Computer Networks 185 (2021) 107671

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Flow-level and efficient traffic engineering in conventional routing systems

Nan Geng, Yuan Yang *, Mingwei Xu *

Department of Computer Science and Technology, Tsinghua University, Beijing, China

ARTICLE INFO

Keywords:
Flow-level traffic engineering
Large flow
OSPF
Source and destination addresses

ABSTRACT

Existing solutions of flow-level traffic engineering (TE) usually depend on the deployment of SDN or MPLS. In this paper, we design a flow-level and efficient TE scheme based on the conventional hop-by-hop routing protocol, i.e., OSPF. Motivated by the analysis and modeling on the real Internet traffic, we propose to detect and schedule a few large flows in real-time. The rerouting paths for large flows are computed in a centralized server and are distributed through extended OSPF. A few ACL entries are used for flow-level forwarding. We formalize the link weight assignment-based large flow scheduling problem and prove the problem is NP-hard. We propose to precompute several candidate paths to reduce decision computation overhead and path stretch. We develop an algorithm with performance bounds to allocate large flows to paths, and two algorithms to reduce extra LSA number for different system designs. Experiment results show our scheme can reroute large flows within 0.5 s. Simulation results show our scheme gets congestion metric values (i.e., performance ratios) 10% worse than the optimal for source and destination addresses-based flows. Our optimization mechanisms reduce the extra LSA number and computation time by 87% and 83% respectively for our scheme with pre-computed paths.

1. Introduction

A huge body of literature focuses on traffic engineering (TE) which is essential to improving network performance and resource utilization. TE plays a more and more important role in production networks with the rapid growth of the Internet and the development of new applications. To meet the requirement of the future Internet, fine-grained control upon flow, e.g., per-flow control, is considered to be promising and necessary [1,2]. Software defined networking (SDN) which supports flexible flow control attracts the attention of both network operators and academic researchers. Lots of SDN-based TE solutions have been proposed recently [1].

Although recent years have seen a rapid development of SDN, there are still a considerable proportion of networks using traditional routing technologies. For example, according to the 2020 Global Networking Trends Report published by Cisco [3], there are over 40% percent of organizations leverage traditional techniques to manage Wide Area Networks (WANs) without using any basic or advanced SDN technologies (e.g., SD-WAN). Besides, there are also some obstacles on the way of deploying SDN. First, it is challenging for current commercial SDN switches to contain enough flow table entries to support the fast-growing Internet. 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 800,000 entries.[1] This is because a large set of fields is generally used to match incoming packets. Although much effort has been invested in the compression of flow table [4,5], 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 the incremental deployment of SDN [6] and on the hybrid operation of SDN and conventional routing [7], challenges still exist. As a result, traditional routing technologies will still play an important role in current and future networks.

In this paper, we take another approach which tries to satisfy the need of fine-grained TE without involving the use of SDN, 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 [8], 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

* Corresponding authors.
E-mail addresses: gn16@mails.tsinghua.edu.cn (N. Geng), yangyuan_thu@mail.tsinghua.edu.cn (Y. Yang), xumw@tsinghua.edu.cn (M. Xu).

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

https://doi.org/10.1016/j.comnet.2020.107671
Received 25 December 2019; Received in revised form 22 October 2020; Accepted 29 October 2020
Available online 7 November 2020

a certain length, i.e., how many smaller flows can be obtained, and what are their sizes? See Section 3 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, on the Internet with the conventional routing protocol, i.e., OSPF.

We identify several design challenges that need to be addressed in order to realize large flow scheduling in conventional routing systems. Large flows which incur unbalanced traffic distribution or congestion should be detected in real-time. The rerouting paths should be computed carefully to achieve a good TE performance, and the computation overhead must be small for fast traffic scheduling. Further, the rerouting path should be distributed efficiently, with little overhead on conventional hop-by-hop routing protocols. There are two factors that affect the efficiency of the scheme directly. (1) The match fields to identify a large flow. 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. (2) The limitation of rerouting paths. By limiting the number and length of rerouting paths, less overhead of computation, communication, and storage will be induced, but a loss of TE performance may happen. Meanwhile, computing rerouting paths without path limitations are more likely to achieve a good TE performance.

In this paper, we propose a link weight assignment-based large flow scheduling scheme. We suggest detecting large flows identified by source and destination addresses. Fig. 1 shows an intuitive example to illustrate that using source and destination addresses is enough for achieving good TE performance. A centralized server is leveraged to compute rerouting paths for large flows at each TE interval. To distribute the rerouting path for a large flow, we extend OSPF to carry changed link weights for the large flow, and distributed routers will compute the rerouting path in the same way as normal shortest path computation, i.e., by Dijkstra's algorithm. Three novel contributions in designing our scheme are shown as follows.

First, we formalize the link weight assignment-based large flow scheduling problem and prove that the problem is NP-hard. We propose to compute a limited number of low-stretch candidate paths for any pair of nodes in advance. Our scheme with pre-computed paths works more efficiently, while good TE performance can also be achieved. We also develop the Large Flow Randomized Allocation (LFRA) algorithm to allocate large flows to rerouting paths. We show a set of theoretical results on the TE performance bounds when the number of large flows varies.

Second, we propose two Link State Advertisement Optimization (LSA-Opt) algorithms to reduce the number of extra LSAs needed. The first algorithm reduces extra LSA numbers efficiently but may lead to many unnecessary link weight changes in some cases. The second algorithm is developed based on the first one for better results. We apply the second algorithm in our scheme with pre-computed paths, where LSA optimization can be finished offline and the computation overhead does not need to be considered during the online stage.

Third, 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. The results show that our scheme can reroute a large flow within 0.5 s. We also evaluate our algorithms with simulations based on measured topologies and synthetic traffic. The maximum link utilization of our scheme is around 120% of the optimal solution for destination address-based flows, and around 110% of the optimal solution for source and destination addresses-based flows. Our optimization mechanisms reduce the extra LSA number and computation time by 87% and 83% respectively for our scheme with pre-computed paths.

The rest of the paper is organized as follows. Section 2 summarizes related work. Section 3 shows an analysis and modeling of large flow on real Internet traffic. We introduce the link weight assignment-based large flow scheduling scheme, formalize the problem, and analyze the complexity in Section 4. Section 5 describes the computation of candidate paths and the algorithm to allocate large flows. Section 6 is devoted to the development of two LSA optimization algorithms. We present our experiments and simulations in Section 7. Section 8 concludes the paper.

2. Related work

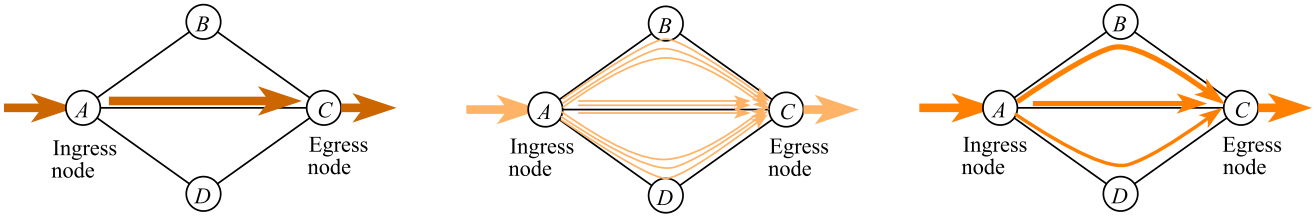
Existing traffic engineering approaches can be classified by either routing technology used or traffic information available.

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 the traditional intra-domain routing protocol, i.e., OSPF, which uses a destination-based forwarding scheme. For instance, Xu et al. [9] 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 [10] are needed to split traffic. Second, there are approaches leveraging MPLS [11] or segment routing [7] to control the traffic forwarding for a given flow. The 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 extremely fine-grained manner [12]. For instance, C.-Y. Hong et al. [13] 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) [14,15], 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 [16]. To address the issue, some alternative approaches choose to use an optimal static routing for arbitrary TMs. Applegate et al. [17] propose 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 a destination-based routing scheme [18]. Another alternative to address the TM problem is to control large flows only [2,8]. 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 [19]. For instance, Al-Fares et al. [8] proposes Hedera 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. Different from existing TE researches based on large flows, we introduce the idea into conventional networks, in which it is not practical to adopt OpenFlow all at once. We note that our approach is not a simple combination of large flow detection and conventional TE scenarios. Particularly, we for the first time analyze and model large flows on the Internet. Besides, we present evidence on that identifying flows by source and destination addresses is fine-grained enough for good TE performance. Previous large flow-based approaches have almost no discussion on designing TE algorithms from the perspective of flow identification. Finally, three algorithms with some theoretical proofs are proposed for allocating large flows in conventional networks.



(a) Destination address. (b) The quintuple of source and destination address, source and destination port, and protocol type. (c) Source and destination addresses.

Fig. 1. An example of large flow scheduling under the different manners of identifying large flows. Subfigure (a) shows a simple TE scenario where there is only one large flow identified by destination address. The traffic load is unbalanced whichever forwarding path is assigned to the large flow because of too coarse-grained control. Subfigure (b) shows that the large flow in (a) can be divided into many small subflows by using the quintuple information. Meanwhile, the traffic load can be balanced well by rerouting these small subflows to multiple paths. However, too many flows for rerouting may induce much overhead such as computation, storage, etc. Subfigure (c) shows that the large flow in (a) can be divided into several subflows by considering source and destination addresses. Meanwhile, good TE performance can also be achieved while no much overhead is introduced.

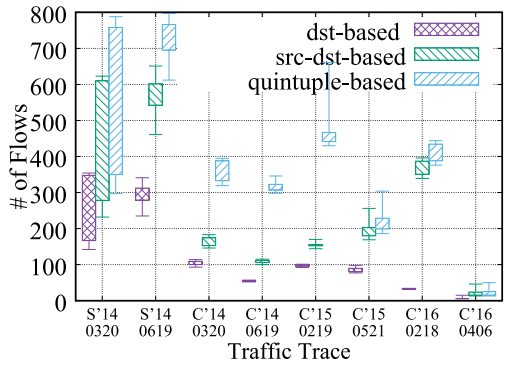


Fig. 2. The least number of flows that compose 35% of total traffic amount from 13:00 to 13:10 for eight traces.

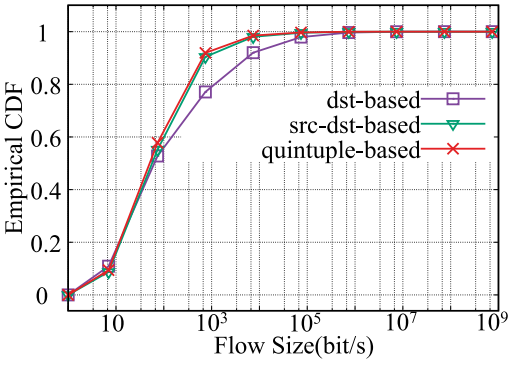


Fig. 3. Empirical CDF of flow size from 13:00 to 13:01 truncated from "C'16 0406".

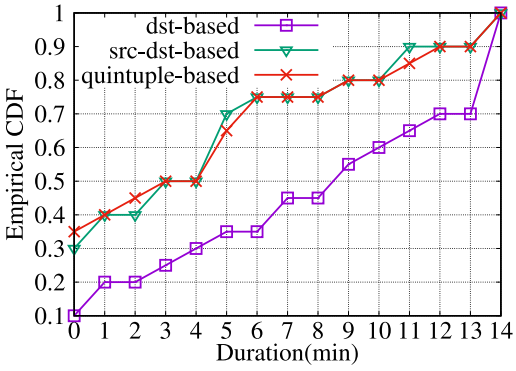


Fig. 4. Empirical CDF of the duration of top 20 largest flows found at 13:00 "C'16 0406".

3. Large flow on Internet: Analysis and modeling

3.1. 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 of 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 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 analyzed by the minute.

Fig. 2 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. 3 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 a log scale, which means that the flow size distribution is at least exponential. Further, a greater portion of flows has a large amount for dst-based flows, compared with src-dst-based flows and quintuple-based flows. Fig. 4 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

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

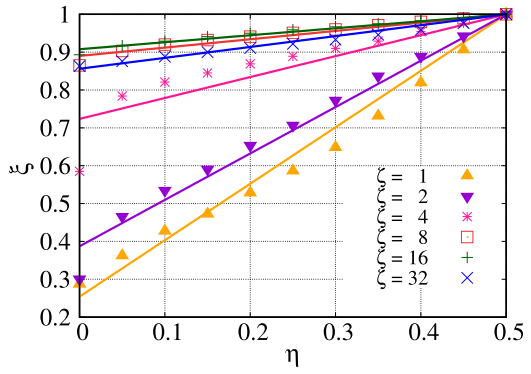


Fig. 5. Empirical probability distributions and their fitting functions for "C'16 0406".

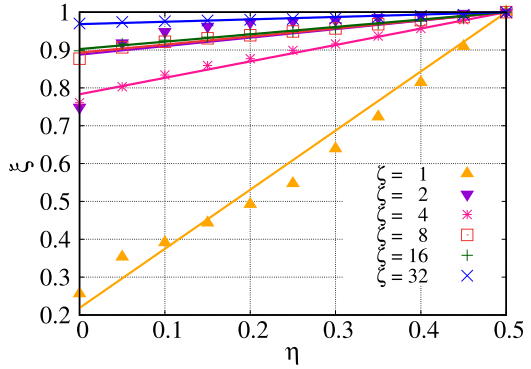


Fig. 6. Empirical probability distributions and their fitting functions for "S'14 0619".

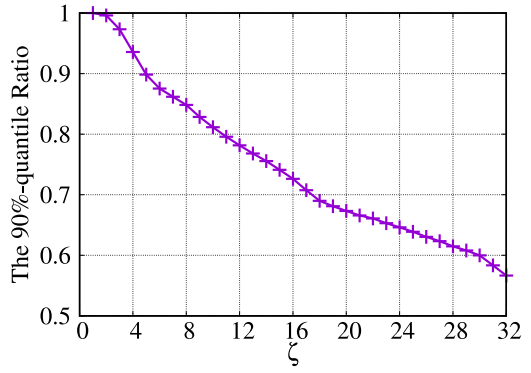


Fig. 7. The 90%-quantile ratio as a function of ζ based on the fitting functions for "C'16 0406".

and the flow amount is still in the top 20. We can see that all 20 flows have a duration of fewer than 14 min. Only 20% of the dst-based flows (i.e., four flows) have a duration less than 1 min, and 40% of the src-dst-based flows and quintuple-based flows have a duration less than 1 min.

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.

3.2. Modeling 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 the 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 a 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 $\zeta \in \{0, 1, \ldots, 32\}$ for IPv4. Let $\langle dst/32, src/\zeta\rangle$ be a srcP-dst-based flow. Particularly, $\langle dst/32, src/0\rangle$ is a dst-based flow. Flow $\langle dst/32, src/\zeta - 1\rangle$ $(\zeta > 0)$ can be split into two flows, both of which can be represented by $\langle dst/32, src/\zeta\rangle$. 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 $\langle dst/32, src/\zeta - 1\rangle$ 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 $F(\eta, \zeta)$, $\zeta \in \{1, 2, \ldots, 32\}$.

To obtain the concrete expressions of $\xi = F(\eta, \zeta)$, we again use the CAIDA traffic traces. We analyze the source addresses of the top 20 dst-based flows in each one minute and obtain the empirical CDFs of splitting ratio η for each $\zeta \in \{1, 2, \ldots, 32\}$. We find that the results from different monitors and periods have some similar features. Figs. 5 and 6 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 the 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. 7 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 a 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 the splitting ratio.

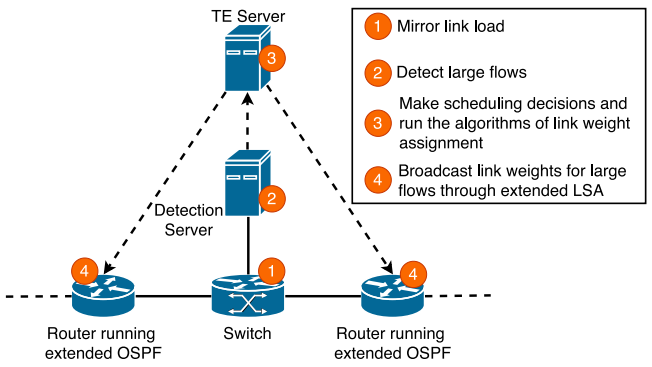


Fig. 8. The overview of our proposed scheme.

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.

4. Link weight assignment-based large flow scheduling

4.1. Overview

In this paper, we focus on intra-domain TE. A node in a network can be a router, a switch, or a Point of Presence (PoP). A PoP is an aggregation of routers/switches which, for example, are located in the same city [17]. 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 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. The overview of our scheme is shown in Fig. 8. In particular, the scheme consists of three components.

4.1.1. 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 cannot count on the OpenFlow counters for per-flow statistics. Fortunately, there has been a set of algorithms developed to detect large flows efficiently [19,20]. Developing a new algorithm is out of the scope of this paper. We study the existing algorithms thoroughly and choose the one most suitable to our problem (considering accuracy, overhead, ease of deployment, etc.).

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. Step 1 and step 2 in Fig. 8 show the process of large flow detection.

4.1.2. 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 [8,21]. However, Internet topologies are not regular, and path computation affects the efficiency of TE schemes greatly. We propose to precompute a limited number of low-stretch candidate paths to make our scheme work efficiently, in Section 5.1. Further, one 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 the randomized algorithm under different granularities of traffic flow, in Section 5.2. In Fig. 8, scheduling decisions are made in step 3.

Note that, the centralized server is different from a typical SDN controller from the perspective of routing computation. In particular, an SDN controller needs to deal with packet-in messages and compute routings for newly coming flows. The routing algorithms only run on the SDN controller. In contrast, the centralized server only keeps a partial control plane. The server helps compute paths only for a few large flows, and the routing information base (RIB) tables for both large flows and other flows are generated on distributed routers, which will be introduced later. Thus, compared with an SDN controller, our server will induce less overhead such as computation, storage, etc. Furthermore, our scheme enhances the flexibility of conventional routing protocols while good scalability and robustness of conventional routing protocols are reserved.

4.1.3. Third, the scheduling decisions need to be executed.

Again, we cannot 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, such a method limits the adaptiveness to topology changes. A dynamic routing which can adapt to topology changes and recover from failures automatically would be more appropriate. To achieve dynamic routing, we can take advantage of the conventional hop-by-hop routing protocols like OSPF. The new routing decisions can be converted to the assignments of link weights which will be broadcast through LSA. Then, each router will be able to compute paths independently for adapting to possible topology changes.

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 [22]. 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 [23]. We formulate our large flow scheduling problem with link weight assignment overhead as a constraint in Section 4.2, and propose two algorithms to reduce the overhead in Section 6. In Fig. 8, step 3 executes the algorithms of link weight assignment, and the related routers broadcast LSA to advertise the link weights in step 4.

Although our system is implemented by extending OSPF, our large flow-based approach can also be applied to other intra-domain protocols like IS-IS. Similar to OSPF, IS-IS is also a kind of link-state protocol

and computes routings by the shortest path first (SPF) algorithm. Thus, our large flow-based algorithms can also be used in IS-IS but there will be some differences in the system implementation.

4.2. Problem formalization

Formally, a network is modeled 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 [24]. 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 [25], 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.

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

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

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

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

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

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

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

$$\forall i \in D :$$

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

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

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

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

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

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

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

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. Eqs. (5) and (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 $w^i_{u,v}$.

4.3. 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 [26]. 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 the 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 cannot 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 [27].

According to Theorem 1, it is not easy to solve problem P1 directly. Thus we take a heuristic to solve the problem. In particular, we first compute the routing for large flows, without the constraint on link weight assignment overhead. We describe rerouting path computation and large flow allocation in Sections 5.1 and 5.2 respectively. Then, we try to distribute the rerouting paths with a minimal number of link weight assignments. We describe the two proposed algorithms in Section 6.

5. Rerouting path computation for large flows

In this section, we compute rerouting paths for large flows, without the constraint on link weight assignment overhead. 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. In particular, we first compute a set of candidate paths and splitting weights over these paths. Then we propose a large flow allocation algorithm to randomly allocate each large flow to a path by taking the splitting weights as the probabilities.

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

5.1. Computation of candidate path set

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. We define the large flows with the same ingress and egress node pair as an *integrated flow*. By solving the slacked MCF problem with the integrated flows as the traffic demands (background traffic is not rerouted) and computing the augmenting paths, we can obtain the set of candidate paths and the splitting weights over them. A splitting weight of a path means the traffic amount of large flows passing the path divided by the traffic amount of the integrated flow.

Note that, there is no path limitation in the MCF problem, i.e., any possible path connecting the ingress node and the egress node can be used as a candidate path in the solution as well as the objective value can be minimized. When the input (i.e., large flows) changes, the MCF problem can be solved again to obtain a new set of candidate paths with splitting weights. Therefore, the optimal objective value can always be reached theoretically for various inputs.

However, there are two problems in such a method. First, candidate paths need to be computed at every TE interval as well as the LSA optimization upon these paths, which is time-consuming and slows down the flow scheduling. Further, the time to solve an MCF problem increases greatly as the increment of the topology scale. In a worse case, the solving process may have not been finished before the end of some large flows, which leads to significant TE performance loss. Second, the obtained candidate paths may have a long stretch, i.e., too many hops are needed to reach the egress node. Besides network performance degradation like long packet delay, path stretch also increases the number of extra LSA because we may need to modify many link weights to enable the rerouting path for a large flow.

To address the issues, we propose to compute several candidate paths for each pair of ingress and egress nodes in advance, which has been studied in many previous works [15,28]. In particular, for any pair of ingress node and egress node, we compute K (e.g., 20) candidate paths which will be used to carry the traffic originating from the ingress node and going to the egress node. Then we can formulate an MCF problem with path limitations. In such a design, we only need to obtain the splitting weights over these pre-computed candidate paths. By precomputing a limited number of candidate paths, we can accelerate the solving process of the MCF problem greatly because of problem simplification. For example, the number of decision variables in an MCF problem with path limitations is $O(|V|^2 \cdot K)$, which is smaller than that of an MCF problem without path limitations, i.e., $O(|V|^2 \cdot |E|)$. By setting a large enough value of K, the pre-computed candidate paths can have the potential to accommodate various inputs well although under path limitations. In addition, LSA optimization for each candidate path can be finished offline, which saves much time for online flow scheduling.

To reduce path stretch, we need to limit the length of each candidate path. In this paper, we compute K-shortest paths for each node pair using Yen's algorithm [29]. By taking hop numbers as the cost of a path, we can limit the length of paths directly, and reduce the cost of link weight assignment at the same time. The computation complexity of Yen's algorithm is $O(K \cdot |V|^3)$ which is relatively time-consuming especially for large-scale topologies. Fortunately, we only need to execute the algorithm once at the beginning of our system as long as the network topology keeps unchanged. Our simulation results show an amazing improvement in the efficiency of our scheme by precomputing a limited number of low-stretch paths.

5.2. Large flow randomized allocation

Next, we describe the large flow allocation algorithm and give some theoretical results on the TE performance bounds.

Formally, let D_j be the amount of an integrated flow. We solve the MCF problem with the integrated flows as traffic demands. 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. The candidate paths can be obtained from the solution by computing the augmenting paths similar to [30], and we can get the probability to choose each path at the same time. For each large flow, we randomly select a candidate path as the large flow's rerouting path. We call the algorithm large flow randomized allocation (LFRA).

It can be observed that LFRA is reduced to randomized rounding [30] 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 [31], LFRA performs better when the number of large flows becomes greater. Intuitively, the traffic is more likely to be spread on the rerouting 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 $x_1, x_2, \ldots, x_n$. 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.

On the one hand,

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

Now let, $\gamma = (1+\Delta) \cdot np$, and $\Delta > 0$, (11)

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

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

On the other hand,

Similarly,

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

Now let, $\gamma = (1-\Delta) \cdot np$, and $0 < \Delta < 1$, (14)

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

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

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

Then we have gotten two inequalities,

$$Pr(Y_l \ge (1+\Delta) \cdot Dp) \le e^{-\frac{\Delta^2 np}{3}}, \quad \text{if } \Delta > 0,$$

$$Pr(Y_l \le (1-\Delta) \cdot Dp) \le e^{-\frac{\Delta^2 np}{2}}, \quad \text{if } 0 < \Delta < 1.$$

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

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

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

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

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

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 $D_j (j = 1, 2, \dots, m)$. 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,* $Pr(\Delta_1 \le \beta, \Delta_2 \le \beta, \dots \Delta_m \le \beta) \ge \prod_{j=1}^{m}(1 - 2e^{-\frac{\beta^2 n_j p_j}{3}})$ *for* $0 < \beta < 1$, *and* $Pr(\Delta_1 \le \beta, \Delta_2 \le \beta, \dots \Delta_m \le \beta) \ge \prod_{j=1}^{m}(1 - e^{-\frac{\beta^2 n_j p_j}{3}})$ *for* $\beta \ge 1$.

Proof. According to Eqs. (17) (18) we obtain

$$Pr(\Delta_j \le \beta) \ge 1 - 2e^{-\frac{\beta^2 n_j p_j}{3}}, \text{ for } 0 < \beta < 1,\ j = 1, \dots, m.$$

$$Pr(\Delta_j \le \beta) \ge 1 - e^{-\frac{\beta^2 n_j p_j}{3}}, \text{ for } \beta \ge 1,\ j = 1, \dots, m.$$

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.

$$Pr(\Delta_1 \le \beta, \Delta_2 \le \beta, \dots \Delta_m \le \beta) \ge \prod_{j=1}^{m}(1 - 2e^{-\frac{\beta^2 n_j p_j}{3}}),$$

for $0 < \beta < 1$,

Similarly according to Eq. (18) we obtain

$$Pr(\Delta_1 \le \beta, \Delta_2 \le \beta, \dots \Delta_m \le \beta) \ge \prod_{j=1}^{m}(1 - e^{-\frac{\beta^2 n_j p_j}{3}}),$$

for $\beta \ge 1$. ■

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*

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

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

$$Pr(Y_l \le (1 + \beta)Y^*) \ge Pr(Y_l \le (1 + \beta)Y_l^*)$$

Through the proof of Theorem 3, we can get

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

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

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

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

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

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

Theorem 4 tells us that, the probability of obtaining a good TE performance increases when n_j for any integrated flow increases. Therefore, when we identify a flow by more match fields, more large flows may be obtained for a given integrated flow, and a good TE performance is more likely to be achieved.

6. Link state advertisement optimization

In this section, we consider the problem of changing link weights for the large flows whose rerouting paths have been decided 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.

Two heuristic algorithms are developed to optimize link state advertisement. The first one runs efficiently and reduces many extra LSAs. Due to its efficiency, it is suitable to our scheme where candidate path computation and LSA optimization are executed online, i.e., at each TE interval. The second one improves the effectiveness of LSA optimization further. It is suitable to our scheme where candidate paths are precomputed and LSA optimization can be finished offline, i.e., at the beginning of our system. Therefore, the computation overhead of the second algorithm can be ignored when our scheme works online. Since the link weights for each large flow is independent, we consider one target path in our algorithm development.

6.1. LSA optimization algorithm

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 described in Section 4.3. Because there may be a large number of shortcuts, e.g., the shortest path, the 2nd shortest path, etc., it is time-consuming to find and eliminate all these shortcuts so as to increase their path weights by increasing a few link weights (See a discussion in Section 6.2). 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 independence is very important here. Without link independence, 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. 9 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.

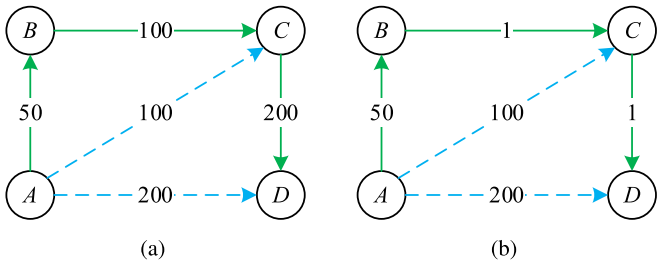


Fig. 9. An example of reducing link weights. (a) Original topology. The solid lines indicate the target path, and the dashed lines are shortcuts. (b) A correct solution where two link weights are reduced and the target path becomes the shortest path now.

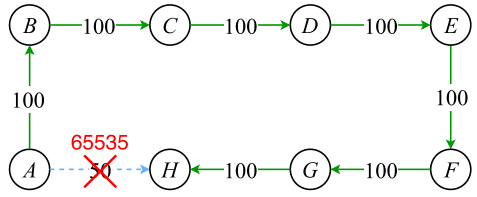


Fig. 10. An example of increasing link weights. The solid lines indicate the target path and the dashed line is a shortcut.

Algorithm 1 LSA-Opt1()

Input: P, t, W;
Output: New matrix of link weights W^*;
1: $W^* = W$
2: **for** u *in* $P.reverse()$ **do**
3: **if** $P(u,t)$ *has no shortcuts* **then**
4: *continue*
5: **for** v *in* $P(u,t)$ **do**
6: **while** $w(P(u,v),W^*) >= w(shor(u,v),W)$ **do**
7: $e = maxweight(P(u,v),W^*)$
8: $W^*(e) = 1$
9: **return** W^*

We develop the LSA-Opt1 algorithm as shown in Algorithm 1. The inputs of LSA-Opt1 include target path P which is from node s to node t, and normal link weight matrix W. The output is new link weight matrix W^*. Step 1 initializes W^*. The outer loop from Step 2 to Step 8 visits the nodes in P in the reverse order, and the inner loop from Step 5 to Step 8 visits the nodes in $P(u,t)$ which is the sub-path of P from node u to node t, an then we obtain sub-path $P(u,v)$. Note that each shortcut of $P(u,v)$ is guaranteed to be link independent with $P(u,v)$ due to the order of visiting $P(u,v)$. Step 6 to 8 deal with the shortest shortcut of $P(u,v)$, by changing the maximum link weight to 1 until $P(u,v)$ is shorter than the shortest shortcut. In particular, $w(P(u,v),W^*)$ returns the weight of $P(u,v)$ with W^*, and $w(shor(u,v),W)$ returns the weight of the link independent shortest shortcut $shor(u,v)$ with W. Note that $w(shor(u,v),W)$ can be computed in advance. Step 7 finds link e which has the maximum weight in $P(u,v)$ with W^*. Step 8 changes the weight of link e. We can keep a sorted list of link weights on $P(u,v)$ in each iteration of Step 5. So the complexity of Step 7 can be $O(1)$, and the complexity of the LSA-Opt1 algorithm is $O(|V|^3)$.

6.2. Improved LSA optimization algorithm

As described above, LSA-Opt1 only considers decreasing link weights along the target path, which, however, may lead to many unnecessary link weight changes in some cases. Fig. 10 shows an example of such cases. According to LSA-Opt1, all weights of links (i.e., total seven) along the target path (solid lines) need to be reduced. However, we can eliminate the shortcut (dashed line) by changing only one link weight, i.e., increasing the weight of link (A,H) to a large enough value, e.g., 65,535. We can see that increasing path weights of shortcuts is sometimes more effective on LSA optimization. However, as described in the above subsection, it is not easy to find and eliminate all shortcuts efficiently by increasing a few link weights. In this subsection, we take a combination of two methods to overcome the challenge.

Next, we first describe the method that increases the least link weights to eliminate all the shortcuts of a target path. Then we show the method that reduces the overhead of finding all the shortcuts. Finally, we present our improved LSA optimization algorithm.

Given a target path, we can calculate all the shortcuts that are shorter than a target path by Yen's algorithm based on normal link weights. We define an independent link as the link on a shortcut but not on the target path. To eliminate all the shortcuts and make the target path be the shortest, the easiest way is to increase the weight of an independent link on each shortcut to a large enough value. To further reduce the number of link weight changes, we make the following observation. Some shortcuts may share a common independent link. By increasing the weight of the common link to a large enough value, all the related shortcuts can be eliminated together. We consider the independent link set where the weight of each independent link can be increased to eliminate a set of shortcuts. The problem is to find the least number of independent links so that all the shortcuts can be eliminated by increasing these links' weights. By considering each independent link as a set of related shortcuts and all shortcuts as the universe, this problem is equivalent to Set Covering Problem, which is NP-hard [32]. In this paper, we take a greedy algorithm to solve the problem: at each step, select the independent link which can eliminate the most shortcuts, until all the shortcuts have been eliminated.

However, the time of computing all the shortcuts of a target path can be unacceptable for large topologies even when LSA optimization can be executed offline in our scheme with pre-computed paths. We observe that the shortest shortcut is computed firstly, then the 2nd shortest shortcut, etc. If we eliminate the shortest shortcut firstly by increasing the weight of one independent link, some of the other shortcuts may be eliminated at the same time, and the number of shortcuts that need to be computed can be reduced. Thus, we propose to compute and eliminate shortcuts in batch. In particular, at each step, we compute and eliminate a limited number of shortcuts until there is no shortcut. By doing this, we can reduce the execution of Yen's algorithm greatly.

By combining the above two methods, we develop the algorithm as shown in Algorithm 2. The inputs of algorithm 2 include target path P, normal link weight matrix W, and the batch size of shortcuts to be eliminated B. The output is a new link weight matrix W^*. Step 1 initializes W^*. The outer loop from Step 2 to Step 8 eliminates shortcuts incrementally until there are no shortcuts. Step 3 computes at most B shortcuts, i.e., P_{shr}, through Yen's algorithm based on W^*. Step 4 obtains independent link set of P_{shr}. The inner loop from Step 5 to

Algorithm 2

Input: P, W, B;
Output: New matrix of link weights W^*;
1: $W^* = W$
2: **while** *There exist shortcuts* **do**
3: $P_{shr} = Yen's(W^*, B)$
4: *Compute independent link set* L
5: **while** *Not all shortcuts in* P_{shr} *are eliminated* **do**
6: $e = maxcover(L)$
7: $W^*(e) = 65535$
8: $L.remove(e)$
9: **return** W^*

Algorithm 3 LSA-Opt2()

Input: P, t, W, B;
Output: New matrix of link weights W^*;
1: $W_1^* = \text{LSA-Opt1}(P, t, W)$
2: $W_2^* = \text{Algorithm2}(P, W, B)$
3: $W^* = bestof(W_1^*, W_2^*)$
4: **return** W^*

Step 8 executes the greedy algorithm to eliminate the shortcuts by increasing the weight of the link e. In particular, $maxcover(L)$ returns the independent link which covers the largest number of uneliminated shortcuts.

Note that, Algorithm 2 gets better results than LSA-Opt1 in some cases, and vice versa. Thus we combine both of the algorithms to improve the effectiveness of LSA optimization, and propose LSA-Opt2 which chooses the best result of LSA-Opt1 and Algorithm 2. The improved LSA optimization algorithm LSA-Opt2 is shown in Algorithm 3.

Obviously, LSA-Opt2 is more time-consuming than LSA-Opt1. However, when candidate paths are computed in advance, LSA optimization can be finished offline and does not need to be executed at every TE interval. By default, we apply LSA-Opt1 in our scheme without pre-computed candidate paths, and apply LSA-Opt2 in our scheme with pre-computed candidate paths.

Discussion: LSA-Opt2 cannot get optimal solutions all the time. An optimal solution should consider link weight increase and reduction at the same time during optimization instead of considering only one of the two aspects. However, it is difficult to develop an optimal algorithm because of the problem's complexity. We leave this as our future work.

6.3. LSA broadcast strategy

To change a link weight for a large flow, extended LSA will be broadcast through the whole network. Therefore, reducing one link weight change will reduce lots of LSA messages during the broadcast, which is much more important for large topologies.

By executing LSA optimization algorithms for the rerouting paths, we can get the policy of LSA flooding to enable the rerouting paths for each large flow. Note that, some large flows may be assigned to the same rerouting path, and these large flows can be rerouted through flooding the same set of link weight changes for the path. Thus during LSA flooding, each LSA message can be used for enabling one target path but carry multiple source address and destination address entries (i.e., multiple large flows assigned to the path). This will reduce the cost of LSA flooding further.

7. Performance evaluation

7.1. Experiments

7.1.1. 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 [19], 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 [33]. Then, we invoke the LFRA algorithm to compute the rerouting path of each large flow and use the LSA-Opt1 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 5000 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.

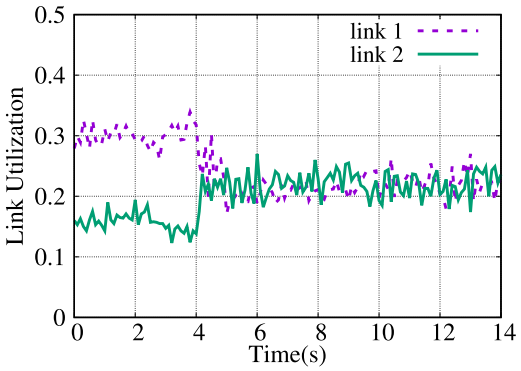


Fig. 11. The evaluation of time for detecting and rerouting large flows.

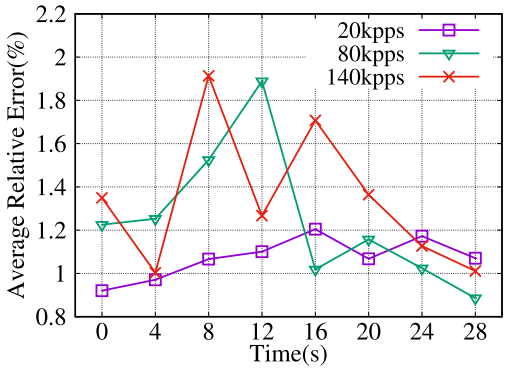


Fig. 12. Average relative errors of measuring the flow size of top 10 flows at different traffic replay rates.

7.1.2. 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 $t = 0$ s, 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 $t = 4$ s, the detecting servers report traffic information to the centralized TE server. Fig. 11 shows the result. We can observe that some large flows are shifted to another path which passes link 2 in 0.5 s. 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.

Table 1
The rocketfuel topologies.

AS No.	Name	No.of nodes	No.of links
1221	ASN-TELSTRA	104	153
1239	SprintLink	315	972
1755	STUPI AB	87	161
3257	TINET-BACKBONE	161	328
3967	Nationstar Mortgage	79	147
6461	Metromedia	138	374

We also evaluate the accuracy of large flow detection, by recording the average relative errors of the flow size of the top 10 large flows, using CAIDA traffic data. Fig. 12 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 rates are usually larger and fluctuate more fiercely. This is because a larger rate incurs more packet losses, and the relative error is related to packet loss ratio besides the errors incurred by the algorithm itself.

7.2. Simulations

7.2.1. Simulation setup

We use six measured topologies provided by Rocketfuel project [34] in our simulations as shown in Table 1. The link capacity is determined by the degrees of the two nodes adjacent to the link, following [35]. In particular, the link capacity is set to 9953 Mbps if the two adjacent nodes both have a degree greater than 5; otherwise, the link capacity is set to 2488 Mbps. The link weight is set as the suggested value [34] 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 10% to 15% 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 150 Mbps and 200 Mbps. Third, we divide each dst-based large flow into srcP-dst-based flows using the probabilistic splitting functions mentioned in Section 3.2, and we assume all srcP-dst-based flows have a flow size which is at least 1 Mbps. All the nodes use the 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. By default, we use $K = 20$ candidate paths for each integrated flow.[5]

We compare the performance of our scheme with/without pre-computed candidate paths, and with both dst-based flows and srcP-dst-based flows. We also evaluate the effectiveness of LFRA algorithm and two LSA optimization algorithms. For comparison, we compare performance with conventional OSPF[6] and the optimal solution that can only be achieved by fractional routing. We first evaluate the effectiveness of our scheme, 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*. Note that, the value of performance ratio should always be no smaller than 1.0, and the performance ratio equaling 1.0 means the optimal solution. Second, we evaluate the number of extra LSA broadcasts needed to enable the rerouting paths of the large flows. One link weight change will lead to one extra LSA broadcast. We compare our scheme to an OpenFlow-like scheme, which configures each router along a target path. Third, we evaluate the computation time of our scheme under different settings. Fourth, we evaluate the effect on our scheme as the source address prefix length of srcP-dst-based flow increases. Finally, we evaluate the effect on our scheme as the number of candidate paths for each integrated flow (i.e., K) increases. All simulations are performed on a PC which has 4-core Intel Core i7-4790 3.6 GHz CPU and 8 GB memory. We use Gurobi to solve MCF problems with or without path limitations. Each point in our results is the average of 20 independent simulations.

[5] The number of candidate paths for each integrated flow can be less than 20 if there are less than 20 paths in total.

[6] Traffic will be split evenly over equal cost multiple paths, which is supported by OSPF protocol.

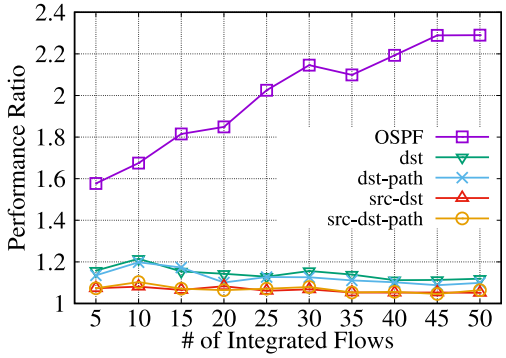


Fig. 13. Performance ratio as a function of integrated flow number, in the topology of AS 1755.

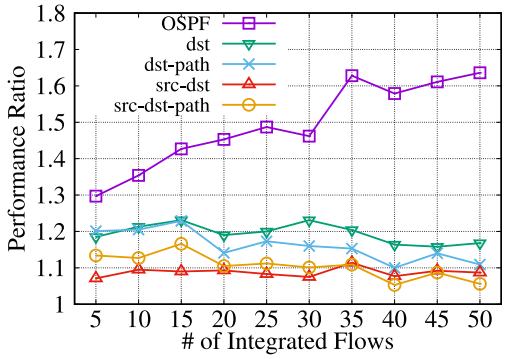


Fig. 14. Performance ratio as a function of integrated flow number, in the topology of AS 6461.

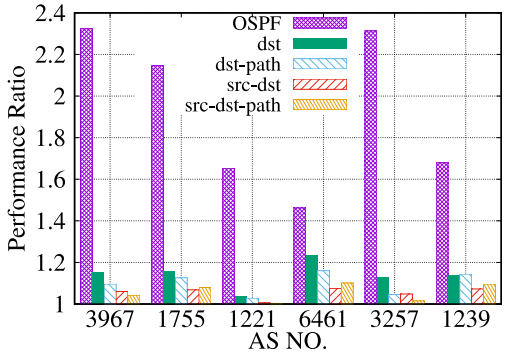


Fig. 15. Performance ratio in different topologies.

7.2.2. Simulation results

Performance ratio. Fig. 13 shows the performance ratio as a function of integrated flow number in the topology of AS 1755. For brevity, we use "dst" and "src-dst" to represent the results of dst-base flows and

src-dst-based flows *without* pre-computed candidate paths, respectively. We use "dst-path" and "src-dst-path" to represent the results of dst-base flows and src-dst-based flows *with* pre-computed candidate paths, respectively. We can see that for both dst-based 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 congestion as the integrated flow number increases. We find that the performance ratio of "src-dst" is very close to 1.0 for src-dst-based flows, which is averagely 7% less than the performance ratio of "dst" for dst-based flows. This implies that by identifying flows with the only source address and destination address, a near-optimal TE performance can be achieved. When there are 30 integrated flows, "src-dst" reduces the performance ratio by 50% compared to conventional OSPF routing. We also find that our schemes with pre-computed paths (i.e., "dst-path" and "src-dst-path") achieve similar performance to those without pre-computed path (i.e., "dst" and "src-dst"). This is because a relatively large number (i.e., 20) of candidate paths for each integrated flow is enough to balance the traffic load well. For example, when there are 10 integrated flows, the performance of "dst", "dst-path", "src-dst", and "src-dst-path" are 1.22, 1.20, 1.08 and 1.10, respectively.

Fig. 14 shows similar results to those in Fig. 13. The main difference from Fig. 13 is that "dst-path" and "src-dst-path" perform even better than "dst" and "src-dst" respectively as the increment of integrated flow number. When there are 50 integrated flows, "src-dst-path" reduces the performance ratio by 10%, 5%, and 3% compared to "dst", "dst-path", and "src-dst", respectively. This is because the optimal solution of the MCF problem with path limitations is more likely to split traffic over fewer paths than that of the MCF problem without path limitations, and then LFRA is less likely to make bad decisions.

The results in Figs. 13 and 14 are not special. Fig. 15 shows the performance ratio in all six topologies. We can observe that our scheme always has good performance, for both types of flow. Our schemes with pre-computed paths achieve performance near to those without path limitations. "src-dst-path" reduces the performance ratio by 35% to 56% compared to OSPF.

Cost of extra LSA broadcasts. Fig. 16 shows the number of extra LSA broadcasts as a function of integrated flow number in the topology of AS 1755. We can see that the number of extra LSA broadcasts increases as the number of integrated flow gets larger. This is because when there are many large flows, more rerouting paths will need to be enabled. The extra LSA number needed by src-dst-based flows is greater than that of dst-based flows because the number of src-dst-based flows is larger than that of dst-based flows. However, LSA-Opt1 reduces the extra LSA by about 50%, so the extra LSA broadcast number of "src-dst" becomes smaller than 300. What is more, by taking K-shortest paths as the candidate paths, we reduce extra LSA broadcast numbers further. We can find "src-dst-path" brings a cost very close to "dst-path", and the extra LSA broadcast number of "src-dst-path" is always smaller than 100, which is acceptable for current network size. The results imply that a combination of LSA optimization and low-stretch path computation can reduce the cost of extra LSA broadcasts greatly. Similar results can be found in Fig. 17 which shows the results in the topology of AS 6461.

Fig. 18 shows the results in all six topologies, which are similar to those in both Figs. 16 and 17. For brevity, we only show the results for src-dst-based flows. We can see that a combination of LSA optimization and path precomputation reduces LSA cost by averagely 87%, and makes the cost no larger than 61 for all six topologies.

Computation time. We evaluate the computation time of our scheme. A shorter computation time means faster traffic scheduling. The computation time of "dst" and "src-dst" includes solving the MCF problem, large flow randomly allocation, and LSA optimization (i.e., LSA-Opt1).

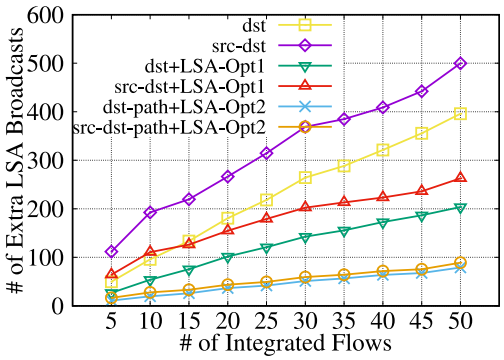


Fig. 16. Extra LSA broadcast number as a function of integrated flow number, in the topology of AS 1755.

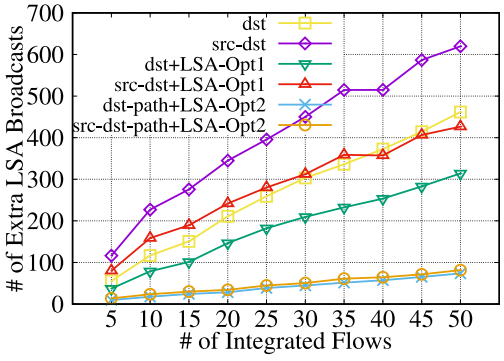


Fig. 17. Extra LSA broadcast number as a function of integrated flow number, in the topology of AS 6461.

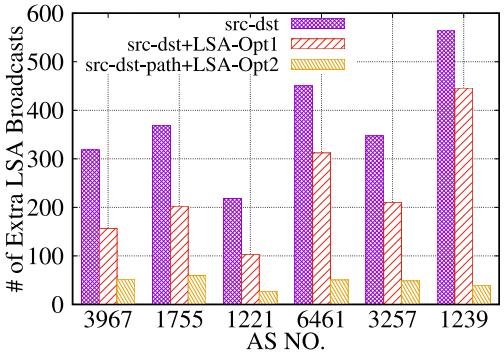


Fig. 18. Extra LSA broadcast number in different topologies.

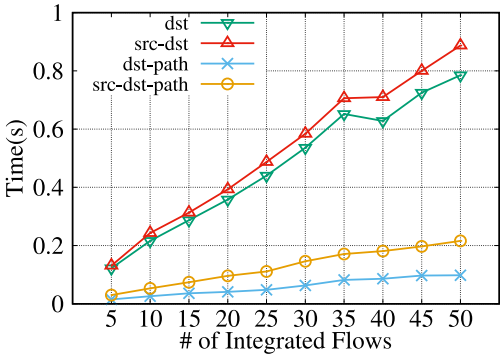


Fig. 19. Computation time as a function of integrated flow number, in the topology of AS 1755.

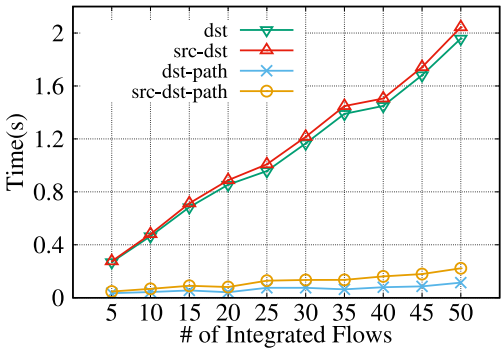


Fig. 20. Computation time as a function of integrated flow number, in the topology of AS 6461.

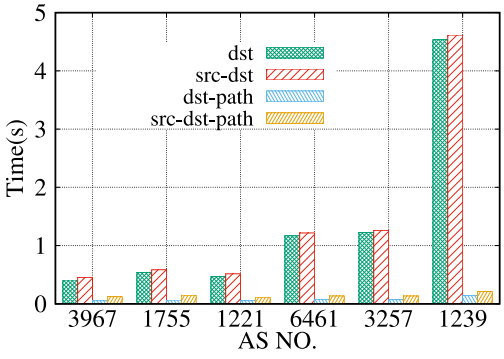


Fig. 21. Computation time in different topologies.

The computation time of "dst-path" and "src-dst-path" includes solving the MCF problem with path limitations and large flow randomly allocation, and LSA optimization is not included because it needs to be executed only once at the beginning of our system.

Fig. 19 shows the computation time of our scheme. We see that the time is increasing linearly with the integrated flow number. The computation time for src-dst-flows is slightly longer than that for dst-based flows because there are more large flows needed to be rerouted for src-dst-flows. We also find that our scheme with pre-computed paths reduces computation time greatly for both types of flows and has a slower increment as the integrated flow number. For example, when there are 50 integrated flows, the computation time of "src-dst-path" is only around 0.22 s, and the computation time is reduced by around 72% compared to "src-dst". The results imply that precomputing candidate paths are important for saving computation time and provide a faster reaction to network traffic. Similar results can be found in Fig. 20 where the computation time of "src-dst-path" is limited to around 0.22 s, too.

Fig. 21 shows the results in all six topologies. We find that computation time is shorter than 5 s for all the topologies, and precomputing candidate paths reduces computation time significantly especially for large topologies. "src-dst-path" reduces computation time by averagely 83% compared to "src-dst" with respect to all six topologies. The results also show that the computation time of both "dst-path" and "src-dst-path" increases much more slowly with the topology scale than the schemes without path limitations. The computation time of "src-dst-path" is only around 0.2 s for the largest topology.

Effect of source address prefix length. Next, we evaluate the effect of source address prefix length in "srcP-dst" and "srcP-dst-path". Fig. 22 shows the performance ratio as a function of source address prefix length in the topology of AS 1755. We can see the performance ratio decreases as source address prefix length becomes larger and is lower than 1.08 when prefix length is 32 bits. This is because a longer source

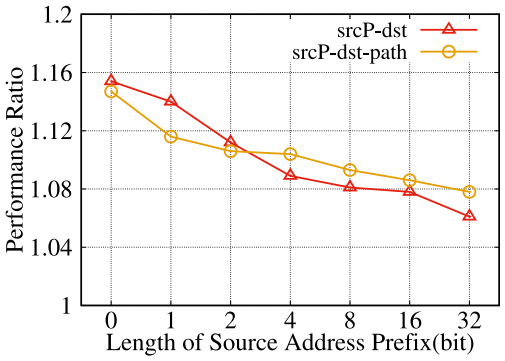


Fig. 22. Performance ratio as a function of source address prefix length, in the topology of AS 1755.

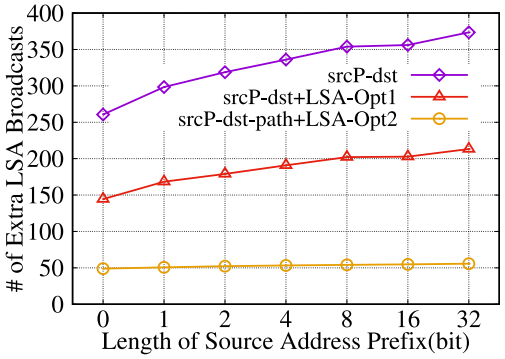


Fig. 23. Extra LSA broadcast number as a function of source address prefix length, in the topology of AS 1755.

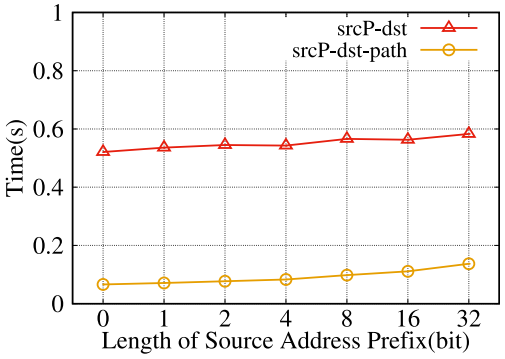


Fig. 24. Computation time as a function of source address prefix length, in the topology of AS 1755.

address prefix length can split a dst-based flow into more srcP-dst-based flows. According to the analysis of LFRA, good performance is more likely to be achieved with more flows for scheduling. We also find our two schemes achieve similar performance for different lengths of source address prefix, and "srcP-dst" slightly outperforms "src-dst-path" for the prefix lengths larger than 2 bits. The results mean that better performance can be achieved by matching a longer length of source address prefix.

Fig. 23 shows the number of extra LSA broadcasts as a function of source address prefix length in the topology of AS 1755. We can see the result value increases slowly with the lengths of source address prefix, and "srcP-dst-path" with LSA-Opt2 leads to an ignorable increment of broadcast number. When the source prefix length is set to 32 bits, the broadcast number of "srcP-dst-path" with LSA-Opt2 is only 56, which is only 7 more than that for the source prefix length

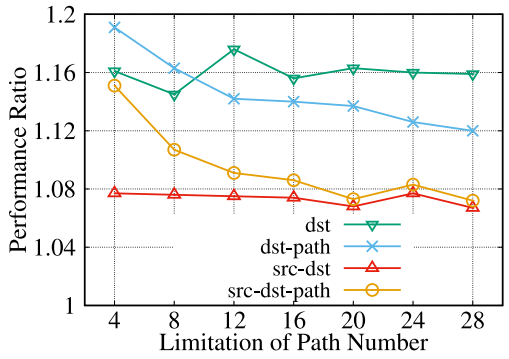


Fig. 25. Performance ratio as a function of path number, in the topology of AS 1755.

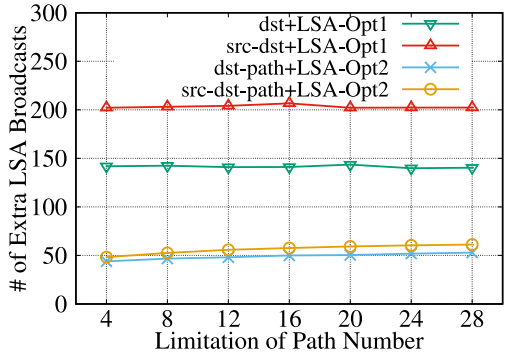


Fig. 26. Extra LSA broadcast number as a function of path number, in the topology of AS 1755.

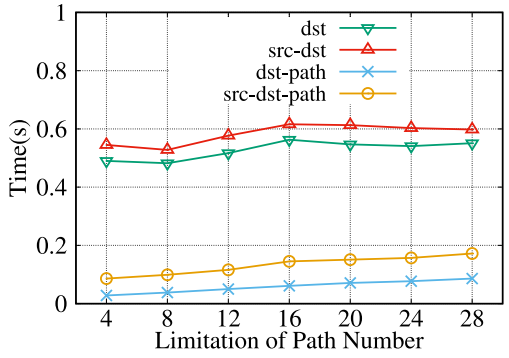


Fig. 27. Computation time as a function of path number, in the topology of AS 1755.

of 0 bit, and is 85% and 73% less than that of "srcP-dst" and "srcP-dst" with LSA-Opt1, respectively. The results imply that increasing the length of source address prefix introduces little broadcast number because of both mechanisms, i.e., LSA optimization and candidate path precomputing.

Fig. 24 shows the computation time as a function of source address prefix length in the topology of AS 1755. The computation time increases slowly as the increment of the length of source address prefix. The result value for the length of 32 bits is only 0.07 (0.06) seconds more than that for the length of 0 bit under the scheme of "src-dst-path" ("src-dst"). "src-dst-path" limits the computation time smaller than 0.14 s for all the cases. The results imply that increasing the length of source address prefix does not introduce too much computation burden.

Effect of path number limitation. Finally, we evaluate the effect of path number limitation, i.e., K. Fig. 25 shows the performance ratio as a function of path number limitation in the topology of AS 1755. We also present the result values of "dst" and "src-dst" for comparison, and these results fluctuate slightly because of the feature of LFRA. We can see the performance ratio of "dst-path" and "src-dst-path" gets smaller when more paths can be used for each integrated flow. When the limitation of path number is 20, "src-dst-path" makes very close performance to "src-dst". The results show that better performance can be achieved by increasing the path number properly.

Fig. 26 shows the number of extra LSA broadcasts as a function of path number limitation in the topology of AS 1755. We can find the result value increases as the increment of path number limitation for both of "dst-path" and "src-dst-path", and "dst-path" leads to slightly smaller result value than "src-dst-path". Even so, the result value of "src-dst-path" is only 61 for the path number limitation of 28. The results mean that increasing path number limitation will lead to only a bit more extra LSA broadcasts.

Fig. 27 shows the computation time as a function of path number limitation in the topology of AS 1755. We can find the computation time gets larger as the increment of path number limitation, but the computation time of "src-dst-path" is smaller than 0.2 s for the largest path number limitation, i.e., 28. The results show that increasing path number limitation will lead to only a little longer computation time.

8. Conclusion

In this paper, we studied flow-level traffic engineering by scheduling a few large flows in conventional routing systems. We did an analysis and modeling of real traffic traces. We formalized the link weight assignment-based large flow scheduling problem and developed heuristics to solve it. We precomputed a set of low-stretch candidate paths to reduce the overhead of computation and LSA. We developed an algorithm to allocate large flows randomly and two algorithms to optimize LSA broadcast for our schemes with and without precomputed candidate paths, respectively. We conducted experiments and extensive simulations to validate the effectiveness and efficiency of our scheme. Our research data would be available online after publication [36].

In practice, the deployment of our system will cause some overhead including the overhead of upgrading the software on routers and the overhead of deploying the servers. We note that there is no need to replace a large number of physical routing devices, so there will not be too much overhead introduced. Furthermore, our extended routing protocol is compatible with the original routing protocol, and thus incremental deployment can be taken to reduce the negative impact of software updates. The investigation of efficient incremental deployment is an interesting future direction.

CRediT authorship contribution statement

Nan Geng: Conceptualization, Methodology, Software, Validation, Formal analysis, Writing - original draft. **Yuan Yang:** Conceptualization, Methodology, Writing - review & editing. **Mingwei Xu:** Conceptualization, Writing - review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported by the National Key R&D Program of China (2018YFB1800302), the National Natural Science Foundation of China (61625203, 61832013, and 61872209), and the Independent Scientific Research Project of NUDT (ZZKY-ZX-03-02-02). We thank the reviewers for their valuable comments.

References

[1] A. Mendiola, J. Astorga, E. Jacob, M. Higuero, A survey on the contributions of software-defined networking to traffic engineering, IEEE Commun. Surv. Tutor. (2017).

[2] N. Geng, Y. Yang, M. Xu, Flow-level traffic engineering in conventional networks with hop-by-hop routing, in: IEEE/ACM IWQoS, 2018, pp. 1–10.

[3] Global networking trends report, 2020, https://www.cisco.com/c/dam/m/en_us/solutions/enterprise-networks/networking-report/files/GLBL-ENG_NB-06_0_NA_RPT_PDF_MOFU-no-NetworkingTrendsReport-NB_rpten018612_5.pdf.

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

[5] S. Yang, M. Xu, D. Wang, G. Bayzelon, J. Wu, Scalable forwarding tables for supporting flexible policies in enterprise networks, in: IEEE INFOCOM, 2014, pp. 208–216.

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

[7] R. Hartert, S. Vissicchio, P. Schaus, O. Bonaventure, C. Filsfils, T. Telkamp, P. Francois, A declarative and expressive approach to control forwarding paths in carrier-grade networks, ACM SIGCOMM CCR 45 (4) (2015) 15–28.

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

[9] K. Xu, M. Shen, H. Liu, J. Liu, F. Li, T. Li, Achieving optimal traffic engineering using a generalized routing framework, IEEE/ACM TPDS 27 (1) (2016) 51–65.

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

[11] D.O. Awduche, MPLS and traffic engineering in IP networks, IEEE Commun. Mag. 37 (12) (1999) 42–47.

[12] S. Agarwal, M. Kodialam, T. Lakshman, Traffic engineering in software defined networks, in: IEEE INFOCOM, 2013, pp. 2211–2219.

[13] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, R. Wattenhofer, Achieving high utilization with software-driven WAN, ACM SIGCOMM CCR 43 (4) (2013) 15–26.

[14] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, C. Diot, Traffic matrix estimation: Existing techniques and new directions, ACM SIGCOMM CCR 32 (4) (2002) 161–174.

[15] P. Kumar, Y. Yuan, C. Yu, N. Foster, R. Kleinberg, P. Lapukhov, C.L. Lim, R. Soulé, Semi-oblivious traffic engineering: The road not taken, in: USENIX NSDI, 2018.

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

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

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

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

[20] C. Hu, B. Liu, H. Zhao, K. Chen, Y. Chen, C. Wu, Y. Cheng, Disco: Memory efficient and accurate flow statistics for network measurement, in: IEEE ICDCS, 2010, pp. 665–674.

[21] A.R. Curtis, W. Kim, P. Yalagandula, Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection, in: IEEE INFOCOM, 2011, pp. 1629–1637.

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

[23] P. Narvaez, K.-Y. Siu, H.-Y. Tzeng, New dynamic SPT algorithm based on a ball-and-string model, IEEE/ACM TON 9 (6) (2001) 706–718.

[24] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, Network Flows: Theory, Algorithms, and Applications, Prentice Hall, 1993.

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

[26] S. Even, A. Itai, A. Shamir, On the complexity of time table and multi-commodity flow problems, SIAM J. Comput. 5 (4) (1976) 691–703.

[27] R.G. Michael, S.J. David, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman, 1979.

[28] M. Leconte, A. Destounis, G. Paschos, Traffic engineering with precomputed pathbooks, in: IEEE INFOCOM, 2018.

[29] J.Y. Yen, Finding the k shortest loopless paths in a network, Manage. Sci. 17 (11) (1971) 712–716.

[30] P. Raghavan, C.D. Tompson, Randomized rounding: a technique for provably good algorithms and algorithmic proofs, Combinatorica 7 (4) (1987) 365–374.

[31] M. Andrews, A.F. Anta, L. Zhang, W. Zhao, Routing for power minimization in the speed scaling model, IEEE/ACM TON 20 (1) (2012) 285–294.

[32] C.E. Leiserson, R.L. Rivest, T.H. Cormen, C. Stein, Introduction to Algorithms, Vol. 6, MIT press Cambridge, MA, 2001.

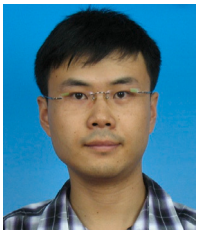
[33] Gurobi, 2018, http://www.gurobi.com.

[34] N. Spring, R. Mahajan, D. Wetherall, Measuring ISP topologies with rocketfuel, ACM SIGCOMM CCR 32 (4) (2002) 133–145.

[35] Y. Yang, M. Xu, D. Wang, S. Li, A hop-by-hop routing mechanism for green internet, IEEE/ACM TPDS 27 (1) (2016) 2–16.

[36] Research data, Data for: Flow-level traffic engineering in conventional routing systems, 2019, Mendeley Data, v2.

Nan Geng received B.Sc. degree from Beijing University of Posts and Telecommunications. He is currently a Ph.D. student of the Department of Computer Science and Technology, Tsinghua University. His research interest includes traffic engineering and network optimization.



Yuan Yang received the B.Sc., M.Sc., and Ph.D. degrees from Tsinghua University. He was a Visiting Ph.D. Student with The Hong Kong Polytechnic University. He is currently an Assistant Researcher with the Department of Computer Science and Technology, Tsinghua University. His major research interests include computer network architecture and routing protocols.

Mingwei Xu received the B.Sc. and Ph.D. degrees from Tsinghua University. He is currently a Full Professor with the Department of Computer Science and Technology, Tsinghua University. His research interest includes computer network architecture, high-speed router architecture, and network security.