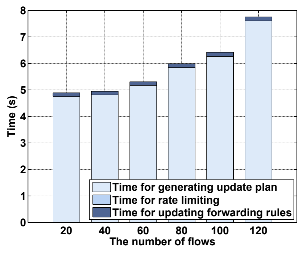
TABLE III
OPENFLOW MESSAGES USED IN OUR EXPERIMENTS.

Openflow message	Type
OFPT_FLOW_MOD	Controller-to-Switch
OFPT_GROUP_MOD	Controller-to-Switch
OFPT_BUNDLE_CONTROL	Controller-to-Switch
OFPT_PACKET_IN	Asynchronous
OFPT_PACKET_OUT	Asynchronous

Setup: Our experiments are performed over 5-server testbed, equipped with two intel E5-2650 CPUs with 12 cores and 64 GB memory. Each server runs a software-based Openflow switch [12]. We adopt a small scale topology with 5 switches and 7 1Gbps links as illustrated in Fig. 6. We

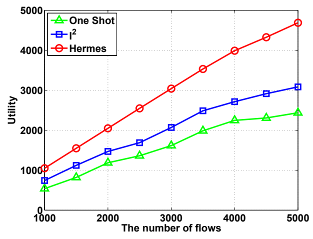


(a) Hermes

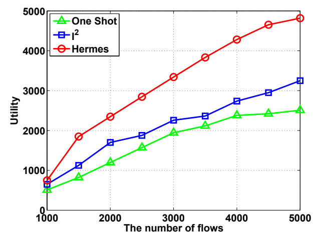


(b) SWAN

Fig. 8. The total time in our testbed experiments.



(a) The number of utility functions for each type is identical.



(b) The number of utility functions for each type is random.

Fig. 9. Utility variation with different numbers of flows.

use `pktgen` to generate different numbers of UDP flows in each run. The aggregate flow rate is 1 Gbps in each port. The forwarding rules are installed and updated via Dpctl API [12]. We use `IP_PROTO`, `UDP_SRC`, `UDP_DST` and `VLAN_ID` as the matching field to perform routing forwarding. Note that we should set `IP_PROTO` field to be 17 in order to match UDP packets.

Experiment Results: Fig. 8 shows the total time results for Hermes and SWAN. In this set of experiments, we vary the number of flows from 20 to 120 at the increment of 20. We perform the same experiment with ten runs for both Hermes and SWAN, and report the average values. We evaluate the total time T. It includes three parts: time for generating update plan T_g (running time of algorithm), time for rate limiting T_r and time for updating the forwarding rules T_u.

$$T = T_g + T_r + T_u$$

Above of all, we discuss the first part T_g. SWAN introduces around 2∼4 intermediate stages in our experiments and the total running time is the sum of the running time solving each LP for each stage. We can observe that the running time for SWAN increases more significantly. The difference between Hermes and SWAN is minimum when the number of flows is 20, while the difference between them is maximum when the number of flows is 120. On average, Hermes can reduce the running time by 60% compared with SWAN. The second part T_r is the time for reducing the flow rate. The rate-limiting time for SWAN is zero in our setting as we leave 10% link capacity vacant in advance to guarantee its congestion-free update plan always exists. The rate-limiting time for Hermes is 0.2s and 1.3s when the number of flows is 20 and 120 respectively. Finally, we measure the update time T_u. As Openflow `barrier` feature cannot provide accurate acknowledgments [31] to indicate the completion of update operation, we use `tcpdump`—a powerful packet analyzer—to confirm when the new rules take effect. Looking more closely into Fig. 8(a) and Fig. 8(b), the improvement for Hermes is more obvious: it decreases the time for updating forwarding rules by 35% from SWAN. This demonstrates that Hermes in general leads to less update time as it does not require additional update operations involved in multi-stage update schemes. In summary, Hermes can reduce the total time T by 55% compared with SWAN on average. Meanwhile, during the experiments, we found that the time for generating update plan T_g account for around 95% of the total time T. The time for updating forwarding rules T_u just account for around 5%.

B. Simulation

We also conduct extensive simulations to thoroughly evaluate our algorithms at scale.

Setup. In addition to the small-scale topology used in our testbed, here we use a large-scale synthetic scale-free topology that is randomly produced by the `scale_free_graph` function [32]. There are 100 switches and 586 10 Gbps links in total. We generate different numbers of flows for each source-destination switch pair in each run. The utility functions for four typical services are chosen from Tab. II. We run our algorithms on a server with Intel(R) Xeon(R) CPU E5-2650 and 64 GB memory. Each data point is an average of ten runs.

We first investigate the utility variations with different numbers of flows. We set the link capacity to be 6 Gbps and vary the number of flows from 1000 to 5000 at the increment of 500. We can see that, as the number of flows increases, Hermes significantly increases more utility compared to I^2 and One Shot. Specifically, in Fig. 9(b), the total utility for Hermes, I^2 and One Shot is 3341, 2258 and 1944, respectively, when the number of flows is 3000. Looking more closely into Fig. 9(a) and Fig. 9(b), the improvement for Hermes is more significant: it increases the total utility by 49% and 85% from I^2 and One Shot on average. This demonstrates that our algorithms take full advantage of different properties of objective function and significantly increase utility by determining the reduced flow rate with the least utility reduction.

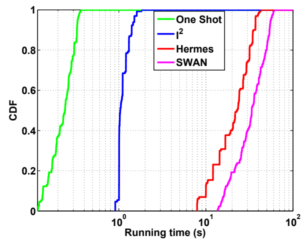


Fig. 10. Running time

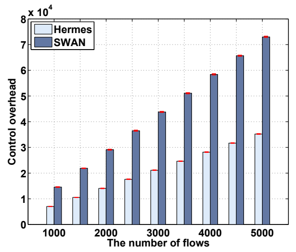


Fig. 11. Control overhead

We evaluate the running time of our algorithms which is illustrated in Fig. 10. We can see that, most updates using Hermes finish within 33 seconds while SWAN takes 54 seconds. The essential reason is that congestion-free update usually introduces more intermediate stages and thus involves more variables in LP, which results in more running time. Even worse, since SWAN has to solve a series of LP to obtain a congestion-free update plan, the total running time is the sum of the running time solving each LP. In contrast, Hermes and I^2 only require one stage during the update and thus save a lot of running time. One Shot, as the lower bound of the running time, is also evaluated in our experiments. It does not respect the network utility and needs the least running time.

We define control overhead as the number of flow table rules that needs to be added, removed or modified during the update. Essentially this measures the number of operations, as well as the number of flow table entries required to perform the update. Fig. 11 shows the control overhead varies with the number of flows. We observe that SWAN introduces more control overhead than Hermes. When the number of flows is 5000, the control overhead of SWAN and Hermes is 72970 and 35207, respectively. SWAN is almost twice as that of Hermes. The reason is that SWAN usually takes multiple stages to avoid congestion when transitioning from initial stage to final stage. Each stage involves a series of update operations. In contrast, Hermes uses only one stage during the update, which guarantees congestion-freedom by reducing the flow rate and saves a lot of update operations.

VI. Conclusion

In this paper, we presented Hermes, a utility-aware network update system, which produces a rate-limiting scheme before update which maximizes the total utility, while ensuring congestion-freedom during asynchronous update. We formulated it as an optimization program, and a series of algorithms are developed. Evaluation results show that Hermes can increase the total utility and reduce the control overhead.

Acknowledgment

The authors would like to thank anonymous reviewers for their valuable comments. This research is supported by the National Key R&D Program of China 2018YFB1003202, the National Natural Science Foundation of China under Grant Numbers 61772265, 61602194, 61672276, 61672353, 61502229, 61472252 and 61321491, the Fundamental Research Funds for the Central Universities under Grant Number 021014380079 and 020214380046, the Collaborative Innovation Center of Novel Software Technology and Industrialization, and the Jiangsu Innovation and Entrepreneurship (Shuangchuang) Program, the Royal Society project IEC/170324, EPSRC IAA project CSA7114, EPSRC project EP/K011693/1 and EU CROWN project PIRSES-GA-2013-610524.

References

[1] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: experience with a globally-deployed software defined wan," in *SIGCOMM*, 2013, pp. 3–14.

[2] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven wan," in *SIGCOMM*, 2013, pp. 15–26.

[3] K. Nagaraj, D. Bharadia, H. Mao, S. Chinchali, M. Alizadeh, and S. Katti, "Numfabric: Fast and flexible bandwidth allocation in datacenters," in *SIGCOMM*, 2016, pp. 188–201.

[4] S. Kandula, I. Menache, R. Schwartz, and S. R. Babbula, "Calendaring for wide area networks," in *SIGCOMM*, 2014, pp. 515–526.

[5] H. Zhang, K. Chen, W. Bai, D. Han, C. Tian, H. Wang, H. Guan, and M. Zhang, "Guaranteeing deadlines for inter-data center transfers," *IEEE/ACM Transactions on Networking*, vol. 25, no. 1, pp. 579–595, 2017.

[6] N. Gvozdiev, B. Karp, and M. Handley, "Fubar: Flow utility based routing," in *HotNets*, 2014, pp. 1–7.

[7] H. H. Liu, X. Wu, M. Zhang, L. Yuan, R. Wattenhofer, and D. A. Maltz, "zupdate: updating data center networks with zero loss," in *SIGCOMM*, 2013, pp. 411–422.

[8] X. Jin, H. H. Liu, X. Wu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford, and R. Wattenhofer, "Dynamic scheduling of network updates," in *SIGCOMM*, 2014, pp. 539–550.

[9] W. Wang, W. He, J. Su, and Y. Chen, "Cupid: Congestion-free consistent data plane update in software defined networks," in *INFOCOM*, 2016, pp. 1–9.

[10] J. Zheng, H. Xu, G. Chen, and H. Dai, "Minimizing transient congestion during network update in data centers," in *ICNP*, 2015, pp. 1–10.

[11] K. He, J. Khalid, A. Gember-Jacobson, S. Das, C. Prakash, A. Akella, L. E. Li, and M. Thottan, "Measuring control plane latency in sdn-enabled switches," in *SOSR*, 2015, pp. 25:1–25:6.

[12] "Cpqd ofsoftswitch," https://github.com/CPqD/ofsoftswitch13.

[13] D. Dolev, A. Zymnis, S. P. Boyd, D. Bickson, and Y. Tock, "Distributed large scale network utility maximization," in *ISIT*, 2009, pp. 829–833.

[14] E. Wei, A. E. Ozdaglar, and A. Jadbabaie, "A distributed newton method for network utility maximization," in *CDC*, 2010, pp. 1816–1821.

[15] D. X. Wei, C. Jin, S. H. Low, and S. Hegde, "FAST TCP: motivation, architecture, algorithms, performance," *IEEE/ACM Transactions on Networking*, vol. 16, no. 6, pp. 1246–1259, 2006.

[16] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pfabric: Minimal near-optimal datacenter transport," in *SIGCOMM*, 2013, pp. 435–446.

[17] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: a centralized "zero-queue" datacenter network," in *SIGCOMM*, 2014, pp. 307–318.

[18] A. Ludwig, J. Marcinkowski, and S. Schmid, "Scheduling loop-free network updates: It's good to relax!" in *PODC*, 2015, pp. 13–22.

[19] A. Ludwig, S. Dudycz, M. Rost, and S. Schmid, "Transiently secure network updates," in *SIGMETRICS*, 2016, pp. 273–284.

[20] Y. Chen, S. Jain, V. K. Adhikari, Z. Zhang, and K. Xu, "A first look at inter-data center traffic characteristics via yahoo! datasets," in *INFOCOM*, 2011, pp. 1620–1628.

[21] S. Shenker, "Fundamental design issues for the future internet," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 7, pp. 1176–1188, 1995.

[22] N. Laoutaris, M. Sirivianos, X. Yang, and P. Rodriguez, "Inter-datacenter bulk transfers with netstitcher," in *SIGCOMM*, 2011, pp. 74–85.

[23] S. Chopra and M. R. Rao, "The partition problem," *Math. Program.*, vol. 59, pp. 87–115, 1993.

[24] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[25] M. Udell and S. P. Boyd, "Bounding duality gap for separable problems with linear constraints," *Computational Optimization and Applications*, vol. 64, no. 2, pp. 355–378, 2016.

[26] W. Deng, M. Lai, Z. Peng, and W. Yin, "Parallel multi-block ADMM with o(1 / k) convergence," *J. Sci. Comput.*, vol. 71, no. 2, pp. 712–736, 2017.

[27] S. P. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[28] "Fnss," http://fnss.github.io/.

[29] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," in *SIGCOMM*, 2012, pp. 323–334.

[30] "Openflow switch specification," https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf.

[31] M. Kuzniar, P. Peresíni, and D. Kostic, "Providing reliable FIB update acknowledgments in SDN," in *CoNEXT*, 2014, pp. 415–422.

[32] "Networkx," https://networkx.github.io/.