Flow-level Traffic Engineering in Conventional Networks with Hop-by-Hop Routing

Nan Geng, Yuan Yang, Mingwei Xu
Department of Computer Science and Technology, Tsinghua University, Beijing, China
Beijing National Research Center for Information Science and Technology(BNRist), Beijing, China
Email:gn16@mails.tsinghua.edu.cn, yangyuan_thu@mail.tsinghua.edu.cn, xumw@tsinghua.edu.cn

***Abstract*—A fine-grained traffic engineering (TE) that enables per-flow control is considered to be necessary in future Internet. In this paper, we study to realize flow-level TE in conventional networks, where hop-by-hop routing is available, and advanced technologies such as SDN and MPLS are not deployed. Based on analysis and modelling on real Internet traffic, we propose to detect and schedule a few large flows in real time, which dominate the traffic amount. The proposed scheme leverages advanced algorithms for detection, computes the rerouting paths in a centralized server, uses extended OSPF to distribute the routing, and uses a few ACL entries for flow-level forwarding. We formalize the link weight assignment-based large flow scheduling problem and prove that the problem is NP-hard. We develop algorithms to compute the routing and reduce extra LSA number required. We present a set of theoretical results on the TE performance bounds when the number of large flows varies. Experiment and simulation results show that our scheme can reroute large flows within 0.5 second, and the maximum link utilization is within 102% of the optimal solution for source and destination addresses-based flows, while the extra LSA number is small.**

I. Introduction

Traffic engineering (TE) is among the most significant tasks of network operators. Rapid growth of the Internet and development of new applications have imposed challenges continually to TE. In particular, fine-grained control upon traffic, e.g. per-flow control, is considered to be necessary in future Internet [1]. To overcome the challenges, Software defined networking (SDN) has been proposed as a promising solution. A huge body of literature focusing on SDN-based TE has emerged. See [1] for a comprehensive survey.

In practice, however, SDN has not been widely deployed for many reasons. First, the flow table capacity of current commercial SDN switches is far from satisfactory. For instance, a high-end Pica8 SDN switch can only hold tens of thousands of flow table entries, far from sufficient for the Internet routing table which has exceeded 700,000 entries.[1] This is because of the large set of match fields for packet processing. Although advanced algorithms have been developed to compress flow table [2], a fundamental change is not made. Second, it is not easy to obsolete existing facilities completely to deploy SDN. Although there have been studies on incremental deployment of SDN [3] and on hybrid operation of SDN and conventional routing [4], challenges still exist. As a result, only a few organizations have deployed SDN for TE, such as Google that developed B4 [5] with proprietary technologies. It is not practical for most ISPs to develop and deploy the whole scheme by themselves.

To address the issue, we take another approach which tries to satisfy the need of fine-grained TE without involving the use of SDN in this paper, i.e. enabling flow-level TE in conventional routing systems. In fact, conventional routing does support per-flow control on traffic, i.e. policy routing based on access control list (ACL). However, similar to flow tables in SDN, ACL also faces the problem of scalability. Fortunately, in the context of TE, a good performance can be achieved by just rerouting a few large flows which dominate the traffic amount. Little extra storage cost for TE will be introduced in this way. Such an idea has been validated in studies on data center networks [6], where large flows can be observed normally. As for the Internet, we conduct an analysis on real Internet traffic trace provided by CAIDA. We find that a small number of large flows dominate the total traffic amount, and most of these large flows last for several minutes. We also develop a model to capture how a large flow is split by a source address prefix with certain length, i.e. how many smaller flows can be obtained, and what are their sizes? See Section III for a more detailed discussion. These results motivate us to materialize the idea of real time large flow detection and rerouting for a better TE, in the Internet with conventional routing protocol, i.e. OSPF.

To realize large flow scheduling in conventional networks, several challenges have to be overcome. A large flow detection mechanism is needed to detect large flows which incur unbalanced traffic distribution or congestion in real time. The rerouting paths of large flows must be computed quickly, while a good TE performance can be achieved. Further, the rerouting path should be distributed efficiently, with little overhead on conventional hop-by-hop routing protocols. An intrinsic tradeoff lies between the match fields to idenfity a flow and the efficient of the scheme. A coarse-grained control needs only a little overhead on computation (large flow detection and rerouting path computation), communication (distribution of rerouting paths), and storage (i.e. ACL entries in TCAM), but the TE performance may degrade. Meanwhile, an over-fine-grained TE is also unacceptable.

In this paper, we propose a link weight assignment-based large flow scheduling scheme. The proposed scheme includes three key components for large flow detection, routing compu-

[1]BGP Routing Table Analysis Reports. http://bgp.potaroo.net/

978-1-5386-2542-2/18/$31.00

tation, and routing distribution. In particular, we extend OSPF to carry changed link weights for a large flow and compute the rerouting path in the same way as normal shortest path computation, i.e. by Dijkstra's algorithm. We formalize the link weight assignment-based large flow scheduling problem, and prove that the problem is NP-hard. We develop the Large Flow Randomized Allocation (LFRA) algorithm to compute the rerouting paths of large flows. We show a set of theoretical results on the TE performance bounds when the number of large flows varies. We further propose the Link State Advertisement Optimization (LSA-Opt) algorithm to reduce the number of extra LSAs needed.

We implement a prototype based on an open source software router, i.e. Quagga, and we conduct experiments based on real topology and playback of real traffic traces to validate our scheme. We also evaluate our algorithms with simulations based on measured topologies and synthetic traffic. The results show that our scheme can reroute a large flow within 0.5 second. The maximum link utilization of our scheme is around 110% of the optimal solution for destination address-based flows, and within 102% of the optimal solution for source and destination addresses-based flows, while the extra LSA broadcast number required is only a few hundreds.

The rest of the paper is organized as follows. Section II summarizes related work. Section III shows an analysis and modelling of large flow on real Internet traffic. We introduce the link weight assignment-based large flow scheduling scheme, formalize the problem, and analyse the complexity in Section IV. Section V is devoted to algorithm development and theoretical analysis. We present our experiments and simulations in Section VI. Section VII concludes the paper.

II. Related Work

Existing traffic engineering approaches can be classified by either routing technology used or traffic information available.

From the routing technology point of view, the approaches differ in the granularity of traffic that can be controlled. First, there are approaches based on traditional intra-domain routing protocol, i.e. OSPF, which uses a destination-based forwarding scheme. For instance, Xu *et. al.* [7] propose to extend the link weights used by OSPF. By assigning link weights appropriately, the traffic can be distributed by equal cost multi-paths (ECMPs) to achieve load balancing. In such approaches, techniques such as ECMP or prefix splitting [8] are needed to split traffic. Second, there are approaches leveraging MPLS [9] or segment routing [4] to control the traffic forwarding for a given flow. Term "flow" here can refer to different granularities of traffic, while most existing approaches deal with the traffic for a given ingress node and egress node pair due to the computation complexity. Third, there are approaches based on advanced technologies such as Openflow , which can control "flows" in an extreme fine-grained manner [10]. For instance, C.-Y. Hong *et. al.* [11] propose a system called SWAN for inter-DC WANs which centrally allocates network paths and achieves a high utilization of the network resource.

Our approach in this paper belongs to the first category, i.e. based on OSPF. We also do some extensions to the protocol to enable a more fine-grained control, i.e. for a large flow with a given source address and a destination address. Nevertheless, we neither require sophisticated mechanisms such as MPLS and Openflow, nor introduce extra overhead to packet headers.

From the traffic information point of view, most approaches deal with a traffic matrix (TM) [12], which includes the traffic demand of each ingress node and egress node pair. However, it is not practical to obtain a TM in real time, due to both the measurement cost and the complexity to derive the TM [13]. To address the issue, some alternative approaches choose to use an optimal static routing for arbitrary TMs. Applegate *et. al.* [14] propose a linear programming to compute the optimal static routing from an ingress node to an egress node, which can minimize the performance ratio of the static routing to the optimal dynamic routing. Known as oblivious routing, the approach is further extended to destination-based routing scheme [15]. Another alternative to address the TM problem is to control large flows only. This is based on the observation that the traffic in a network is dominated by just a small number of flows. Such an idea is made possible by adopting advanced techniques that can measure large flows efficiently [16]. For instance, Al-Fares *et. al.* [6] proposes Hedra for data center networks, which adopts Openflow to find flows that have a large traffic amount and schedule them.

Our approach also focuses on large flows for traffic engineering. To the best of our knowledge, we are the first to introduce the idea into conventional networks, in which it is not practical to adopt Openflow all at once.

III. Large Flow on Internet: Analysis and Modelling

A. An Analysis on Internet Traffic

To validate whether a large flow-based TE is suitable for Internet backbone networks, we need to understand some characteristics of Internet traffic. In particular, we need to answer the following questions. Do large flows dominate the total traffic amount? What is the distribution of flow size? How much time does a large flow last?

We do an analysis on CAIDA anonymized Internet trace captured by two high-speed monitors, i.e. "equinix-sanjose" and "equinix-chicago".[2] The two monitors record an hour of trace four times a year, usually from 13:00 UTC to 14:00 UTC. Each trace contains billions of IPv4 packets, and the data rate is over 1.5 Gbps mostly. The latest traces of "equinix-chicago" available are captured in 2014, 2015, and 2016. In particular, four traces captured by "equinix-chicago" in 2014 are on dates of March 20th, June 19th, September 18th, and December 18th. We select the first two traces of each year for the three years, and the number of the selected traces is six. The latest traces of "equinix-sanjose" available are captured in 2014. Similarly, we select the first two traces of 2014. So we

[2]CAIDA Data. http://www.caida.org/data/

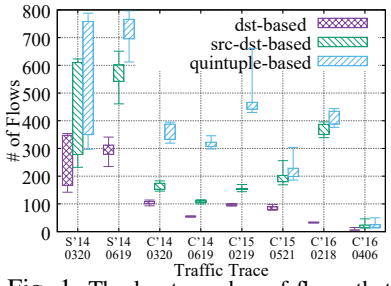

Fig. 1. The least number of flows that compose 35% of total traffic amount from 13:00 to 13:10 for eight traces.

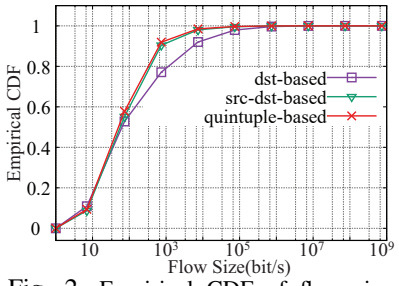

Fig. 2. Empirical CDF of flow size from 13:00 to 13:01 truncated from "C'16 0406".

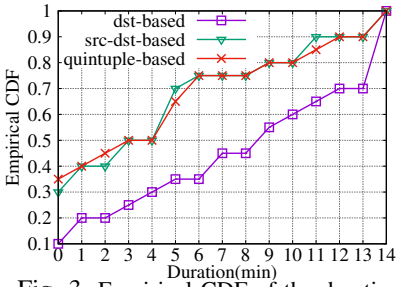

Fig. 3. Empirical CDF of the duration of top 20 largest flows found at 13:00 "C'16 0406".

totally select eight traces, which dominate half of the datasets provided by CAIDA from 2014 to 2016.

We identify a flow with three different granularities: 1) the packets with the same destination address (*dst-based flow*); 2) the packets with the same source and destination address (*src-dst-based flow*); and 3) the packets with the same quintuple of source address, destination address, source port, destination port, and protocol type (*quintuple-based flow*). All the traces are analysed by the minute.

Fig. 1 shows the least number of flows that compose 35% of total traffic amount with respect to eight traffic traces. "S'14 0320" means the traffic trace of "equinix-sanjose" high-speed monitor from 13:00 to 13:10 on March 20th, 2014, and other traces are named in a similar way. We can observe that the result values for src-dst-based flows and quintuple-based flows are always larger than that for dst-based flows. The statistics of the eighth traffic trace shows that the result is less than 15 for dst-based flows, and less than 50 for other two flow types, and the total flow number is about 100,000, 730,000, and 1,000,000, for the three flow types respectively. Fig. 2 shows the flow size of each flow from 13:00 to 13:01 truncated from the eighth traffic trace. We can see that for all three types of flows, only a small portion of flows have a large size. Note that the x axis is in log scale, which means that the flow size distribution is at least exponential. Further, a greater portion of flows have a large amount for dst-based flows, compared with src-dst-based flows and quintuple-based flows. Fig. 3 shows the duration of the top 20 largest flows found at 13:00 for the eighth traffic trace. Here, the duration means that during this time, a flow still has packets and the flow amount is still in top 20. We can see that all 20 flows have a duration less than 14 minutes. Only 20% of the dst-based flows (i.e. four flows) have a duration less than 1 minute, and 40% of the src-dst-based flows and quintuple-based flows have a duration less than 1 minute.

We make a few observations that motivate our study on large flow-based TE in conventional networks. First, the distribution of flow size is uneven. A small number of large flows dominate the total traffic amount, so a good TE performance may be achieved by only rerouting the large flows, and the overhead can be little. Second, most large flows last for a long time, so routing instability may be avoided. Further, identifying a flow with source and destination addresses can split the traffic to an extent similar to that of identifying a flow with the quintuple. This means that we may identify a flow in a simple way and avoid using a large set of match fields.

B. Modelling Large Flow Splitting

Note that dst-based, src-dst-based, and quintuple-based flows are all special cases, and there are many other granularities. In particular, network operators are more likely to use prefixes to identify flows for TE. Using a proper prefix instead of the whole address can make a good balance between flow size and number of flows. In this subsection, we answer the following question: when a dst-based large flow is split by a source address prefix with certain length, how many smaller flows can be obtained, and what are their sizes? We develop a model to capture these characteristics. Our model can be used to compute the optimal prefix length, which results in a proper number of large flows with proper sizes. We will show an example later in this subsection. Our model also provides a means to generate synthetic traffic for simulations, since real traffic trace is not always available for real networks.

Let the packets with the same source address prefix and destination address be a *srcP-dst-based flow*. Given a dst-based flow, the first bit in the source addresses of the packets can split the dst-based flow into two srcP-dst-based flows. Note that these two srcP-dst-based flows may not have the same flow size. Each srcP-dst-based flow can further be split into two srcP-dst-based flows by one more bit in the source address, and so on. The splitting ratio of a certain bit in source address is not a constant. We use a set of probability expressions to describe the distribution of splitting ratio in each splitting step. Formally, let ζ be the source address prefix length and $\zeta \in \{0, 1, ..., 32\}$ for IPv4. Let $< dst/32, src/\zeta >$ be a srcP-dst-based flow. Particularly, $< dst/32, src/0 >$ is a dst-based flow. Flow $< dst/32, src/\zeta - 1 > (\zeta > 0)$ can be split into two flows, both of which can be represented by $< dst/32, src/\zeta >$. Let η be the splitting ratio, which is defined as the size of the smaller flow after splitting, divided by the original flow size, so $\eta \in [0, 0.5]$. Particularly, $\eta = 0$ means that $< dst/32, src/\zeta - 1 >$ is not split. Let $F(\eta, \zeta)$ be the probability distribution function, i.e. cumulative distribution function, of η, when the source address prefix length increases from $\zeta - 1$ to ζ $(\zeta > 0)$. In other words, $F(\eta_0, \zeta)$ is the probability that η is smaller than η_0. For IPv4, there are 32 different functions $F(\eta, \zeta)$, $\zeta \in \{1, 2, ..., 32\}$.

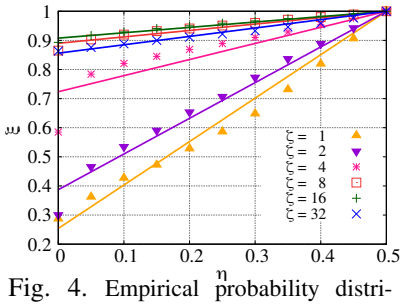

Fig. 4. Empirical probability distributions and their fitting functions for "C'16 0406".

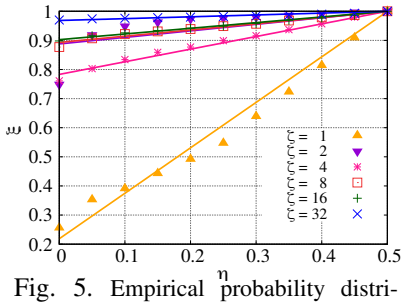

Fig. 5. Empirical probability distributions and their fitting functions for "S'14 0619".

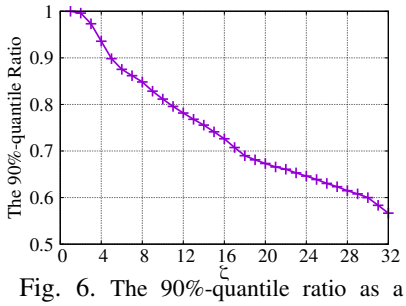

Fig. 6. The 90%-quantile ratio as a function of ζ based on the fitting functions for "C'16 0406".

To obtain the concrete expressions of $\xi = F(\eta, \zeta)$, we again use the CAIDA traffic traces. We analyse the source addresses of the top 20 dst-based flows in each one-minute period, and obtain the empirical CDFs of splitting ratio η for each $\zeta \in \{1, 2, ..., 32\}$. We find that the results from different monitors and time periods have some similar features. Fig. 4 and Fig. 5 show the results of "C'16 0406" and "S'14 0619", respectively.

We observe that the splitting ratio is not evenly distributed. In particular, the probability of no splitting, i.e. $F(0, \zeta)$, is greater than 0.2 for $\zeta = 1$. In most cases, the probability of no splitting increases with the increment of ζ. This is because src-dst-based flows cannot be split anymore, and we get more src-dst-based flows when ζ is greater. We also see that most empirical CDFs are linear, while some CDFs are sub-linear. This means that a flow is always split unevenly, and it is slightly more likely to have a small splitting ratio. In this paper, we set $F(\eta, \zeta)$ to be linear functions.

To show how a dst-based large flow is split by different lengths of source address prefix intuitively, we use $F(\eta, \zeta)$ to simulate splitting process. Given a source address prefix length ζ, we randomly split a dst-based large flow into several srcP-dst-based flows using $F(\eta, \zeta)$. We compute the ratio of the largest srcP-dst-based flow size divided by the original flow size. Fig. 6 shows the 90%-quantile of the results, which means that a ratio is less than the value shown in Fig. 6 with a probability of 90%. Result values are obtained by 10,000 simulations. For example, consider a network manager who wants to split a dst-based large flow with source address prefix. If he/she wants to make the largest flows after splitting be no large than 0.85 times of the original flow size with a probability of 0.9, an 8-bit source address prefix can be used.

Discussion: Note that the addresses in the CAIDA traffic traces are anonymous, which may affect the distribution of splitting ratio. However, we still find some irregular distributions for certain ζ in our results, e.g. when $\zeta = 10, 11, 29, 30, 31, 32$, which may be due to IP address allocations. Flow splitting in real address space is still an open problem, and we leave this to future work.

IV. Link Weight Assignment-based Large Flow Scheduling

A. Overview

In general, a set of packets entering into the network from an ingress node and leaving the network from an egress node is called *traffic demand*. A traffic demand can be divided into *flows*. Here, a flow is a generalized abstract, which means the set of packets with some common label/labels, e.g. IP source/destination address, protocol type, port number, etc. Such a definition allows us to concentrate on the underlying properties of flows, i.e. flow size and forwarding path. It also gives us the flexibility to choose any combination of labels to identify a flow, yet we advocate identifying flows by source and destination addresses for a good TE performance. A *large flow* is a flow whose flow size is greater than a given threshold, e.g. 100 Mbps. Other flows in the network are small flows, which constitute background traffic in our problem.

Our large flow scheduling scheme dynamically detects large flows and makes scheduling decisions to achieve load balancing. In particular, the scheme consists of three components.

1) First, a large flow detection mechanism is needed to detect large flows which incur unbalanced traffic distribution or even congestion in real time.: Since we are working on conventional networks, we can not count on the Openflow counters for per-flow statistic. Fortunately, there has been a set of algorithms developed to detect large flows efficiently [16]. Developing a new algorithm is out of the scope of this paper, and we just choose an appropriated one from the state-of-the-art algorithms.

To realize the detecting mechanism in practice, one possible method is to incorporate the detecting algorithm into each interface of the routers in the network. This method introduces a large amount of cost. An alternative method is to insert a devoted middlebox into each link for detection, but this introduces additional latency to packet forwarding. We use a third method which inserts a switch into each link. The switch forwards the copied traffic of the link to a separate detecting server, which runs the detecting algorithm. Then, the detecting server reports the data periodically to a centralized TE server.

2) Second, a centralized TE server is needed to make scheduling decisions.: In particular, the centralized TE server collects large flow information from detecting servers, creates a global view of current forwarding paths, and computes a routing which will be distributed in real time. The functionality of the TE server is similar to a path computation element (PCE). There have been existing studies on routing computation for large flows in data center networks [6]. However, the challenge of our problem lies in the traffic characteristics of Internet backbones which are different from those in data

center networks. We extend the existing studies by investigating the performance of randomized algorithm under different granularities of traffic flow, in Section V.

3) Third, the scheduling decisions need to be executed.: Again, we can not use such mechanisms as Openflow to distribute the routing computed by the TE server. One method is to configure the ACL of related routers straightforwardly. However, a dynamic routing which can adapt topology change and recover from failures automatically would be more appropriate. In this paper, we extend the conventional hop-by-hop routing protocol, i.e. OSPF, to realize the computed routing. In particular, besides the normal link weights used by OSPF, each link in the network will be assigned with a separate link weight for each large flow that needs rerouting. Such link weights are then advertised throughout the network automatically with extended LSA in a method similar to [17]. Then, each router computes and maintains a separate shortest path tree (SPT) for each large flow, in which the rerouting path of the large flow is consistent with the one computed by the TE server. Finally, the resulting forwarding rule is transformed into an ACL entry in the router. Note that the overhead of LSA advertisement and storage can be minimized if we choose a minimum set of links to assign weights for a given large flow, while leaving the other link weights unchanged, i.e. using the normal OSPF link weights. Less link weight changes also reduce the overhead of SPT computation, with the advanced algorithm which computes SPT incrementally [18]. We formulate our large flow scheduling problem with link weight assignment overhead as a constraint in Section IV.B, and propose an algorithm to reduce the overhead in Section V.B.

B. Problem Formalization

Formally, a network is modelled as directed graph $G(V, E)$ with node set V and edge set E. Let $l(u, v)$ be the undirected link of edge $(u, v) \in E$, which means $l(u, v) = l(v, u)$. Let c_l be the capacity of link l, and f_l be the total traffic amount on link l, which includes the traffic on edges $(u, v) \in l$ and $(v, u) \in l$. Let b_l be the total traffic amount of small flows on link l, i.e. background traffic in our problem. Let D be the set of large flows. For each large flow $i \in D$, let d_i, s_i, t_i be the flow size, ingress node and egress node of i, respectively. Note that multiple large flows may have the same ingress node and egress node, while the flow sizes are separated. Let $y^i_{u,v}$ be a binary variable, which equals 1 if large flow i traverses edge (u, v) or 0 otherwise.

Let $w_{u,v}$ be the weight of edge $(u, v) \in E$, $w^i_{u,v}$ the weight of edge $(u, v) \in E$ with respect to large flow $i \in D$, and $p^i_{s,t}$ denote the path weight of the shortest path from node $s \in V$ to node $t \in V$. To ensure that the path with source node s is shortest, we specify that for all edges $(u, v) \in E$, the sum of $p^i_{s,u}$ and $w^i_{u,v}$ is not less than $p^i_{s,v}$. These constraints are based on the property of shortest path [19]. Let $x^i_{u,v}$ be a binary variable, which equals 1 if $w_{u,v}$ equals $w^i_{u,v}$ or 0 otherwise. The sum of $x^i_{u,v}$ reflects the overhead of assigning link weights for the large flows, and let M be a bound that the sum of $x^i_{u,v}$ must not exceed.

Let $h(f_l)$ be the cost of link l when the traffic amount is f_l. This cost function is used to evaluate the performance of TE. In particular, we choose function $h(f_l) = \frac{f_l^{1+\alpha}}{c_l^{1+\alpha}(1+\alpha)}, \alpha \geq 0$ following [20], which is a general form of several existing cost functions. Let Φ be the maximal cost value over all links. We model the link weight assignment-based large flow scheduling problem (P1) as follows.

$$\min \ \Phi \tag{P1}$$

$$s.t. \quad h(f_l) \leq \Phi, \quad \forall l : (u, v) \in E, \tag{1}$$

$$f_l = b_l + \sum_{(u,v)\in l} \sum_i y^i_{u,v} d_i, \quad \forall l : (u, v) \in E, \tag{2}$$

$$f_l \leq c_l, \quad \forall l : (u, v) \in E, \tag{3}$$

$$y^i_{u,v} \in \{0, 1\}, \quad \forall (u, v) \in E, \forall i \in D, \tag{4}$$

$$\forall i \in D, \forall u \in V :$$

$$F^i(u) = \sum_{v:(u,v)\in E} y^i_{u,v} d_i - \sum_{r:(r,u)\in E} y^i_{r,u} d_i, \tag{5}$$

$$\forall i \in D :$$

$$F^i(u) = \begin{cases} 0, & \text{if } \forall u \in V/\{s_i, t_i\}, \quad (6a) \\ d_i, & \text{if } u = s_i, \quad (6b) \\ -d_i, & \text{if } u = t_i, \quad (6c) \end{cases}$$

$$\forall s \in V, \forall (u, v) \in E, \forall i \in D :$$

$$w^i_{u,v} + p^i_{s,u} - p^i_{s,v} \geq 0, \tag{7}$$

$$p^i_{s_i,t_i} = \sum_{u,v} y^i_{u,v} w^i_{u,v}, \quad \forall i \in D, \tag{8}$$

$$\forall i \in D, \forall (u, v) \in E :$$

$$x^i_{u,v} = \begin{cases} 1, & \text{if } w^i_{u,v} \neq w_{u,v}, \quad (9a) \\ 0, & \text{if } w^i_{u,v} = w_{u,v}, \quad (9b) \end{cases}$$

$$\sum_i \sum_{u,v} x^i_{u,v} \leq M. \tag{10}$$

Eq. (1) means that Φ is the maximum cost among all links. Eq. (2) means that the total traffic amount on each link consists of background traffic and large flows traversing the link. Eq. (3) means that the total traffic amount on each link must not exceed the capacity. Eq. (4) specifies that $y^i_{u,v}$ is a binary variable. Eq. (5) and Eq. (6) mean that for each flow, the traffic entering and leaving each node should be consistent. Eq. (7) guarantees that $p^i_{s,t}$ is the shortest path weight with respect to $w^i_{u,v}$. And Eq. (8) specifies the rerouting path of each flow to be the shortest path. Eq. (9) defines $x^i_{u,v}$, and Eq. (10) specifies the bound on link weight assignment overhead. The decision variables of problem P1 are $w^i_{u,v}$, $x^i_{u,v}$, $y^i_{u,v}$, and $p^i_{s,v}$. Actually, we just need to compute link weight $w^i_{u,v}$, while the others can be computed with $w^i_{u,v}$.

C. Problem Analysis

Theorem 1: Problem P1 is NP-hard.

Proof: The theorem can be proved by reducing problem P1 to the two-commodity integral flow (TCIF) problem in polynomial time, which is NP-complete [21]. This can be done by specifying that there are only two flows and M is sufficiently large. ■

There are two factors of problem P1 reduced in the proof above. First, the flows in problem P1 may have the same ingress node and egress node, while the flows in the TCIF problem do not. In such a case, we show how to construct an instance of problem P1. Assume that the two-commodity flows are from s_1 to d_1 and from s_2 to d_2, respectively. we add additional source node s as the ingress node of problem P1, and add edges (s, s_1) and (s, s_2), whose capacities are equal to the amount of the two-commodity flows, respectively. Similarly, we add addition destination node d as the egress node of problem P1, and add edges (d_1, d) and (d_2, d) with the same capacities as (s, s_1) and (s, s_2), respectively. Let problem P1 has two large flows, with the same amounts as the two-commodity flows. It can be seen that the two instances are equivalent.

Second, the overhead of link weight assignment is slacked in the proof. We show that it is non-trivial to satisfy such a constraint, even if there is only one large flow to be routed. Assume that we have computed a target path that the large flow *must* follow, and assume that all link weights, i.e. $w_{u,v}$, are 1. Then, the problem is to find a set of links to cut,[3] such that the target path is the shortest path. When M is small, the problem is equivalent to minimize the set of cut links. Note that, we can not simply remove the target path and compute a minimum cut. This may not produce a feasible solution when the target path has more than one hop, because there may be a shortcut for a sub-path of the target path. In fact, we need a cut for each sub-path of the target path, and the optimal solution must use the minimal links to cover each of the cuts. This makes the problem a minimum dominating set problem, which is NP-hard [22].

V. Algorithms

In this section, we take a heuristic to solve problem P1 which is NP-hard as shown above. In particular, we first compute the paths of the large flows, without the constraint on link weight assignment overhead. Then, we try to distribute the computed paths information with minimal number of link weight assignments.

A. Large Flow Randomized Allocation

As discussed above, the major difference between problem P1 and the typical multi-commodity flow (MCF) problem is that, there may be multiple large flows from an ingress node and an egress node in problem P1, while each of these flows has to be routed in an integrated manner. To overcome the challenge, we propose to randomly choose a path for each large flow, from a set of candidate paths.

The key observation is that, by considering the large flows with the same ingress and egress nodes as a whole, we obtain an MCF problem, and by solving the slacked linear programming of the MCF problem and computing the augmenting paths, we can obtain the set of candidate paths and the probability of choosing each of them. Formally, let the large flows with the same ingress and egress node pair be an *integrated flow*. Let D_j be the amount of an integrated flow. We solve the MCF problem with the integrated flows as the traffic demands (background traffic is not rerouted). In the fractional optimal solution, let $\bar{y}_l^j$ be the amount of the traffic demand (i.e. the integrated flow) that traverses link l, divided by D_j. Then, $\bar{y}_l^j$ is the probability that a large flow belonging to D_j traverses link l. We call the algorithm large flow randomized allocation (LFRA).

It can be observed that LFRA is reduced to randomized rounding [23] when there is at most one large flow for each ingress and egress node pair. While randomized rounding has a large bound on the TE performance [24], LFRA performs better when the number of large flows becomes greater. Intuitively, the traffic is more likely to be spread on the augmented paths instead of integrated on one path or two. To validate the performance of LFRA, we show some theoretical results.

We first consider a simple situation in which there is only one integrated flow consisting of n large flows. The integrated flow has a total amount D, and each large flow has the same flow size $S = \frac{D}{n}$. We also assume that there is no background traffic. The process of n large flows selecting link l or not can be considered as n independent Bernoulli trails $x_1, x_2, \cdots, x_n$. According to LFRA, we have $Pr(x_j = 1) = p = \bar{y}_l$. Let $X_l = \sum_{k=1}^{n} x_k$, which means the number of large flows traversing link l. Let Y_l be the total amount of large flows on link l, so $Y_l = S \cdot X_l$. $Y_l^* = E[Y_l] = Dp$ is the optimal traffic amount on link l.

Theorem 2: Let $\Delta = \frac{|Y_l - Y_l^*|}{Y_l^*}$, and β be a constant. Then $Pr(\Delta \le \beta) \ge 1 - 2e^{-\frac{\Delta^2 np}{3}}$ for $0 < \beta < 1$, and $Pr(\Delta \le \beta) \ge 1 - e^{-\frac{\Delta^2 np}{3}}$ for $\beta \ge 1$.

Proof:

$$Pr(X_l \ge \gamma) = \sum_{k=\lceil\gamma\rceil}^{n} \left(\binom{n}{k} \cdot p^k \cdot (1-p)^{n-k}\right),$$

$$\text{Now let,} \quad \gamma = (1+\Delta) \cdot np, \text{ and } \Delta > 0, \tag{11}$$

$$Pr(X_l \ge \gamma) = Pr(X_l \ge (1+\Delta) \cdot np) \le e^{-\frac{\Delta^2 np}{3}}, \tag{12}$$

$$Pr(Y_l \ge \gamma \cdot \frac{D}{n}) = Pr(X_l \ge \gamma) \le e^{-\frac{\Delta^2 np}{3}}. \tag{13}$$

Similarly,

$$Pr(X_l \le \gamma) = \sum_{k=0}^{\lfloor\gamma\rfloor} \left(\binom{n}{k} \cdot p^k \cdot (1-p)^{n-k}\right),$$

$$\text{Now let,} \quad \gamma = (1-\Delta) \cdot np, \text{ and } 0 < \Delta < 1, \tag{14}$$

$$Pr(X_l \le \gamma) = Pr(X_l \le (1-\Delta) \cdot np) \le e^{-\frac{\Delta^2 np}{2}}, \tag{15}$$

$$Pr(Y_l \le \gamma \cdot \frac{D}{n}) = Pr(X_l \le \gamma) \le e^{-\frac{\Delta^2 np}{2}}. \tag{16}$$

The two inequalities in Eq. (12) and Eq. (15) follows from two derivation inequalities from Chernoff bound [4] .

Let β be a constant. if $0 < \beta \le 1$ and according to Eq. (11)(13)(14)(16), we can get

$$Pr(\Delta \le \beta) \ge 1 - e^{-\frac{\Delta^2 np}{3}} - e^{-\frac{\Delta^2 np}{2}} \ge 1 - 2e^{-\frac{\Delta^2 np}{3}}. \tag{17}$$

[3] In fact, we just set a sufficient large weight to these links for the given large flow, and this is equivalent to cut the links.

[4] Chernoff Bound. https://en.wikipedia.org/wiki/Chernoff_bound

If $\beta \geq 1$ and according to Eq. (11)(13), we can get

$$Pr(\Delta \leq \beta) \geq 1 - e^{-\frac{\Delta^2 np}{3}}. \quad (18)$$

■

Theorem 2 shows two lower bounds of the probability that the deviation Δ falls into β. The bounds increase when n increases, which means that LRFA will perform better if the integrated flow contains more large flows, e.g. by using more labels to identify a flow. Theorem 2 can be extended to the situation with m integrated flows with size $D_j (j = 1, 2, \cdots, m)$. Each integrated flow consists of n_j large flows with the same size $S_j = \frac{D_j}{n_j}$. Similarly, we can define x_k^j, $X_{j,l}$, $Y_{j,l}$, and $Y_{j,l}^*$ with respect to each integrated flow.

Theorem 3: Let $\Delta_j = \frac{|Y_{j,l} - Y_{j,l}^*|}{Y_{j,l}^*}$, and β be a constant. Then, $Pr(\Delta_1 \leq \beta, \Delta_2 \leq \beta, \cdots \Delta_m \leq \beta) \geq \prod_{j=1}^{m}(1 - 2e^{-\frac{\beta^2 n_j p_j}{3}})$ for $0 < \beta < 1$, and $Pr(\Delta_1 \leq \beta, \Delta_2 \leq \beta, \cdots \Delta_m \leq \beta) \geq \prod_{j=1}^{m}(1 - e^{-\frac{\beta^2 n_j p_j}{3}})$ for $\beta \geq 1$.

Proof:

According to Eq. (17)(18) we obtain

$$Pr(\Delta_j \leq \beta) \geq 1 - 2e^{-\frac{\beta^2 n_j p_j}{3}}, \text{for } 0 < \beta < 1, \ \mathrm{j} = 1, \cdots, \mathrm{m}.$$

$$Pr(\Delta_j \leq \beta) \geq 1 - e^{-\frac{\beta^2 n_j p_j}{3}}, \text{for } \beta \geq 1, \ \mathrm{j} = 1, \cdots, \mathrm{m}.$$

The selection procedures of large flows from a same integrated flow or different ones are all independent. So, we get the upper theorem by multiplying m inequations simply. ■

Assume that all the links are with a same link capacity c_l. We define $Y^* = Max\{Y_l^*, \ \forall l : (u, v) \in E\}$. Let OPT be the optimal network cost which equals $\frac{Y^{*1+\alpha}}{c_l^{1+\alpha}(1+\alpha)}$ with respect to the assumption. We have the following theorem.

Theorem 4: Let all links have identical capacity, h_{max} the maximum link cost with LRFA, and $0 < \beta < 1$, then

$$Pr(h_{max} \geq (1+\beta)^{1+\alpha} OPT) \leq \sum_{l:(u,v)\in E} \sum_{j=1}^{m} 2e^{-\frac{\beta^2 n_j p_j}{3}}.$$

Proof: Through the definition of Y^*, we have

$$Pr(Y_l \leq (1+\beta)Y^*) \geq Pr(Y_l \leq (1+\beta)Y_l^*)$$

Through the proof of theorem 3, we can get

$$Pr(Y_l \leq (1+\beta)Y^*) \geq \prod_{j=1}^{m}(1 - 2e^{-\frac{\beta^2 n_j p_j}{3}})$$

$$\geq 1 - \sum_{j=1}^{m} 2e^{-\frac{\beta^2 n_j p_j}{3}}, \ 0 < \beta < 1$$

The probability that Y_l exceeds $(1+\beta)Y^*$ is not more than $\sum_{j=1}^{m} 2e^{-\frac{\beta^2 n_j p_j}{3}}$. So as for a network with $|E|$ edges, the probability that at least one Y_l of link l exceeds $(1+\beta)Y^*$ is not more than

$$1 - \prod_{l:(u,v)\in E}(1 - \sum_{j=1}^{m} 2e^{-\frac{\beta^2 n_j p_j}{3}}) \leq \sum_{l:(u,v)\in E} \sum_{j=1}^{m} 2e^{-\frac{\beta^2 n_j p_j}{3}}$$

From previous definitions, we have known that $OPT = \frac{Y^{*1+\alpha}}{c_l^{1+\alpha}(1+\alpha)}$ and $h_{max} = Max\{\frac{Y_l^{*1+\alpha}}{c_l^{1+\alpha}(1+\alpha)}, \ \forall l : (u, v) \in E\}$. So, for $0 < \beta < 1$ we can express upper inequality as

$$Pr(h_{max} \geq (1+\beta)^{1+\alpha} OPT) \leq \sum_{l:(u,v)\in E} \sum_{j=1}^{m} 2e^{-\frac{\beta^2 n_j p_j}{3}},$$

■

Theorem 4 tells us that, the probability of obtaining a good TE performance increases when n_j for any integrated flow increases.

B. Link State Advertisement Optimization

Now, we consider the problem of changing link weights for the large flows whose rerouting paths have been computed by LFRA. As discussed earlier, we try to reduce the number of link weight changes so as to reduce the overhead of link state advertisements. Since the link weights for each large flow is independent, we consider one target path.

To ensure that target path P is the shortest path after changing some link weights, each sub-path of P must not have a shortcut, as discussed in Section IV.C. Because there may be a large number of shortcuts, e.g., the shortest path, the 2nd shortest path, etc., it is difficult to increase the path weights of all these shortcuts by increasing a few link weights. Thus, we use an alternative method instead. We multiply each normal link weight by a positive constant, such that the shortest paths are not changed and each link weight is greater than the hop number of the longest target path possible (e.g. the number of links in the network). This can be done at any time before a large flow scheduling is performed, and no extra storage is required. Then, we decrease the link weights of P. In the extreme case when all links in P have the possible least weight, i.e. 1 in OPSF, there will no longer be any shortcut for any sub-path of P.

To further reduce the number of link weight changes, we make the following observation. If P and each of its shortcuts are link independent, then P will become the shortest path if the weight of P becomes less than the weight of the shortest shortcut. In such a case, we can greedily assign the largest link weight of P to 1 until the condition is satisfied, and thus the least number of link weight changes may be achieved.

Note that link independency is very important here. Without link independency, even if the weight of P is less than the weight of the shortest shortcut, the 2nd shortest shortcut may not be eliminated, which produces an incorrect result. Fig. 7 shows an example, in which $ABCD$ is the target path. There are two shortcuts ACD and AD, and AD is the shortest shortcut. After we change link weight of (C, D) to 1, AD is no longer a shortcut, but ACD remains shorter than $ABCD$ because they share link (C, D). To address the issue, we propose to eliminate the shortcuts for sub-paths of P first. This guarantees P and each of its shortcuts are link independent.

We develop the LSA-Opt algorithm which contains three loops. The first one is to visit the nodes in the target path in the reverse order. The second one is to find the shortcuts, and the third one is to eliminate shortcuts by changing link

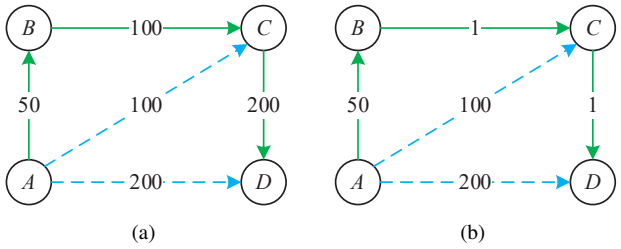

Fig. 7. An example of reducing link weights. (a) Original topology. Solid lines indicate target path and dashed lines are shortcuts. (b) A correct solution where two link weights are reduced and the target path becomes the shortest path now.

weights. The time complexity of the LSA-Opt algorithm is $O(|V|^3)$. Due to page limit, we omit the pseudocode hear.

VI. Performance Evaluation

A. Experiments

1) Experiment Setup: We implement a prototype system with three components of our large flow scheduling scheme. First, we implement detecting algorithms for large flows and background traffic in PCs. In particular, we implement Space-Saving [16], one of the state-of-the-art algorithms, that can detect large flows, and update the results in real time with sliding windows. The traffic on each link is copied by a switch and forwarded to a detecting server.

Second, we implement a centralized TE server to collect traffic information and compute rerouting paths for large flows. In particular, the TE server receives flow statistics about large flows and background traffic from every detecting server. An MCF problem is constructed with the traffic information, and solved by Gurobi. Then, we invoke the LFRA algorithm to compute the rerouting path of each large flow, and use the LSA-Opt algorithm to reduce the extra LSA broadcast number. Finally, the link weights for large flows are configured to corresponding routers.

Third, we modify the conventional OSPF protocol to support extended LSA which carries the link weight for each large flow. The implementation is based on open source software routing suite Quagga, and we modified about 5,000 lines of C codes. In particular, a router running the extended OSPF will flood extended LSAs as normal ones and then compute a separate shortest path tree for each large flow with Dijkstra's algorithm. The result is a forwarding rule with multiple match fields, which is then added into ACL.

We construct an experimental network using nine commercial routers whose software is replaced by our modified one. We use the topology of the Abilene Research network. Each link has a capacity of 1 Gbps, and has a normal link weight of 50. The detecting server and the centralized TE server are on laptops with a 2-core Intel Core i5 2.4 GHz CPU and 4 GB memory.

2) Experiment Results: To evaluate the time to detect and reroute large flows, we conduct a simple experiment. In particular, we generate 300 Mbps traffic from Denver node to New York node when $t = 0s$, which traverses link 1. And other links are already carrying 170 Mbps traffic as background traffic. All the traffics are generated by replaying CAIDA traffic data. When $t = 4s$, the detecting servers report traffic information to the centralized TE server. Fig. 8 shows the result. We can observe that some large flows are shifted to another path which passes link 2 in 0.5 second. This implies that our scheme reroutes large flows rapidly. In practice, the detecting server reports traffic information periodically, e.g. four seconds in our experiment. The centralized TE server will also compute rerouting paths periodically.

We also evaluate the accuracy of large flow detection, by recording the average relative errors of the flow size of top 10 large flows, using CAIDA traffic data. Fig. 9 shows the results with different traffic rates. We can find that the errors are at most 2% as time goes on for three kinds of rates, which means our large flow detection is accurate and stable. We also observe that the errors for larger rate are usually larger and fluctuate more fiercely. This is because larger rate incurs more packet losses, and the relative error is related to packet loss ratio besides the errors incurred by the algorithm itself.

B. Simulations

1) Simulation Setup: We use six measured topologies provided by Rocketfuel project [25] in our simulations. The link capacity is determined by the degrees of the two nodes adjacent to the link, following [26]. In particular, the link capacity is set to 9,953 Mbps if the two adjacent nodes both have a degree greater than 5; otherwise, the link capacity is set to 2,488 Mbps. The link weight is set as the suggested value [25] multiplied by 50.

We use synthetic traffic amount. First, the background traffic amount on each link is generated randomly, which has a uniform distribution within 15% to 20% of the link capacity. Second, we choose a set of ingress and egress node pairs, each of which has an integrated flow. The number of node pairs (integrated flows) ranges from 5 to 50. Each integrated flow contains one or two dst-based large flows with equal probability. The flow size of each dst-based large flow is randomly generated between 100 Mbps and 150 Mbps. Third, we divide each dst-based large flow into srcP-dst-based flows using the probabilistic splitting functions mentioned in section III.B. All the nodes use a same set of coefficients.

A few default values are set as follows. α in link cost function $h(f_l)$ is set to 0, which means the maximum link utilization ratio. The default number of node pairs is 30. The default source address prefix length of srcP-dst-based flow is 32 bits, and we use src-dst-based flow to represent the default srcP-dst-based flow. The srcP-dst-based flows, whose flow sizes exceed a threshold, are considered as large flows in our simulations. Others are dealt with the same as background traffic. The default value of this threshold is 10 Mbps.

We evaluate the LFRA algorithm with the LSA-Opt algorithm, with both dst-based flows and srcP-dst-based flows. For comparison, we also evaluate conventional OSPF and the optimal solution that can only be achieved by fractional routing. We first evaluate the effectiveness of our scheme,

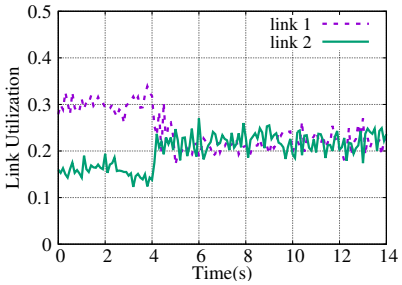

Fig. 8. The evaluation of time for detecting and rerouting large flows.

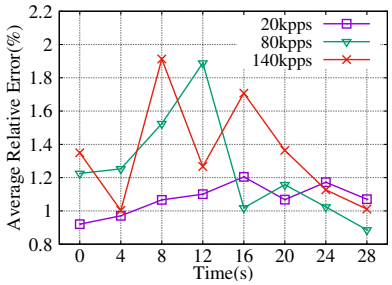

Fig. 9. Average relative errors of measuring the flow size of top 10 flows at different traffic replay rates.

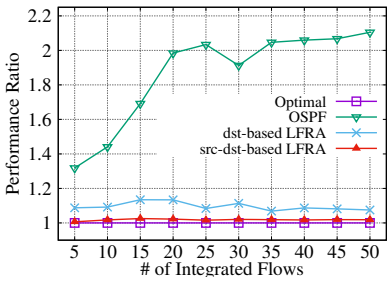

Fig. 10. Performance ratio as a function of integrated flow number, in topology of AS 1755.

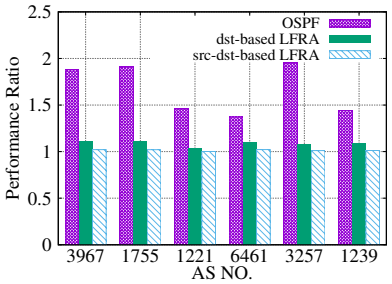

Fig. 11. Performance ratio in different topologies.

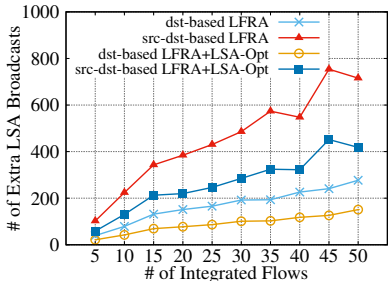

Fig. 12. Extra LSA broadcast number as a function of integrated flow number, in topology of AS 1755.

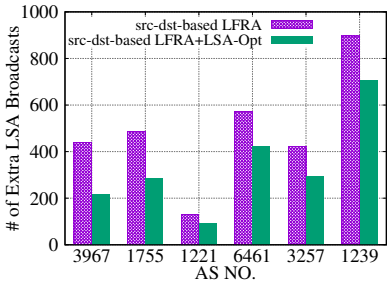

Fig. 13. Extra LSA broadcast number in different topologies.

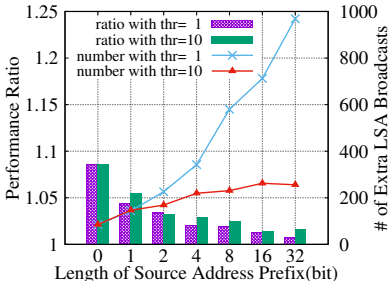

Fig. 14. Performance ratio and extra LSA broadcast number as a function of source address prefix length, in topology of AS 1755.

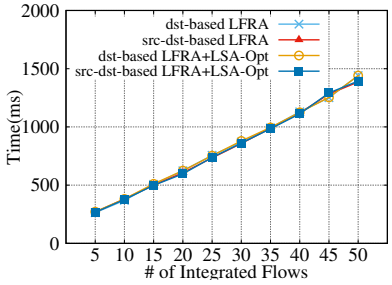

Fig. 15. Computation time as a function of integrated flow number, in topology of AS 1755.

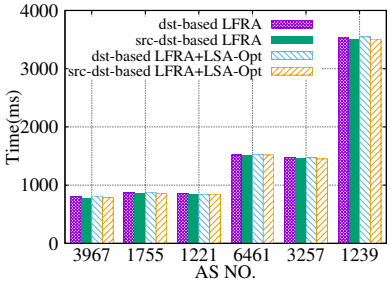

Fig. 16. Computation time in different topologies.

which is measured by the maximum link cost Φ of our scheme divided by that of the optimal routing. We call this measure the *performance ratio*. Second, we evaluate the number of extra LSA broadcasts needed to enable the rerouting paths of the large flows. We compare our scheme to an Openflow-like scheme, which configures each router along a target path. Third, we evaluate the effect to performance ratio and the number of extra LSA broadcasts as the source address prefix length of srcP-dst-based flow increases. At the same time, we evaluate the effects of threshold identifying large flow to results. Finally, we evaluate the execution time of the algorithms. All simulations are performed on a PC which has 4-core Intel Core i7-6700 3.4 GHz CPU and 8 GB memory. We use Gurobi to solve the MCF problem. Each point in our results is the average of 10 independent simulations.

2) Simulation Results: Fig. 10 shows the performance ratio as a function of integrated flow number in topology of AS 1755. We can see that for both dst-base flows and src-dst-based flows, the performance ratio of our scheme changes little with the increment of integrated flow number, while the performance ratio of OSPF is increasing rapidly. This is because our algorithms can balance the large flows effectively, while with OSPF, the large flows are more likely to incur a congestion as the integrated flow number increases. We find that our scheme has a performance ratio very close to 1 for src-dst-based flows, which is about 10% less than the performance ratio for dst-based flows. This implies that by identifying flows with only source address and destination address, a near-optimal TE performance can be achieved. When there are 30 integrated flows, our scheme reduces the performance ratio by 90% compared to conventional OSPF routing.

The results in Fig. 10 are not special. Fig. 11 shows the performance ratio in all six topologies. We can observe that our scheme always has a good performance, for both types of flow. The performance ratio of our scheme for src-dst-based flows is 40% to 90% less than that of OSPF.

Fig. 12 shows the number of extra LSA broadcasts as a function of integrated flow number in topology of AS 1755. We see that the number of extra LSA broadcasts increases in general, but it decreases a little when the integrated flow

number is large. This is because when there are many large flows, some of the large flows can use a path close to the original shortest path to achieve load balancing. The extra LSA needed by src-dst-based flows is greater than that of dst-based flows, because the number of flows increases. However, LSA-Opt reduces the extra LSA by about 50%, so the extra LSA broadcast number is less than 450, which is acceptable for current network size. Fig. 13 shows the results in all six topologies, which are similar to that in Fig. 12.

Fig. 14 shows the performance ratio and the number of extra LSA broadcasts as a function of source address prefix length in topology of AS 1755. We can see the performance ratio decreases as the source address prefix length increases, while the number of extra LSA broadcasts increases on the contrary. This is because a longer source address prefix length can split a dst-based flow into more srcP-dst-based flows. We also observe that the threshold equalling to 1 Mbps when identifying large flows leads to a smaller performance ratio than the threshold equalling 10 Mbps. On the contrary, extra LSA broadcast number is larger if the threshold equals to 1 Mbps. This is because more flows will be considered and scheduled as large flows when using a smaller threshold.

Fig. 15 shows the computation time of our algorithms, including solving the MCF problem, large flow randomly allocation, and LSA optimization. We see that the time is increasing linearly with the integrated flow number, reaching 1.4 seconds for 50 integrated flows. The results change little for both flow types, and also change little when LSA-Opt is used or not. These imply that the main complexity is from solving the MCF problem, and our processes consume little time. Fig. 16 shows the results in all six topologies. We find that most topologies have an acceptable computation time, while largest network, i.e. AS 1239, needs around 3.5 seconds. Since the time is mainly introduced by solving the MCF problem, we plan a study on reducing the integrated flow number before invoking Gurobi in our future work.

VII. Conclusion

In this paper, we studied flow-level traffic engineering in conventional networks that have no SDN deployed. We proposed to detect and schedule a few large flows in real time, which dominate the traffic amount according to our analysis on real Internet traffic traces. We leveraged extended OSPF to distribute rerouting paths, formalized the link weight assignment-based large flow scheduling problem and developed heuristics to solve it. We conducted experiments and simulations that validate the effectiveness of our scheme.

VIII. Acknowledgment

The research is supported by the National Natural Science Foundation of China under Grant (61625203, 61502268), the National Key R&D Program of China under Grant (2017YFB0803202, 2016YFC0901605) and the Science&Technology Program of Beijing (Z171100005217001).

References

[1] A. Mendiola, J. Astorga, E. Jacob, and M. Higuero, "A survey on the contributions of software-defined networking to traffic engineering," *IEEE Communications Surveys & Tutorials*, 2017.

[2] H. Zhu, M. Xu, Q. Li, J. Li, Y. Yang, and S. Li, "MDTC: An efficient approach to TCAM-based multidimensional table compression," in *IFIP Networking*, 2015, pp. 1–9.

[3] Y. Guo, Z. Wang, X. Yin, X. Shi, and J. Wu, "Traffic engineering in SDN/OSPF hybrid network," in *IEEE ICNP*, 2014, pp. 563–568.

[4] 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," *ACM SIGCOMM CCR*, vol. 45, no. 4, pp. 15–28, 2015.

[5] 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," *ACM SIGCOMM CCR*, vol. 43, no. 4, pp. 3–14, 2013.

[6] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks." in *NSDI*, 2010, pp. 1–15.

[7] K. Xu, M. Shen, H. Liu, J. Liu, F. Li, and T. Li, "Achieving optimal traffic engineering using a generalized routing framework," *IEEE/ACM TPDS*, vol. 27, no. 1, pp. 51–65, 2016.

[8] N. Kang, M. Ghobadi, J. Reumann, A. Shraer, and J. Rexford, "Efficient traffic splitting on commodity switches," in *ACM CoNEXT*, 2015, pp. 1–13.

[9] D. O. Awduche, "MPLS and traffic engineering in IP networks," *IEEE communications Magazine*, vol. 37, no. 12, pp. 42–47, 1999.

[10] S. Agarwal, M. Kodialam, and T. Lakshman, "Traffic engineering in software defined networks," in *IEEE INFOCOM*, 2013, pp. 2211–2219.

[11] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven WAN," *ACM SIGCOMM CCR*, vol. 43, no. 4, pp. 15–26, 2013.

[12] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot, "Traffic matrix estimation: Existing techniques and new directions," *ACM SIGCOMM CCR*, vol. 32, no. 4, pp. 161–174, 2002.

[13] M. M. Rahman, S. Saha, U. Chengan, and A. S. Alfa, "IP traffic matrix estimation methods: Comparisons and improvements," in *IEEE ICC*, 2006, pp. 90–96.

[14] D. Applegate and E. Cohen, "Making intra-domain routing robust to changing and uncertain traffic demands: Understanding fundamental tradeoffs," in *ACM SIGCOMM*, 2003, pp. 313–324.

[15] M. Chiesa, G. Rétvári, and M. Schapira, "Lying your way to better traffic engineering," in *ACM CoNEXT*, 2016, pp. 1–8.

[16] R. Ben-Basat, G. Einziger, R. Friedman, and Y. Kassner, "Heavy hitters in streams and sliding windows," in *IEEE INFOCOM*, 2016, pp. 1–9.

[17] F. Baker, "draft-ietf-ospf-ospfv3-lsa-extend-14." Internet Draft, 2017.

[18] P. Narvaez, K.-Y. Siu, and H.-Y. Tzeng, "New dynamic SPT algorithm based on a ball-and-string model," *IEEE/ACM TON*, vol. 9, no. 6, pp. 706–718, 2001.

[19] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network flows: theory, algorithms, and applications*. Prentice Hall, 1993.

[20] M. Shafiee and J. Ghaderi, "A simple congestion-aware algorithm for load balancing in datacenter networks," in *IEEE INFOCOM*, 2016, pp. 1–9.

[21] S. Even, A. Itai, and A. Shamir, "On the complexity of time table and multi-commodity flow problems," *SIAM Journal on Computing*, vol. 5, no. 4, pp. 691–703, 1976.

[22] R. G. Michael and S. J. David, *Computers and intractability: a guide to the theory of NP-completeness*. W.H. Freeman, 1979.

[23] P. Raghavan and C. D. Tompson, "Randomized rounding: a technique for provably good algorithms and algorithmic proofs," *Combinatorica*, vol. 7, no. 4, pp. 365–374, 1987.

[24] M. Andrews, A. F. Anta, L. Zhang, and W. Zhao, "Routing for power minimization in the speed scaling model," *IEEE/ACM TON*, vol. 20, no. 1, pp. 285–294, 2012.

[25] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with rocketfuel," *ACM SIGCOMM CCR*, vol. 32, no. 4, pp. 133–145, 2002.

[26] Y. Yang, M. Xu, D. Wang, and S. Li, "A hop-by-hop routing mechanism for green internet," *IEEE/ACM TPDS*, vol. 27, no. 1, pp. 2–16, 2016.