Multiple bulk data transfers scheduling among datacenters

Yiwen Wang a, Sen Su a,*, Alex X. Liu b, Zhongbao Zhang a

a State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, 10 Xi Tu Cheng Road, 10086 Beijing, China
b Department of Computer Science and Engineering, Michigan State University, 2132 Engineering Building, East Lansing, MI 48824-1226, United States

A R T I C L E   I N F O

Article history:
Received 16 May 2013
Received in revised form 7 January 2014
Accepted 18 February 2014
Available online 27 February 2014

Keywords:
Traffic engineering
Bulk data transfers scheduling
Inter-datacenter traffic
Optimization models and methods

A B S T R A C T

Bulk data migration between datacenters is often a critical step in deploying new services, improving reliability under failures, or implementing various cost reduction strategies for cloud companies. These bulk amounts of transferring data consume massive bandwidth, and further cause severe network congestion. Leveraging the temporal and spatial characteristics of inter-datacenter bulk data traffic, in this paper, we investigate the Multiple Bulk Data Transfers Scheduling (MBDTS) problem to reduce the network congestion. Temporally, we apply the store-and-forward transfer mode to reduce the peak traffic load on the link. Spatially, we propose to lexicographically minimize the congestion of all links among datacenters. To solve the MBDTS problem, we first model it as an optimization problem, and then propose a novel Elastic Time-Expanded Network technique to represent the time-varying network status as a static one with a reasonable expansion cost. Using this transformation, we reformulate the problem as a Linear Programming (LP) model, and obtain the optimal solution through iteratively solving the LP model. We have conducted extensive simulations on a real network topology. The results prove that our algorithm can significantly reduce the network congestion as well as balance the entire network traffic with practical computational costs.

© 2014 Elsevier B.V. All rights reserved.

1. Introduction

1.1. Background and motivation

Bulk data transfers across geographically distributed datacenters play a critical role in running data backup applications, delivering business services, and implementing various cost reduction strategies. To improve the fault tolerance, terabytes to petabytes of data are regularly replicated among three or more datacenters [1]. To provide a highly available and scalable service, cloud service providers always migrate large amounts of application content to datacenters closer to users, e.g., Amazon CloudFront [2]. Besides, considering the spatial and temporal variability of each datacenter on operational costs, service providers may migrate data to datacenters located in the areas with low electricity costs [3], or carbon emission [4].

The volume of transferring bulk data always ranges from terabytes to petabytes [5]. Due to the peta-scale amount of data, the bulk data transfers impose a heavy load to the links between datacenters, and dominate the inter-datacenter aggregate traffic [6]. Especially when there exist multiple bulk data transfers, an inevitable problem arises with scheduling these bulk data transfers. A schedule without careful design may lead to highly
unbalanced use of the link capability in both spatial and temporal dimensions. This unbalanced bulk traffic always saturates links between datacenters and further induces severe network congestion [7,8] which, not only deteriorates the performance of other coexisting interactive transfers [9], but also impedes the capability of satisfying the flash traffic that may occur in various parts of the network.

To alleviate the network congestion, we can lease more links from ISPs to enlarge the capacity of networks, or choose to transfer bulk data via shipping storage devices by companies such as FedEx [10]. However, both of these methods increase the operational cost of cloud services providers on data transfers. In order to co-ordinate multiple bulk data transfers with time constraints, a careful strategy is called for to schedule transfers, so that the network congestion can be minimized, and bulk traffic can be balanced across the inter-datacenter links as well.

1.2. Limitations of prior art

Most existing works [11–13] investigate bulk data transfer scheduling in the grid networks. Leveraging the delay-tolerance feature of bulk transfers, Chen and Primet [11] study the scheduling for deadline guaranteed multiple bulk data transfers in grid networks. They aim at minimizing the network congestion through providing a flexible scheduling algorithm. Rajah et al. [12] propose a multi-slice scheduling framework for multiple bulk transfers. They formulate the problem as a maximum concurrent multiple flows model and try to improve the throughput of transfers. Rajah et al. continue their work in [13] and propose a novel flexible admission control and periodic scheduling framework to lower the request rejection ratio. All of the works above study the bulk data transfer problem in static networks, and do not consider the practical spatial and temporal dynamics of network resources.

In [14], Laoutaris et al. propose to complete the delay-tolerant bulk transfers via the store-and-forward mode, such that the un-utilized bandwidth resource can be reused, while saving the transfer cost. They further extend their work in [9], based on the predictable link capacity information, NetStitcher stitches together unutilized time-varying bandwidth across multiple datacenters for bulk transfers, so that the existing link capacity is maximally utilized. However, these works mainly focus on dealing with a single bulk data transfer, the more realistic problem of scheduling multiple bulk data transfers still remains unsolved.

1.3. Proposed approach

In this paper, we take the first step towards congestion avoidance in scheduling multiple bulk data transfers with time constraints. Our work is based on the observation of temporal and spatial patterns in the inter-datacenter traffic. Temporally, the inter-datacenter traffic exhibits a strong diurnal pattern [6]. This is because bandwidth resources of datacenters always over-provision to guarantee the peak demands. Suffering from the diurnal pattern of user demands [15], a substantial capacity of the different links is unused during the night. Spatially, the traffic distribution among datacenters is unbalanced across the network. This is because datacenters are geographically distributed in different time zones. When datacenters located in one time zone experience their peak traffic, the datacenters in other time zones may be idle [9]. The temporal and spatial characteristics of inter-datacenter traffic provide us more opportunities of network resources multiplexing.

Combining the delay-tolerant feature of bulk data transfers with the inter-datacenter traffic characteristics, we propose to reduce the link traffic load from both the temporal and spatial dimension. To reduce the peak traffic load over time, we propose to adopt the store-and-forward approach to complete bulk transfers. When the link load is high, we first store the bulk transferring data temporarily in intermediate datacenters, and forward them to the next node at a later time when more link capacities are available.

A toy example is given to illustrate our idea in Fig. 1. There are two transfers $r_1$ and $r_2$ that share the same link with fixed capacity of 3 Gbps. $r_1$ requires to transfer 400 Gb data within time window $[100 s, 300 s]$, and $r_2$ requires to transfer 400 Gb data within time window $[0 s, 400 s]$. Fig. 1(a) shows the end-to-end based scheduling, where $r_1$ and $r_2$ obtain constant 2 Gbps and 1 Gbps bandwidth allocation in their respective time windows. We can observe that all the 3 Gbps link capacities are occupied during the time $[100 s, 300 s]$ in such scheduling. If there comes a transfer $r_3$ with 200 Gb data within time window $[100 s, 300 s]$, it has to be blocked. Fig. 1(b) shows the store-and-forward based scheduling, where $r_1$ obtains 2 Gbps bandwidth in $[100 s, 300 s]$, and $r_2$ obtains 2 Gbps bandwidth in intervals $[0 s, 100 s]$ and $[300 s, 400 s]$. We can observe that the total traffic load remains to be 2 Gbps during the time $[0 s, 400 s]$, since we store the traffic of $r_2$ temporarily when the peak link load is high, and continue its transfer at a later time when link capacity is available. Benefiting from the store-and-forward based scheduling, the network capability of accommodating flash traffic is improved, and $r_3$ can be accepted as well.

To balance the bulk data traffic across links between inter-datacenters, we propose to split the bulk data into blocks and transfer them along multiple multi-hop paths. The existing works on balancing traffic [16,17,11] only focus on minimizing the maximally loaded links congestion. However, when scheduling multiple bulk transfers with different data amounts and time constraints, each transfer passes along different paths, leading the lower bounds of each link congestion to become different as well. Thus, only minimizing the maximally loaded links congestion will lead the traffic on other links to be unbalanced and their congestion to be unabated. To minimize all links congestion as much as possible, but not just the highest loaded links, we propose the lexicographical minimization as our objective. More specifically, the lexicographical minimization first tries to minimize the traffic of the maximally loaded link in the network. Then, it attempts to minimize the traffic of the second maximally loaded link in the network, etc., until all links congestion are minimized.

Fig. 2 shows a toy example to illustrate the benefits of lexicographical minimization in balancing bulk traffic.
Fig. 2(a) depicts the network topology and the numbers next to links denote the capacity. There are two transfers, where $r_1$ and $r_2$ require to transfer 4 and 10 volumes of data from $D_1$ to $D_4$ and $D_2$, respectively. We use the ratio of flow bandwidth allocation and link capacity to denote the link congestion factor. In Fig. 2(b), the allocation is made on a single path for each transfer, which results in an infeasible flow assignment on link $(D_1, D_4)$, whose congestion factor reaches extremely high, even up to 2. In Fig. 2(c), the maximal link congestion is minimized to be 1. However, the traffic on link $(D_1, D_2)$ still can be balanced across other parts of the network. In Fig. 2(d), all link congestion is lexicographically minimized. In such a setting, the traffic load on link $(D_1, D_2)$ is balanced across the path $(D_1, D_3, D_2)$, and the congestion of link $(D_1, D_2)$ is reduced to 0.6 as well. Clearly, the lexicographical minimization achieves a more balanced network status.

In light of these observations, we present a new set of algorithms to minimize the congestion caused by inter-datacenter bulk data transfers. Our solution takes advantage of the delay tolerance of bulk data transfers and the inter-datacenter traffic patterns. Taking into account practical constraints of transfer deadline time and limited link capacities, we try to reduce the network congestion by adopting the store-and-forward approach and by lexicographically balancing traffic across multiple paths.

1.4. Technical challenges and solutions

There are two key technical challenges in developing our approach. First, as the start and end time of transfer requests are different, the temporal dimension has to be taken into account when scheduling multiple bulk transfers, which increases the complexity of the problem significantly. Additionally, since we adopt the store-and-forward transfer mode, storage and bandwidth resources scheduling need to be co-ordinated. Second, when we balance the traffic across links, we should ensure that the traffic cannot be rerouted to other heavily loaded links with higher congestion values. It is difficult to find the tradeoff between balancing the traffic of highly loaded links and reducing the consumption of other links capacities, when scheduling multiple transfers. Moreover, routing and network resources allocation must be jointly considered, since the resources allocation depends heavily on the path along which it is routed. This additional freedom of routing and network resource allocation causes the problem to be more complicated.

For the first challenge, we develop an Elastic Time-Expanded Network technique. It allows us to represent the network status in a given time period as a static network, through creating copies of the network at each time slot and connecting them by adding holdover edges. To prevent the resulting network size expanding to become too large to solve, we elastically duplicate the network, which ensures the network size to be bounded by the variation degree of transfers and networks. Benefiting from this transformation, much convenience is offered to formulate the bulk transfers scheduling problem, while honoring a reasonable expanding cost. To address the second challenge, we formulate the minimize maximal (min–max) congestion model in the elastic condensed time-expanded...
network, and design the corresponding optimal algorithm as well. To obtain the lexicographically minimized link traffic allocation, we iteratively solve the min–max model by the Linear Programming technique. Specific heuristics are employed in identifying links with minimized traffic allocation, such that the algorithm can complete the computation in an efficient manner.

To evaluate the performance of algorithms, extensive simulations are conducted on a real interconnect network topology of datacenters [18]. We compare our algorithms with the simple flow based method, which transfers bulk data over a single end-to-end path. Simulations show that our scheduling algorithm can significantly reduce the network congestion and balance the bulk traffic across links with the simple flow based method, which transfers bulk computation in an efficient manner.

Specifically, our main contributions are: (i) we propose to lexicographically minimize the network bulk traffic load via adopting store-and-forward transfer mode and routing the traffic over multiple paths; (ii) we design an Elastic Time-Expanded Network technique to represent the network status at each time slot as a static one, at a reasonable expanding expense; (iii) we design an optimal algorithm for scheduling multiple bulk transfers and prove its optimality.

The rest of this paper is organized as follows. Section 2 presents the model of multiple bulk data transfers scheduling problem. Section 3 shows the construction of Elastic Time-Expanded Network and the reformulated static model. Section 4 depicts the scheduling algorithms, and Section 5 shows the evaluation results. The related work is discussed in Sections 6, and 7 concludes our paper.

2. Network model and problem formulation

2.1. Network model

Networks: We use a directed graph \( G = (V, E) \) with \( N := |V| \) nodes and \( M := |E| \) edges to represent a network. Each node \( v \in V \) denotes a datacenter, and \( C_v(t) \) denotes the available storage capacity at \( v \) at time slot \( t \). Each edge \( e = (v, w) \in E \) refers to an inter-datacenter link from \( v \) to \( w \). \( C_e(t) \) denotes the available link capacity of \( e \) at time slot \( t \).

We assume that the link capacity for bulk data transfers varies in a time-slotted fashion, since the coexisting interactive traffic varies over time. In each unit of time slot \( t \), we assume that the link capacity remains stable. In a given period \( T \) that consists of multiple time slots, the pattern of these interactive traffic can be predicted according to the historical statistics, since there is strong dependence of the traffic levels on the time-of-day and day-of-week [19–22]. Also, in general, the network administrator determines the links capacity schedule for a certain time period. Thus, the available capacities of inter-datacenter links for bulk data transfers at each time slot \( t \) are assumed to be known in advance, which is a reasonable assumption.

Bulk data transfers: We use \( R = \{r_1, r_2, \ldots, r_k\} \) to denote the set of bulk data transfer requests. Each request is specified as a 5-tuple \( r_i = \{s_i, d_i, dem_i, T_{r_i}^- , T_{r_i}^+\} \), where \( s_i \) and \( d_i \) denote the source and the destination, respectively, \( dem_i \) denotes the demand volume of data to transfer, \( T_{r_i}^- \) and \( T_{r_i}^+ \) denote the request start time and deadline time, respectively. We use \( V_{src} = \{s_1, s_2, \ldots, s_k\} \) and \( V_{dest} = \{d, d_2, \ldots, d_k\} \) to denote the set of source and destination nodes of all the requests, respectively.

Within the transfer deadline time, the transferring data can be temporarily stored in the relay node and then be forwarded to the next hop or destination. Moreover, each request can split its data into blocks and transfer them along multiple multi-hop paths from \( s_i \) to \( d_i \). We view the transferring data on links as flows, where \( f_e(t) \geq 0 \) refers to the flow on link \( e \) at time slot \( t \). We use \( \mu_e(t) := f_e(t)/C_e(t) \) to denote the link congestion factor, which implies the traffic load of link \( e \) at time slot \( t \).

Transfers schedule over time: We use \( S \) to denote the schedule for bulk data transfers over time period with \( T \) time slots. \( S \) decides the amount of data to be stored in each node, and assigns the flow routing paths and bandwidth allocation \( f_e(t) \) for each transfer \( r_i \in R \) over each link \( e \in E \) from time slot 1 to \( T \). Both the routing paths and bandwidth allocation for each transfer varies over time.

2.2. Lexicographically minimize links congestion

When scheduling multiple bulk data transfers, different transfers pass over different links in the network, since the amount of data and the time constraints among transfers are different. Thus, the lower bounds of bulk data traffic that each link has to accommodate are different as well. To minimize the entire network congestion as much as possible, we propose to lexicographically minimize (lex-min) all the links congestion factors at each time slot.

Note that, when we reroute the traffic from one link to the others to minimize its congestion, the load can only be rerouted to the links with lower congestion, but not to the ones that are already congested. This will ensure that the traffic loads are evenly distributed across the network. Formally, the lexicographical minimization can be defined as follows.

Assume that there are \( n \) links in the network, and we use \( \mu(t) \) to denote the network congestion vector at time slot \( t \). Define its corresponding non-increasing ordered vector \( \overline{\mu}(t) = < \mu_1(t), \mu_2(t), \ldots, \mu_n(t)> \), \( \mu_{\hat{m}}(t) \geq \mu_{\hat{m}+1}(t) \) for \( m = 1, 2, \ldots, n-1 \), and \( \mu_{\hat{m}}(t) \) denotes the congestion of link \( e_i \in G \) at time slot \( t \).

Given two ordered link congestion vectors \( \overline{\mu}(t) \) and \( \overline{\nu}(t) \), \( \overline{\mu}(t) \) is lexicographically smaller than \( \overline{\nu}(t) \) if \( \mu_i(t) < \nu_i(t) \) for any \( 1 \leq i \leq m \) and \( \mu_i(t) = \nu_i(t) \) for any \( 1 \leq i < m \) and \( \mu_i(t) < \nu_m(t) \).

A network link congestion vector \( \overline{\mu}(t) \) is lexicographically minimal in \( \Gamma \) iff for every \( \gamma(t) \in \Gamma \), its ordered vector \( \overline{\gamma}(t) \) is lexicographically smaller than \( \overline{\gamma}(t) \).

2.3. Problem formulation

The Multiple Bulk Data Transfer Scheduling (MBDTS) problem is described as follows. Given a network \( G \) with time-varying link and storage capacity \( C(t) \), and the bulk data transfers set \( R \), for each transfer \( r_i = \{s_i, d_i, dem_i, T_{r_i}^-, T_{r_i}^+\} \in R \), the problem tries to derive the transfer schedule \( S \) so as to lexicographically minimize the network congestion vector \( \mu(t) \) in a given period with
$T$ time slots. To ensure the feasibility of the solution, the following constraints must be satisfied for each bulk data transfer:

**Link capacity constraint:** At any time slot $t$, the aggregate traffic along any link cannot exceed the link capacity, and $f^e_r(t)$ denotes the flow of $r_i$ that passes through link $e$ in $t$.

$$\sum_{r_i \in R} f^e_r(t) \leq \mu_r(t) \cdot C_e(t), \forall e \in E \tag{1}$$

**Storage capacity constraint:** At any time slot $t$, the aggregate traffic stored at each node cannot exceed the storage capacity, where $\delta^+(v)$ and $\delta^-(v)$ denote the set of incoming and outgoing edges of $v$, respectively.

$$\sum_{t=0}^T \left( \sum_{e \in \delta^+(v)} f_e^r(t) - \sum_{e \in \delta^-(v)} f_e^r(t) \right) \leq C_v(\theta), \quad \forall v \in V \setminus V_{src}, \theta \in [0,T-1] \tag{2}$$

**Flow conservation constraint I:** No node $v$ can send more flows than that it received at any time before $T$.

$$\sum_{t=0}^T \left( \sum_{e \in \delta^+(v)} f_e^r(t) - \sum_{e \in \delta^-(v)} f_e^r(t) \right) \leq 0, \quad \forall r_i \in R, \forall v \in V \setminus V_{src}, \theta \in [0,T-1] \tag{3}$$

**Flow conservation constraint II:** For any transfer $r_i$, it allows no flows at any node other than nodes in $V_{src}$ or $V_{dest}$ till its deadline time $T^d_i$.

$$\sum_{t=T^d_i}^T \left( \sum_{e \in \delta^+(v)} f_e^r(t) - \sum_{e \in \delta^-(v)} f_e^r(t) \right) = 0, \quad \forall r_i \in R, \forall v \in V \setminus (V_{src} \cup V_{dest}) \tag{4}$$

**Volume constraint:** For any transfer $r_i$, the total amount of flows sent from the source and received at the destination equals its specified volume till its deadline $T^d_i$.

$$\sum_{t=0}^{T^d_i} \left( \sum_{e \in \delta^+(v)} f_e^r(t) - \sum_{e \in \delta^-(v)} f_e^r(t) \right) = \text{dem}_{r_i}, \quad v = s_i, \quad \forall r_i \in R, \quad s_i \in V_{src}, \quad d_i \in V_{dest} \tag{5}$$

Next, we give the model of the MBDTS problem. We aim at computing a feasible bulk transfers scheduling solution that lexicographically minimizes the network congestion vector $\mu(t)$ in a given period $T$, satisfying the constraints above, which is:

$$\text{lex} - \min \mu(t) := \mu_{t_1}(t), \mu_{t_2}(t), \ldots, \mu_{t_n}(t) > t \in [0,T] \tag{6}$$

$$\text{s.t. } (1), (2), (3), (4), (5)$$

In our framework, all the network resources are managed by a central controller. This is reasonable since all datacenters are operated by the same cloud provider, and it is also technically efficient to devise a centralized controller to obtain and access information about the entire network. The central controller periodically schedules the collection of bulk data transfers in each given period of $T$ time slots. The scheduling solution is feasible iff it satisfies $\forall \mu_r(t) \in \mu(t)$, $0 < \mu_r(t) < 1$. If there exists no feasible solution, the central controller has to reject some transfers. The strategy of admission control for transfer requests is out of the scope of the present work. In the rest of the paper, we only consider the scenarios where a feasible schedule exists.

### 3. Simplified modeling on Elastic Time-Expanded Networks

The model of MBDTS problem in Section 2, involves, $\Theta(T \cdot |R| \cdot |E|)$ variables, which incurs a high computational cost to solve. In this section, we propose a novel Elastic Time-Expanded Network technique to simplify the model of MBDTS problem.

#### 3.1. Motivation for Elastic Time-Expanded Network

The Time-Expanded Network (TEN), introduced by Ford and Fulkerson in [23], is a powerful technique to solve the flow over time problem. Through copying the network status at each time slot, it transforms the flow over time problem as a static flow problem. Given a network $G$, a $T$-time-expanded network $G^T$ is composed of $T$ copies of the node set in $G$ for each time slot. Besides, for every edge $e = (u, v)$ in $G$ with link capacity $C_e(t)$, there is a copy edge $e_t = (u_t, v_t)$ between all pairs of time layers with link capacity $C_e(t)$ in $G^T$. To represent the storage capacity $C_v(t)$ of each node $v$, additional holdover edges $(w_t, w_{t+1})$ with capacity $C_v(t)$, are added to connect different time copies of the same node.

Fig. 3 presents a toy example, where (a) shows the original network $G$ and the resulting $T$-time-expanded network $G^T$ with $T = 9$. The numbers next to the edges represent the capacity of links and storages. There exists a transfer request $r_1$ from $s$ to $d$, whose start time is 3 and the deadline time constraint is 7. In the time-expanded network $G^T$, we set $s_3$ and $d_7$ as the source and destination node of $r_1$, respectively, so that the transfer schedule from $s_3$ to $d_7$ in $G^T$ can be mapped to a transfer schedule over time for $r_1$ conveniently and correctly.

Although the time-expanded technique can represent various network status as a static one, it suffers from a huge size of the network which is, in general, exponential in the input size $T$. As consequences, the huge size of $G^T$ will lead to a substantial computational cost. To reduce the size of time-expanded network, a straightforward way is to re-scale the length of unit time slots to larger ones, which implies a rougher time discretization. This is proposed as condensed time-expanded network technique in [24]. However, this uniform condensed time-expanded network still cannot be applied in time-varying networks for the following reasons.

**First,** the precision loss caused by condensing network copies over each time slot may cause the transfer schedule to be infeasible in real scenarios. Fig. 3(b) shows the corresponding condensed time-expanded network of $G^T$, denoted as $G^A$ with $A = 3$. We can observe that (i) the
maximum flow along the path \((s_y', w_y', d_y')\) is 9, whose real maximum flow can only be 6 during time \([4,6]\), since the constraint of bottleneck links \((s_y, w_y), (s_y', w_y')\), \((w_0, d_0)\); (ii) a flow of rates 6 for \(r_1\) on the path \((s_y', w_y', d_y')\) in \(G^3\) is feasible. However, constrained by the deadline of \(r_1\), only the network resources before time slot 7 can be reserved for \(r_1\).

Secondly, the flow allocation in \(G^3\) is difficult to interpret as a practical solution, since it does not consider the network variance information. In Fig. 3(b), the rough discretization leads it hard to calculate the flow on edge \((w_i', d_i')\), since we cannot decide the flow rate assigned to each time slot from 1 to 3.

3.2. Construction of Elastic Time-Expanded Network

Motivated by this limitation, we propose a novel Elastic Time-Expanded Network technique to represent the time-varying network status at each time slot. Our basic idea is to condense each time slot elastically, according to the variability of active time windows for all links, storage nodes and transfer requests. We condense multiple time slots into only one when capacities of links and nodes both remain to be constant. And we define these time slots as an active time window \([T_i, T_j]\), where \(T_i, T_j \in [0, T]\) refer to the start and end time slot of the active time window, respectively. To prevent the time constraint of transfer \(r\) from being omitted in active time windows, when there arrives a transfer request \(r\) in the active time window \([T_i, T_j]\), we have to split the time window into three sub active time windows. The transfer start and deadline time are denoted as \(T_{s'}\) and \(T_{d'}\), and the sub time windows are denoted as \([T_{s'}-T_{s}'), [T_{s'}-T_{d}'), [T_{d'}-T_{d}']\). For similar reasons, when there arrives a transfer request across multiple active time windows \([T_i, T_{i+1}], [T_{i+1}, T_{i+2}], \ldots, [T_{k-1}, T_k]\), we split both the first and the last time windows into two parts as \([T_i, T_{i+1}], [T_{i+1}, T_{i+2}]\) and \([T_{k-1}, T_k]\). To obtain an Elastic Time-Expanded Network of \(G\), we use the following steps:

**Step 1:** For all time slots \([0,1,2,\ldots,T]\), we split them into multiple active time windows \(I_0 = [T_0, T_1], \ldots, I_l = [T_l, T_{l+1}], \) where \(T_i, T_i+1 \in [0, T]\) and both the links and nodes capacities remain constant in \(I_l\). We use \(D' = \{T_0, T_1, \ldots, T_l\}\) to refer to the corresponding time slot sequence.

**Step 2:** For each transfer request \(r\), insert their start time \(T_{s'}\) and end time \(T_{d'}\) into the sequence \(D'\) in an ascending order, and the sorted list is denoted as \(D = \{T_0, T_1, \ldots, T_q\}\). Then we obtain \(q\) active time windows \(I_k = [T_k, T_{k+1}], \) where \(k = 0, q\), and the element \(T_k \in D\) is termed as the border time point. We use \(I = I_1, I_2, \ldots, I_q\) to denote the set of active time windows.

**Step 3:** Create \(q+1\) copies of \(V\), labeled as \(V_0, V_1, \ldots, V_q\), with the kth copy of node \(v\) denoted as \(v_k\).

**Step 4:** If the link \(e_m = (v, w)\) in the original network \(G\) can be used at any time slot \(T_k \in \{T_0, T_1, \ldots, T_q\}\), a transit edge with the capacity of \(C_e(T_k) \cdot (T_{k+1} - T_k)\) is added from \(v_k\) to \(w_k\), which is denoted as \(e_m^k\).

**Step 5:** For each node \(w\), to represent its storage capacity at each active time window \(I_k\), we add a holdover edge with the capacity of \(C_w(T_k)\) from \(w_k\) to \(w_{k+1}\) to the elastically expanded network.

After these five steps, we obtain the Elastic Time-Expanded Network, denoted as \(G^E = (V^E, E^E)\). Given a network \(G\) with \(|V|\) vertices and \(|E|\) edges, the construction of the Elastic Time-Expanded Network depends only on the sorted elements in \(D = \{T_0, T_1, \ldots, T_q\}\), but not on the values of \(T\). The resulting size of network \(G^E\) is limited to \(O(|D| \cdot |V|)\) nodes and \(O(|D| \cdot |E|)\) edges, where \(|D|\) denotes the number of elements in \(D\). Indeed, the sorted list \(D\) represents the variation of networks, including both the time-varying
links and storage nodes, as well as the diversity of the transfer deadline constraints. Thus, $|D|$ decides the size of the resulting $G^0$, and we call it diversity degree.

Fig. 3(c) shows the corresponding elastic time expanded network of $G'$, which we denote it as $G^2$. Compared with the uniform condensed $G^2$ in (b), we have the following observations. (i) The network copy of time slot 6 is not omitted, so that the infeasible schedule caused by the precision loss of link capacity variations will not appear. (ii) The network copy of time slot 7 is not condensed either, so that the difficulty in rate allocation for each time slot is avoided in $G^2$ via allocating the average rate to each time slot.

3.3. Correctness of Elastic Time-Expanded Network

First, we propose the algorithm that interprets $f'$ to a flow over time $f$ in $G$ as shown in Algorithm 1.

**Algorithm 1. Flow Interpretation Algorithm**

1: **Input:** A static flow $f^0$ in $G^0$;
2: /* Merging $f^0$ to flow over time $f$ in $G^*$
3: for $e \in E^0$ do
4:  if $f^0$ passes the transit edge $e_{m,k}$ then (v.k, w.k) with
5:  transfer volumes of $|f^0_{em}|$ data from v to w with a
6:  constant rate $\mu_{em}\frac{|f^0_{em}|}{|T_{k+1} - T_k|}$ during $[T_k, T_{k+1}]$;
7:  end if
8:  store volumes of $|f^0_{em}|$ data at w during $[T_k, T_{k+1}]$.
9: end if
10: end for
11: **Output:** The flow over time assignments $f$ in $G$

**Lemma 1.** Let $D = \{T_0, T_1, \ldots, T_q\}$ with $T_k < T_{k+1}$ for all $k = 0$ to $q - 1$. Then any static flow $f^0$ in $G^0$ corresponds to a feasible flow over time $f$ in $G$, such that the amount of flow reaching $w$ in $G$ by time $T_{k+1}$, $k = 0, \ldots, q - 1$, equals to the amount of flow reaching node $w_{k+1}$.

**Proof.** After interpreting from $f^0$ to $f$, the amount of flow reaching $w$ in $G$ through link $e_m$ is $\mu_{em} \cdot C_{em}(T_k) \cdot |T_{k+1} - T_k|$, which equals the amount of flow passing through $e_{m,k}$ to $w_{k+1}$ in $G^0$. Clearly, this also holds for other incoming links to $w$, and thus the total amount of flow reaching $w$ in $f$ by $T_{k+1}$ equals to that reaching the node $w_{k+1}$ in $f'$.

The flow $f'$ in $G^0$ satisfies the capacity constraint, which is $|f'_{em}| \leq C_{em}(T_k) \cdot |T_{k+1} - T_k|$. Then we have, $|f^0_{em}|/|T_{k+1} - T_k| \leq C_{em}(T_k)$, which shows the capacity constraint of $f$ is guaranteed. By design, $f$ ensures the flow conservation as well, since $f'$ is feasible in $G^0$. Moreover, when we construct the $G^0$, no backward edge from time slot $T_{k+1}$ to $T_k$ is added. This ensures that no data stored at the nodes can be transferred to the sink with the previous available link resource. Therefore, $f$ is a feasible flow over time in $G$. □

**Lemma 2.** Any static flow $f^0$ in $G^0$ corresponds to a feasible flow over time $f$ in $G$, such that the congestion value of each link $e_m$ in $G$ during its atomic time window $I_k = [T_k, T_{k+1})$, $k = 0, \ldots, q - 1$, equals to the congestion of transit edge $e_{m,k}$ in $G^0$.

**Proof.** After interpreting from $f^0$ to $f$, the congestion of link $e_m$ during its atomic active window $I_k = [T_k, T_{k+1})$ is constant, which is $|f^0_{em}|/|T_{k+1} - T_k|$. Clearly, it equals to the congestion value of the transit edge $e_{m,k}$ in $G^0$, which is $|f^0_{em}|/|T_{k+1} - T_k|$. □

Let $\mu(e_{m,k})$ refer to the congestion value of transit edge $e_{m,k}$, and all the congestion values of transit edges in $G^0$ compose the network congestion vector, denoted as $\mu_G = \{\mu(e_{1,1}), \mu(e_{1,2}), \ldots, \mu(e_{m,k})\}$.

**Theorem 1.** Let $f'$ be the optimal static flow assignment in $G^0$ that satisfies all the transfer requests, while lexicographically minimizing the network congestion vector $\mu_G$. Then the corresponding flow over time assignment $f$ in $G$ is the optimal solution that satisfies all the transfer requests, while the resulting network congestion vector $\mu_G$ is lexicographically minimized.

**Proof.** We first prove that the solution $f'$ in $G$ satisfies all the transfer requests. According to Lemma 1, the total amount of flow reaching each sink node $d_i \in V_{dest}$ by time $T_k \in D$ equals to that of flow reaching node $d_i$ in $G^0$. Moreover, because the deadline time of each transfer request is picked up as the border time point $T_k \in D$. We obtain that the flow over time $f'$ can transfer specified volumes of data under the deadline constraint.

Assume that $\mu_G$ is not lexicographically minimized, i.e., there is a congestion value $\mu_{i,j}$ in $\mu_G$ that can be further minimized by increasing the one that has a lower congestion value, denoted as $\mu_{e_{i,j}}$. According to Lemma 2, it implies the corresponding congestion value $\mu(e_{i,j})$ in $G^0$ can be decreased through increasing $\mu(e_{i,j})$, where both $\mu(e_{i,j})$ and $\mu(e_{i,j})$ are the elements of the vector $\mu_G$. This is in contradiction to the condition that $\mu_G$ is lexicographically minimized. Therefore, we prove that the flow over time assignment $f$ in $G$, which corresponds to the optimal flow assignment in $G^0$, is also optimal. □

3.4. Reformulation of MBDTS problem

After obtaining the Elastic Time-Expanded Network $G^0$, we can reformulate the multiple bulk data transfers scheduling problem as a static network flows problem. For each
bulk transfer in $G^0$, its source is the virtual sender node $s(T^+_i)$, and the destination is the virtual receiver node $d(T^-_i)$. $V^0_{src}$ and $V^0_{dest}$ refer to the set of all the requests source nodes and destination nodes in $G^0$, respectively. $E^0_l$ and $E^0_w$ refer to the set of all transit edges and holdover edges, respectively. Let $\mu=\langle \mu(e_1), \mu(e_2), \ldots, \mu(e_k) \rangle$ denote the congestion vector for each transit edge $e_k$ in $G^0$. The model is as follows:

$$\text{lex} - \min \mu_k = \langle \mu(e_1), \mu(e_2), \ldots, \mu(e_k) \rangle$$

s.t. $\sum_{r \in \mathbf{R}} f^0_r \leq \mu(e_k) \cdot C(e_k), \quad \forall e_k \in E^0$ (7)

$$\sum_{r \in \mathbf{R}} f^0_r \leq C(e_w), \quad \forall e_w \in E^0_w$$ (8)

$$\sum_{e \in \mathbf{E}} f^0_r - \sum_{e \in \mathbf{E}} f^0_r = 0, \quad \forall v \notin (V^0_{src} \cup V^0_{dest})$$ (9)

$$\sum_{e \in \mathbf{E}} f^0_r - \sum_{e \in \mathbf{E}} f^0_r \leq \begin{cases} \text{dem}_e, & v = s_i(T^+_i) \\ \text{dem}_e, & v = d_i(T^-_i) \end{cases}$$ (10)

$$\forall r \in \mathbf{R}, \ s_i(T^+_i) \in V^0_{src}, \ d_i(T^-_i) \in V^0_{dest}$$

$$\forall e_k \in E^0, f^0_r \geq 0, \ 0 \leq \mu(e_k) \leq 1$$ (11)

The objective is to compute the lexicographically minimized transit edges congestion vector in $G^0$. Any flow should ensure that the capacity of transit edges constraint (7) is satisfied. The storage constraint (8) is uniformly represented as holdover edges capacity constraint in $G^0$. Flows in $G^0$ still should ensure the flow conservation constraint (9) is satisfied. Constraint (10) ensures that each transfer receives the specified volume of data within its deadline time, where the final sink node $d_i(T^-_i)$ implies the transfer deadline constraint. Constraint (11) ensures that the flow value and the congestion value are feasible. After this reformulation, no temporal variables exist, which allows us to find more efficient solutions for this problem.

4. Lexicographically minimizing network congestion

In this section, we propose an efficient algorithm for solving the multiple bulk data transfers scheduling problem. The basic idea is to iteratively minimize the maximally loaded link congestion, until all links are optimized.

4.1. Optimal scheduling algorithm

To obtain the lex-min congestion vector $\mu$, in $G^0$, the model in Section 3.4 is modified to be a Min–Max Linear Programming (Min–Max LP) form as follows:

$$\text{minimize : } \mu$$

s.t. $\sum_{r \in \mathbf{R}} f^0_r \leq \mu(e) \cdot C(e), \quad \forall e \in E^0_{\text{min}}$ (12)

$$\sum_{r \in \mathbf{R}} f^0_r \leq \mu_{\text{min}}(e) \cdot C(e), \quad \forall e \in E^0$$ (13)

$$\mu(e) \leq \mu, \quad \forall e \in E^0_{\text{min}}$$ (14)

$\mu(e) = \mu_{\text{min}}(e)$, as illustrated in (13).

Our basic idea is to iteratively minimize the maximal congestion bound of transit edges in $G^0$ through solving the Min–Max LP. In each iteration, the edges whose congestion cannot be further decreased using any solutions are added to the set $E^0_{\text{unmin}}$, and their minimized congestion values are also fixed in the following iterations. Then we continue solving the Min–Max LP to minimize the secondary highest congestion bound, etc., until all congestion values are minimized.

The difficulty of this process lies in efficiently identifying the maximally minimized edges. We fix it through solving the Min–Max LP again to test whether the possible edges can be further minimized, under the relaxation that other edges congestion values are set to be the upper bound $\mu$. Furthermore, to speed up the identification process, two heuristics are employed: (i) only links with the same congestion value as the bound are to be tested; (ii) when the bound is minimized to be zero, we can identify that all links in $E^0_l$ are minimized with congestion values of zero.

The Optimal Network Congestion Lexicographical Minimization (Opt-Lexmin) algorithm is shown in Algorithm 2. Its inputs include the Elastic Time-Expanded Network $G^0$ and the transfer request set $R$. It starts with initializing $E^0_{\text{min}}$ and $E^0_{\text{unmin}}$ (Line 2). In iteration process, it first solves the Min–Max LP with the current sets of $E^0_{\text{min}}$ and $E^0_{\text{unmin}}$ (Lines 4 and 5), and obtains the upper bound $\mu_q$ of the common minimized link congestion. If the upper bound is zero, then we can identify all other links are minimized with the congestion value of zero (Lines 7–10). Otherwise, the link $e_i$ that reaches the upper bound $\mu_q$ in set $E^0_{\text{unmin}}$ is the candidate to test. Then, the algorithm performs a test to identify whether the candidate link $e_i$ can be further minimized in different flow assignments or not. Let $E^0_{\text{unmin}}$ only contain $e_i$, and all other edges in $E^0_{\text{unmin}}$ are tagged as temporarily minimized with upper bound $\mu_q$. Then, the Min–Max LP is solved again (Lines 11–13). If the newly calculated $\mu_{\text{temp}}(e_i)$ equals $\mu_q$, it means that $e_i$ is really minimized. We fix $\mu_q$ to be the minimized congestion value of $e_i$, update $E^0_{\text{min}}, E^0_{\text{unmin}}$ and continue the next iteration (Lines 14–17). The algorithm outputs include the flow allocation rate $f^0_r$ of $r_i$ on each edge and the congestion $\mu_q$ of each transit edge $e_i$. Finally, we map the static flow in $G^0$ to the feasible schedule by Algorithm 1.
Algorithm 2. Opt-Lexmin Network Congestion Algorithm

1: **Input:** $R$ and $G^D$;
2: $E_{unmin}^D = E^D$, $E_{min}^D = null$;
3: while $E_{unmin}^D \neq null$ do
4: solve Min–Max LP using $E_{unmin}^D$, $E_{min}^D$
5: output $\mu_q$, $\forall e \in E^D$, $\mu(e)$, $\forall r_i \in R$, $f_{e_i}^{r_i}$
6: [*Identify the maximally minimized edges*]
7: if $\mu_q = 0$ then
8: $E_{min}^D \leftarrow E_{min}^D \cup E_{unmin}^D$;
9: $\forall e_i \in E_{unmin}^D, \mu_i(e_i) \leftarrow 0$;
10: $E_{unmin}^D \leftarrow null$;
11: end if
12: for $e_j \in E_{unmin}^D, \mu(e_j) = \mu_q$ do
13: solve Min–Max LP using $E_{unmin}^D \setminus \{e_j\}$, $\forall e \neq e_j : \mu_{\min}(e) < \mu_q$
14: output $\mu_{temp}(e_j)$, $\forall r_i \in R$, $f_{e_i}^{r_i}$
15: if $\mu_{temp}(e_j) = \mu_q$ then
16: $E_{min}^D \leftarrow E_{min}^D \cup \{e_j\}$
17: $E_{unmin}^D \leftarrow E_{unmin}^D \setminus \{e_j\}$
18: $\mu_{\min}(e_j) \leftarrow \mu_{temp}(e_j)$
19: end if
20: end for
21: end while
22: output $\forall e_i \in E^D$, $\mu_i(e_i)$ and $\forall r_i \in R$, $f_{e_i}^{r_i}$

4.2. Correctness of the scheduling algorithm

Let $|E^D|$ denote the number of transit edges in $G^D$. The algorithm iterates at most $|E^D|$ times, as there must exist one edge with the maximally minimized congestion value $\mu_q$ obtained in each iteration, i.e., we can identify at least one edge in each iteration. Thus, the algorithm terminates in at most $|E^D|$ iterations. Next, we prove its correctness and optimality.

**Lemma 3.** Any edge $e_i$, that is identified as minimized at iteration $q$ cannot obtain a lower congestion value in any solution under the min–max fairness.

**Proof.** Let $\mu_q$ refer to the minimized congestion value in iteration $q$. We first show that the congestion value $\mu_q$ of minimized edges in iteration $q'$ such that $q' < q$ is no lower than $\mu_q$. This is because $\mu_q$ is assigned to be the minimum common congestion upper bound of each edge and it is a non-increasing vector. Then we conclude that if $e_j \in \bigcup_{q=0}^{q'} E_{\max}(q)$, $\mu_i(e_j) \geq \mu_q$.

To prove it by contradiction, suppose $\mu_q$ can be further decreased under min–max fairness. According to the operations in Lines 4 and 5, the congestion of all edges in $E_{unmin}^D$ is at most $\mu_q$. Then we identify the edge with minimized congestion value through testing its minimized value in other flow assignments. And the congestion of edges in $E_{unmin}^D$ is at most the same value with the minimized one, since $\mu_{\min}(e_i) = \mu_{temp}$ is actually equal to $\mu_q$. Thus, any possible decrease for $\mu_q$ has to increase the congestion that is minimized in previous iterations $q'$, whose value is higher than $\mu_q$. This is in contradiction to the constraint of min–max fairness, and we complete the proof.

**Theorem 2.** The solution of Opt-Lexmin algorithm is the optimal flow assignment so that the network links congestion vector is lexicographically minimized.

**Proof.** According to Lemma 3, each minimized congestion value $\mu_q$ obtained from the Opt-Lexmin algorithm cannot be decreased under min–max fairness, and the newly calculated congestion rate $\mu_q+1$ is no higher than $\mu_q$. Thus, the final congestion vector is lexicographically minimized in the non-increasing order. According to Theorem 1, it is also the optimal solution to the multiple bulk transfers scheduling problem.

The complexity of the Opt-Lexmin algorithm mainly depends on the iteration process. For a network with $|E|$ edges, $R$ transfer requests and diversity degree $D$, to identify all minimized congestion edges, the Min–Max LP is required to be solved by $O(|E| \cdot |E| \cdot D)$ times. The Min–Max LP requires to solve $O(|R| \cdot |E| \cdot D)$ variables, whose complexity is denoted as $poly(|R| \cdot |E| \cdot D)$ for simplicity. Thus the overall complexity is $O(|E|^2 \cdot D^2 \cdot poly(|R| \cdot |E| \cdot D))$, which ensures a reasonable computational cost for practical size inputs.

5. Performance evaluation

In this section, we evaluate the performance of our algorithms on a real network topology. Our evaluation results have validated that: (i) by leveraging the store-and-forward transfer mode and lexicographical optimization, the network links congestion can be significantly reduced, and the bulk traffic load is also more balanced in both temporal and spatial dimensions; (ii) by elastically condensing the time-expanded network, the solution can be obtained efficiently within practical time.

5.1. Implementation and settings

In our simulations, the scheduling algorithms run in a periodic fashion. In each period, the network controller schedules the collection of bulk data transfers. In our settings, each period $T$ consists of 100 time slots, and we set 1 h as the length of one uniform time slot. We randomly generate different numbers of transfers. Each randomly generated transfer request attempts to send a data file with the volume ranging from 1 TB to 100 TB, and its active time interval is also uniformly random within time slots from 0 to 100.

The real-world network topology of Softlayer Inc.[18] is modeled in our simulations. As shown in Fig. 4, it consists of 11 nodes and 17 bi-directional links. The available capacities of each link are the same in both directions,
and the capacities are uniformly random from 1 GB to 10 GB within 100 time slots. In each time slot, the capacity remains constant. Fig. 5 shows how links capacities change over time. For each datacenter, its capacities for bulk data transfers stay available for 4 time slots, and then become un-available for 3 time slots. This pattern repeats till the 100th time slot. This represents the temporal characteristic of bandwidth utilization. Each of their available time windows starts at a different time, since it depends on the end time of each of their peak traffic loads. This represents that the bandwidth utilization differs with the locations of datacenters, which is the spatial characteristic of bandwidth utilization. For simplicity, we assume that the storage capacity of each datacenter remains constant for all the 100 time slots.

For each transfer request, the source and destination nodes are randomly chosen from the network node pairs. The volume $dem$ of bulk data that it attempts to send ranges from $[10 \; \text{TB}, 100 \; \text{TB}]$, follows an uniform distribution. We set all transfer requests to start at time slot 0, and their transfer deadlines $T_{dr}$ of each transfer requests range from $[1, 100]$, which also follows an uniform distribution as well. Then, we decide the number of transfer requests by the following steps. First, the mean capacity of each link during 100 time slots is about 5 GB, and the maximum volume of bulk data that can be transferred across the link in 100 time slots is $5 \; \text{GB} \times 100 \times 3600 = 1.5 \; \text{PB}$. Second, the bottleneck of the network lies in 3 links, and we can obtain that the maximum network transfer capability is about $3 \times 1.5 \; \text{PB} = 4.5 \; \text{PB}$. Third, in the worst case where each request tries to send 100 TB data, each with a different deadline time, the maximum number of requests the network can support is about $\frac{10 \; \text{TB}}{45} = 222$. Considering the source and destination node pairs distribute in all the networks, and the volume of transferring data can be less than 100 TB, we choose to generate five sets of requests with value from 10 to 50. These sets of transfers can represent different levels of network load without exceeding the network maximum transferring capability.

We implement four algorithms: Flow-MinMax, SnF-MinMax, SnF-LexMin and SnF-LexMin-eTEG, whose details are shown in Table 1. We use the GNU Linear Programming Kit (GLPK) [26] to solve the LP model in our algorithms. The algorithm execution time is recorded on a machine with 2 GB RAM and Intel Core2 dual-core 2.1 GHz CPU.

5.2. Temporal variability of peak traffic load

To demonstrate the impact of different algorithms on reducing peak traffic in temporal dimension, we evaluate the congestion of networks from the metrics of maximal link congestion value, and evaluate the traffic distribution across links from the metrics of variance value of link congestion. We perform all algorithms under different numbers of requests from 10 to 50, and record the two metrics at every 5th time slot. Because the SnF-LexMin and SnF-LexMin-eTEG algorithms both employ the lexicographical optimization technique, their transfers scheduling results are the same (their difference lies in the computational cost). We only present the result of SnF-LexMin in this sub section. Figs. 6 and 7 show these two results for Flow-LexMin, SnF-MinMax and SnF-LexMin.

### Table 1

<table>
<thead>
<tr>
<th>Notation</th>
<th>Algorithm description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow-MinMax</td>
<td>The algorithm that only allows flow based end-to-end transfer mode, where the storage capability is prohibited, only aiming at minimizing the maximal link congestion in the network</td>
</tr>
<tr>
<td>SnF-MinMax</td>
<td>The algorithm that allows store-and-forward assisted transfer mode, only aiming at minimizing the maximal link congestion in the network</td>
</tr>
<tr>
<td>SnF-LexMin</td>
<td>The optimal algorithm shown in Section 4.1 that lexicographically minimizes the network congestion vector, allowing store-and-forward assisted transfer mode, formulated on the uniformly time-expanded network</td>
</tr>
<tr>
<td>SnF-LexMin-eTEG</td>
<td>The optimal algorithm that lexicographically minimizes the network congestion vector, allowing store-and-forward assisted transfer mode, formulated on the Elastic Time-Expanded Network</td>
</tr>
</tbody>
</table>

![Fig. 4. Softlayer network, with 11 datacenters and 17 bi-directional links.](image)

![Fig. 5. Available link capacity duration time of datacenters.](image)
algorithm, respectively. Due to the space limitation, only three sets of results are shown, and we have the following observations.

- **Flow-MinMax vs SnF-MinMax**: When comparing the Flow-MinMax and SnF-MinMax algorithms, we can conclude that the peak bandwidth consumption along temporal dimension can be reduced significantly by carefully utilizing the storage capacity. The maximal link congestion value of SnF-MinMax is less than that of Flow-MinMax by about 13% in all sets of transfer requests. Further, the maximal congestion values of SnF-MinMax at each time slot keep more steadily than that of Flow-MinMax, i.e., the network traffic load fluctuates little in the temporal dimension. Additionally, the variance values of SnF-MinMax are also less than that of Flow-MinMax, which implies that the bulk traffic distributes more evenly than Flow-MinMax. This is because SnF-MinMax leverages the delay-tolerance feature of bulk transfers via store-and-forward, which provides more opportunities for multiplexing the network resources in time dimension. With the increase of transfer requests, the variance value difference between these two algorithms decreases. This is because the increase of transfers leaves less room for SnF-MinMax to optimize the transfers scheduling.

- **SnF-MinMax vs SnF-LexMin**: We observe that SnF-LexMin achieves a lower variance value of link congestion than SnF-MinMax, which implies its traffic is better balanced. The reason for this lies in that SnF-LexMin pays more attention to minimize the other links congestion. Besides, the maximal link congestion value of SnF-LexMin is the same as SnF-MinMax. This is expected since both of these two algorithms can minimize the congestion of the maximally loaded link.

In summary, this set of simulations prove that the network congestion can be significantly reduced via store-and-forward, and the bulk traffic can also be balanced better in the temporal dimension.

### 5.3. Spatial distribution of bulk data traffic

To demonstrate the impact of different algorithms on balancing bulk data traffic in spatial dimension, we compare the cumulative distribution function (CDF) of links congestion in each time slot. Similarly, we only present the result of SnF-LexMin, but not both of SnF-LexMin and SnF-LexMin-eTEG, since their scheduling results remain the same. Fig. 8 shows the CDF of links congestion with 10, 30 and 50 transfers, respectively, for Flow-MinMax, SnF-MinMax and SnF-LexMin algorithms. Due to the space limitation, only the results at time slot 10, 50 and 90 are shown, and we have the following observations.

- **Flow-MinMax vs SnF-MinMax**: When comparing the CDF of these two algorithms, we observe that SnF-MinMax balances the bulk data traffic better than Flow-MinMax. In most settings of Flow-MinMax, about 30% of network links are left unused, whose congestion value is lower than 0.2. In contrast, about 25% of network links congestion is higher than 0.6 when there are 30 and 50 transfer requests. Even worse, when there are 50 transfers, about 35% of links suffers heavy congestion value higher than 0.8. This shows that...
Flow-MinMax leads most bulk traffic to distribute on a small part of network links. However, in the settings of SnF-MinMax, we can observe that the bulk data traffic distributes evenly across the network. Only about 20% of links congestion remain to be lower than 0.2, and about 55% of link congestion are around 0.3–0.6. This proves that the SnF-MinMax can utilize the network resources better to balance the bulk data traffic.

- **SnF-MinMax vs SnF-LexMin**: We observe that SnF-LexMin outperforms SnF-MinMax in balancing the bulk data traffic. In most settings of SnF-LexMin, there exists no link with congestion value lower than 0.2, which implies all link capacities are used to balance traffic load. Consequently, only about 18% of links congestion values are higher than 0.6, which is less than that of SnF-MinMax about 30%. This proves that the SnF-LexMin can balance the bulk data traffic better via lexicographically minimization, and the congestion of the entire network is reduced as much as possible.

In summary, this set of simulations further show that adopting the store-and-forward transfer mode can balance the network traffic better than only using the flow based transfer mode. Moreover, the lexicographical minimization can produce a more balanced solution in the spatial dimension, at the expense of more link capacities are utilized.

5.4. Algorithm computational cost

To evaluate the benefits of the Elastic Time-Expanded Network, we analyze the computational cost of SnF-LexMin and SnF-LexMin-eTEG algorithms, which are based on the uniform and Elastic Time-Expanded Network, respectively. We measure their each execution time under different request numbers from 10 to 50. Each simulation repeats 5 times, the mean execution time and 95% confidence interval are recorded, as shown in Fig. 9. Then we have the following observation.

Suffering from the huge expanding size, SnF-LexMin consumes more time than SnF-LexMin-eTEG, about five times under all sets of request numbers. Especially for 50 requests, its running time even passes 50 min, which is intolerable. Although the execution time of SnF-LexMin-eTEG increases with the number of transfer requests, it costs no more than 10 min for all settings, which is reasonable in the scenario of non-real-time bulk transfers scheduling.

To evaluate the extra computational overhead of SnF-LexMin-eTEG over SnF-MinMax, we present their mean execution time and 95% confidence interval in Fig. 10. We can observe that the running time of SnF-LexMin-eTEG is much longer than that of SnF-MinMax, since it tries to minimize all links congestion but not only the heaviest one. Although the computational overhead is high, its
largest execution time still stays around 10 min. This is practical in the scenario of scheduling multiple bulk data transfers for the following reasons: (1) The transfer of PB-scale bulk data along network links with Gb-scale capacities always lasts for multiple days or even longer. Compared with such a long transfer time, its running time is reasonable, (2) The bulk data transfer is delay tolerant, which allows us to collect a set of transfer requests and schedule them in an off-line fashion. Compared with the advantage of SnF-LexMin in balancing the whole network bulk traffic, its computational overhead is acceptable indeed.

In summary, the simulations demonstrate that, compared with the uniform time-expanded network, elastically expanding network technique can significantly reduce the computational cost of obtaining the lex-min optimal link congestion vector over time. Additionally, a reasonable computational cost is ensured within the size of practical inputs.

5.5. Store operations cost

To evaluate the costs of adopting the store-and-forward transfer mode, we analyze the number of store operations of SnF-MinMax and SnF-LexMin algorithms. We record the total number of store operations under different requests numbers from 10 to 50. Each simulation repeats 5 times, the mean execution time and 95% confidence interval are recorded, as shown in Fig. 11. We have the following observations.

In all sets of simulations, the SnF-MinMax only consumes about 40% less store operations than that of SnF-LexMin. This is expected since SnF-LexMin tries to minimize all links congestion in the network via the store-and-forward transfer mode. However, this cost of store operations is practical and will not be the bottleneck of transfers, because the storage capacities are much more available than the link capacities among geo-distributed datacenters.

6. Related work

There is a great deal of research on geographically distributed inter-datacenter bulk data transfers, with the goals of performing traffic characteristics analysis, improving bandwidth utilization and reducing the transfer costs [6,14,9,27,28]. Chen et al. [6] study the traffic data-sets of Yahoo! and characterize the inter-data center traffic characteristics. Laoutaris et al. [14] propose to complete the delay-tolerant bulk transfer via store-and-forward mode, such that the unused bandwidth can be utilized, while saving the transfer cost. They further extend their work in [9], based on predictable link capacity information. NetStitcher stitches together unused time-varying bandwidth across multiple datacenters for bulk transfers, so that existing link capacity is maximally utilized. The key limitation of this solution is that it only deals with a single bulk data transfer, and fails to tackle the more general problem of scheduling multiple bulk data transfers. Feng et al. [27] consider different charging schemes of ISPs on different links, aiming to minimize the operational costs on inter-datacenter traffic via scheduling bulk transfers with SnF. In [28], Nagdopal et al. take real-time data transfers into consideration, which own a higher priority to use peak bandwidth. Then, a scheduling algorithm is devised to reduce billable bandwidth usage. In [29], Losifidis et al. study the impact of storage capacity in time varying networks. They try to identify the bottleneck of storage capacity by computing the minimum cut of an expanding network, so that the storage resources can be managed effectively. However, their algorithm ignores that the expanding network size may become too large with the increase of given time horizon, which will lead to a high computational cost. In [30], Awerbuch et al. propose an approximation for the multi-commodity flow problem and local competitive
routing. Their algorithm is simple and runs effectively, however, they fail to leverage the delay tolerance of bulk data transfer. Their algorithm aims at improving the throughput of transfers, which will lead to the congestion of links being high. In [11], Chen et al. investigate the problem of scheduling deadline guaranteed bulk transfers in grids. They propose a technique to expand the temporal network status according to the start and end time of transfer requests. Unfortunately, their technique fails to consider the changes of network resources in the temporal dimension, which will lead to an infeasible scheduling strategy in dynamic networks. Besides, they only aim at minimizing the congestion of the link with the heaviest load, which ignores links that are not maximally loaded. Our work aims at lexicographically minimizing the entire network links congestion. We try to balance the bulk traffic in both spatial and temporal dimensions by carefully scheduling bulk data transfers, so that the network can experience low congestion, and can absorb the flash crowds. The algorithms are designed specifically for traffic engineering in datacenters networks with time-varying bandwidth and storage resources.

The closest research to lexicographical minimization of network congestion are the works in [31,25]. Georgiadis et al. [31] propose to lexicographically balance the backbone network load through recursively solving the min–max transportation problem on them. In [25] Nace et al. present max–min fairness and its application to load balancing in networks. However, these works pay no attention to the delay-tolerance feature of bulk data transfers, and their algorithms cannot be trivially adopted in the scenario of dynamic networks. Thus, our algorithms for scheduling multiple bulk data transfers are complementary to these works.

7. Conclusion

In this paper, we focus on minimizing the congestion of links between datacenters via scheduling bulk data transfers. We try to leverage the delay-tolerance feature to reduce the peak traffic load on links from temporal dimension via the store-and-forward transfer mode. We propose lexicographical minimization to balance the traffic load of the entire network links as much as possible. The Elastic Time-Expanded Network technique is designed to uniformly represent both the storage and link resource, while maintaining a reasonable expanded graph size. In addition, the corresponding optimal algorithm for multiple transfers is designed to obtain the lexicographically optimal solution. Extensive simulations show that the network congestion is significantly reduced when considering the store-and-forward transfer mode, and the bulk data traffic is balanced much better via the lexicographical minimization. Our research can be applied to the network resources to achieve advanced reservation for bulk data transfers. It can also help the network administrator manage the inter-datacenter bulk data traffic caused by data backup or migration among datacenters. In the future, we plan to extend our work to seek for more efficient analytical results for the general case of the problem.

References

Yiwen Wang is a Ph.D. Candidate from Beijing University of Posts and Telecommunications in China. His major is Computer Science. His research interests include network support for cloud computing and network traffic engineering.

Sen Su received the Ph.D. Degree in Computer Science from the University of Electronic Science and Technology, China, in 1998. He is currently a Professor at the Beijing University of Posts and Telecommunications. His research interests include distributed computing systems and the architecture of Internet service.

Alex X. Liu received his Ph.D. degree in Computer Science from the University of Texas at Austin in 2006. He received the IEEE & IFIP William C. Carter Award in 2004 and an NSF CAREER award in 2009. He received the Withrow Distinguished Scholar Award in 2011 at Michigan State University. He is an Associate Editor of IEEE/ACM Transactions on Networking. He received Best Paper Awards from ICNP-2012, SRDS-2012, and LISA-2010. His special research interests are in networking, security, and privacy. His general research interests include computer systems, distributed computing, and dependable systems.

Zhongbao Zhang is a Ph.D. Candidate from Beijing University of Posts and Telecommunications in China. His major is Computer Science. His research interests include network support for cloud computing and network virtualization.