This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2389842, IEEE Transactions on Cloud Computing

Orchestrating Bulk Data Transfers across Geo-Distributed Datacenters

Yu Wu[†], Zhizhong Zhang[†], Chuan Wu[†], Chuanxiong Guo[‡], Zongpeng Li[§], Francis C.M. Lau[†]
[†]Department of Computer Science, The University of Hong Kong, Email: {ywu,zzzhang,cwu,fcmlau}@cs.hku.hk
[‡] Microsoft Research Asia, Beijing, China, Email: chguo@microsoft.com
[§]Department of Computer Science, University of Calgary, Canada, Email: zongpeng@ucalgary.ca

Abstract—As it has become the norm for cloud providers to host multiple datacenters around the globe, significant demands exist for inter-datacenter data transfers in large volumes, *e.g.*, migration of big data. A challenge arises on how to schedule the bulk data transfers at different urgency levels, in order to fully utilize the available inter-datacenter bandwidth. The Software Defined Networking (SDN) paradigm has emerged recently which decouples the control plane from the data paths, enabling potential global optimization of data routing in a network. This paper aims to design a dynamic, highly efficient bulk data transfer service in a geo-distributed datacenter system, and engineer its design and solution algorithms closely within an SDN architecture. We model data transfer demands as delay tolerant migration requests with different finishing deadlines. Thanks to the flexibility provided by SDN, we enable dynamic, optimal routing of distinct chunks within each bulk data transfer (instead of treating each transfer as an infinite flow), which can be temporarily stored at intermediate datacenters to mitigate bandwidth contention with more urgent transfers. An optimal chunk routing optimization model is formulated to solve for the best chunk transfer schedules over time. To derive the optimal schedules in an online fashion, three algorithms are discussed, namely a bandwidth-reserving algorithm, a dynamically-adjusting algorithm, and a future-demand-friendly algorithm, targeting at different levels of optimality and scalability. We build an SDN system based on the Beacon platform and OpenFlow APIs, and carefully engineer our bulk data transfer algorithms in the system. Extensive real-world experiments are carried out to compare the three algorithms as well as those from the existing literature, in terms of routing optimality, computational delay and overhead.

Index Terms—Bulk data transfers, geo-distributed datacenters, software-defined networking

1 Introduction

Cloud datacenter systems that span multiple geographic locations are common nowadays, aiming to bring services close to users, exploit lower power cost, and enable service robustness in the face of network/power failures. Amazon, Google, Microsoft and Facebook have invested significantly in constructing large-scale datacenters around the globe, to host their services [1]. A basic demand in such a geo-distributed datacenter system is to transfer bulk volumes of data from one datacenter to another, *e.g.*, migration of virtual machines [2], replication of contents like videos [3], and aggregation of big data such as genomic data from multiple datacenters to one for processing using a MapReduce-like framework [4]. Despite dedicated broadband network connections being typically deployed among datacenters of the same cloud provider, the bulk data volumes involved in the inter-site transmissions are often high enough to overwhelm the backbone optical network, leading to bandwidth contention among disparate transmission tasks. The situation exacerbates at long-distance cross-continent submarine fiber links. A critical challenge is how to efficiently schedule the dynamically-arising, inter-datacenter transfer requests, such that transmission tasks of different urgency levels, reflected by different data transfer finishing deadlines, can be optimally and dynamically arranged to fully exploit the available bandwidth at any time.

Though a theoretical, online optimization problem in nature, the challenge could not be resolved without addressing the practical applicability of the optimization solution. That is: can an algorithm which solves the online optimization problem, if any, be practically realized in a real-world datacenter-to-datacenter network? It is not easy (if not impossible) to program a global optimization algorithm into a traditional distributed routing network like the Internet, given the lack of general programmability of switches/routers for running extra routing algorithms [5] (limited network programmability is only feasible through proprietary vendor-specific primitives) and the lack of the global view of the underlying network. The recent Software Defined Networking (SDN) paradigm has shed light on easy realization of a centralized optimization algorithm, like one that solves the bulk data transfer scheduling problem, using standard programming interfaces. With a logically central controller in place, the transient global network states, *e.g.*, topology, link capacity, *etc.*, can be more easily acquired by periodic inquiry messages, which are fundamental in practical SDN protocols, between the controller and the switches. For example, a common solution for topology discovery is that, the controller generates both Link Layer Discovery Protocol (LLDP) and Broadcast Domain Discovery Protocol (BDDP) messages and forwards them to all the switches; by identifying the receiving message types, the controller can recognize the active connections and derive the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2389842, IEEE Transactions on Cloud Computing

network topology. Furthermore, with the global knowledge, a centralized optimal scheduling algorithm can be realized in the controller, which would be otherwise impossible in its traditional distributed routing counterpart.

Software defined networking advocates a clean decoupling of the control path from the data path in a routing system [6]. By allowing per-flow routing decisions at the switches/routers, it empowers the network operators with more flexible traffic management capabilities, which are potentially QoS-oriented and globally optimal. To realize the SDN paradigm, standards like OpenFlow have been actively developed [7], which define standard communication interfaces between the control and data layers of an SDN architecture. IT giants including Google and Facebook have advocated the OpenFlow-based SDN architecture in their datacenter systems [8] [9], while switch vendors including Broadcom, HP and NEC have begun production of OpenFlow-enabled switches/routers in the past 2-3 years [10], aiming towards a new era of easy network programmability.

This paper proposes a novel optimization model for dynamic, highly efficient scheduling of bulk data transfers in a geo-distributed datacenter system, and engineers its design and solution algorithms practically within an OpenFlow-based SDN architecture. We model data transfer requests as delay tolerant data migration tasks with different finishing deadlines. Thanks to the flexibility of transmission scheduling provided by SDN, we enable dynamic, optimal routing of distinct chunks within each bulk data transfer (instead of treating each transfer as an infinite flow), which can be temporarily stored at intermediate datacenters and transmitted only at carefully scheduled times, to mitigate bandwidth contention among tasks of different urgency levels. Our contributions are summarized as follows.

First, we formulate the bulk data transfer problem into a novel, optimal chunk routing problem, which maximizes the aggregate utility gain due to timely transfer completions before the specified deadlines. Such an optimization model enables flexible, dynamic adjustment of chunk transfer schedules in a system with dynamically-arriving data transfer requests, which is impossible with a popularly-modeled flow-based optimal routing model.

Second, we discuss three dynamic algorithms to solve the optimal chunk routing problem, namely a bandwidth-reserving algorithm, a dynamically-adjusting algorithm, and a future-demand-friendly algorithm. These solutions are targeting at different levels of optimality and computational complexity.

Third, we build an SDN system based on the OpenFlow APIs and Beacon platform [11], and carefully engineer our bulk data transfer algorithms in the system. Extensive real-world experiments with real network traffic are carried out to compare the three algorithms as well as those in the existing literature, in terms of routing optimality, computational delay and overhead.

In the rest of the paper, we discuss related work in Sec. 2, illustrate our system architecture and the optimization framework in Sec. 3, and present the dynamic algorithms in Sec. 4. Details of our SDN system implementation follow in Sec. 5. Experiment settings and results are reported in Sec. 6. Sec. 7 concludes the paper.

2 Related Work

In the network inside a data center, TCP congestion control and FIFO flow scheduling are currently used for data flow transport, which are unaware of flow deadlines. A number of proposals have appeared for deadline-aware congestion and rate control. D^3 [12] exploits deadline information to control the rate at which each source host introduces traffic into the network, and apportions bandwidth at the routers along the paths greedily to satisfy as many deadlines as possible. D2TCP [13] is a Deadline-Aware Datacenter TCP protocol to handle bursty flows with deadlines. A congestion avoidance algorithm is employed, which uses ECN feedback from the routers and flow deadlines to modify the congestion window at the sender. In pFabric [14], switches implement simple priority-based scheduling/dropping mechanisms, based on a priority number carried in the packets of each flow, and each flow starts at the line rate which throttles back only when high and persistent packet loss occurs. Differently, our work focuses on transportation of bulk flows among datacenters in a geo-distributed cloud. Instead of end-to-end congestion control, we enable store-and-forward in intermediate datacenters, such that a source data center can send data out as soon as the first-hop connection bandwidth allows, whereas intermediate datacenters can temporarily store the data if more urgent/important flows need the next-hop link bandwidths.

Inter-datacenter data transfer is also common today. Chen *et al.* [15] conducted a measurement study of a cloud with five distributed datacenters and revealed that more than 45% of the total traffic is attributed to inter-datacenter transmissions, and the percentage is expected to grow further. Laoutaris *et al.* propose NetStitcher [16], a mechanism exploiting *a priori* knowledge of the traffic patterns across datacenters over time and utilizing the leftover bandwidth and intermediate storage between datacenters for bulk data transfer, to minimize the transmission time of a given volume. In contrast, our study focuses on flows with stringent deadlines, and will not assume any traffic patterns. We apply optimization algorithms to dynamically adjust flow transfer schedules under any traffic arrival patterns. Postcard [17] models a cost minimization problem for inter-datacenter traffic scheduling, based on the classic time expanded graph [18] which was first used in NetStitcher. Though relatively easy to formulate, the state explosiveness of the optimization model, when replicating nodes and links along the time axis, results in a prohibitive growth rate of the computation complexity. Our work seeks to present a novel optimization model which can enable efficient dynamic algorithms for practical deployment in an SDN network. Chen *et al.* [19] study deadline-constrained bulk data transfer in grid networks. Our work differs from theirs by concentrating on a per-chunk routing scheme instead of treating each transfer as flows, which renders itself to a more realistic model with higher complexity in algorithm design. In addition, we assume dedicated links between datacenters owned by the cloud provider, and aim to maximize the overall transfer utility instead of minimizing the network congestion.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2389842, IEEE Transactions on Cloud Computing

In an SDN-based datacenter, Helter *et al.* [20] design ElasticTree, a power manager which dynamically adjusts the set of active links and switches to serve the changing traffic loads, such that the power consumption in the datacenter network is minimized. For SDN-based inter-datacenter networking, Jain *et al.* [9] present their experience with B4, Google's globally deployed software defined WAN, connecting Google's datacenters across the planet. They focus on the architecture and system design, and show that with a greedy centralized traffic engineering algorithm, all WAN links can achieve an average 70% utilization. Hong *et al.* [21] propose SWAN, a system that centrally controls the traffic flows from different services in an inter-datacenter network. Their work focuses more on coordinating routing policy updates among the switches to avoid transient congestion. Falling into the similar line of research for boosting inter-datacenter bandwidth utilization, our research focuses more on scheduling of bulk data transfers to meet deadlines, and complement these existing work well by proposing efficient dynamic optimization algorithms, to guarantee long-term optimal operation of the network.

Deadline-aware resource scheduling in clouds has attracted growing research interest. A recent work from Maria *et al.* [22] presents a meta-heuristic optimization based algorithm to address the resource provisioning (VM) and scheduling strategy in IaaS clouds to meet the QoS requirements. We believe that our work well complements those in this category. In addition, our work focuses on bulk data flows instead of small flows as we echo the argument of Curtis *et al.* [23] that in reality only "significant" flows (*e.g.*, high-throughput "elephant" flows) should be managed by a centralized controller, in order to reduce the amount of switch-controller communication.

3 System Architecture and Problem Model

3.1 SDN-based Architecture

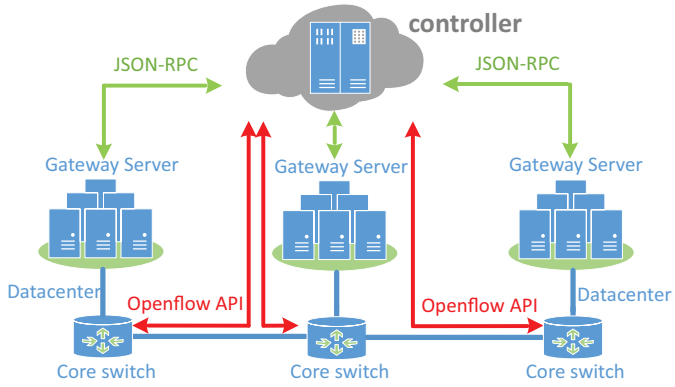


Fig. 1. The architecture of the system.

We consider a cloud spanning multiple datacenters located in different geographic locations (Fig. 1). Each datacenter is connected via a core switch to the other datacenters. The connections among the datacenters are dedicated, full-duplex links, either through leading tier-1 ISPs or private fiber networks of the cloud provider, allowing independent and simultaneous two-way data transmissions. Data transfer requests may arise from each datacenter to move bulk volumes of data to another datacenter. A gateway server is connected to the core switch in each datacenter, responsible for aggregating cross-datacenter data transfer requests from the same datacenter, as well as for temporarily storing data from other datacenters and forwarding them via the switch. It also tracks network topology and bandwidth availability among the datacenters with the help of the switches. Combined closely with the SDN paradigm, a central controller is deployed to implement the optimal data transfer algorithms, dynamically configure the flow table on each switch, and instruct the gateway servers to store or to forward each data chunk. The layered architecture we present realistically resembles *B4* [9], which was designed and deployed by Google for their G-scale inter-datacenter network: the gateway server plays a similar role of the *site controller layer*, the controller corresponds well to the *global layer*, and the core switch at each location can be deemed as the per-site switch clusters in *B4*.

The fundamental core services enabling bulk data transfer in a geo-distributed cloud include:

Task admission control. Once a data transfer task is admitted, we seek to ensure its timely completion within the specified deadline. On the other hand, if completing a transfer task within the specific deadline is not possible according to the network availability when the request arrives, the task should be rejected.

Data routing. The optimal transmission paths of the data in an accepted task from the source to the destination should be decided, potentially through multiple intermediate datacenters.

Store-and-forward. Intermediate datacenters may store the data temporarily and forward them later. It should be carefully computed when a data should be temporarily stored in which datacenter, as well as when and at which rate it should be forwarded at a later time.

The goal of judiciously making the above decisions is to maximize the overall utility of tasks, by best utilizing the available bandwidth along the inter-datacenter links at any given time.

3.2 Problem Model

Let $\mathbb{N}$ represent the set of all datacenters in the system. A data transfer request (or task equivalently) J can be described by a five-tuple $(S_J, D_J, t_J, T_J, \mathbb{U}_J)$, where S_J is the source datacenter where the data originates, D_J is the destination datacenter where the data is to be moved to, t_J is the earliest time J can be transmitted, and T_J denotes the maximum amount of time allowed for transfer of task J to complete, *i.e.*, all data of task J should arrive at the destination no later than $t_J + T_J$. $\mathbb{U}_J$ is a weight modeling the benefit of completing job J, and jobs with higher importance are associated with larger weights.

The system runs in a time-slotted fashion. The data of each job J is segmented into equal-sized chunks at the source datacenter before their transmission, and $\mathbb{W}_J$ denotes the corresponding chunk set. Consider the system lifespan $[0, \Gamma]$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2389842, IEEE Transactions on Cloud Computing

Let $\mathbb{J}$ denote the set of all jobs which arrive and are to be completed in this span. The binary variable I_J denotes if task J is accepted or not, and $x_{m,n}^{(w)}(t)$ indicates whether chunk w is transmitted from datacenter m to datacenter n at time t. The available bandwidth of the connection from datacenter m to datacenter n is described by $B_{m,n}$, as a multiple of a unit bandwidth. The length of a scheduling interval (*i.e.*, each time slot) in the system equals the transmission time of a chunk using a unit bandwidth. Hence, $B_{m,n}$ is equivalent to the maximum number of chunks that can be delivered from m to n in a time slot.

We consider chunk sizes at tens of megabytes in such bulk data transfer, and a unit bandwidth at the magnitude of tens of Mbps, since the dedicated links between datacenters have typical bandwidths up to 100 Gbps [24]. In this case, the scheduling interval length is at tens of seconds, which is reasonable since it may not be feasible in practice to adjust flow tables at the switches more frequently than that. Propagation delays and potential chunk queuing delays are ignored as they are dominated by the transmission times in bulk data transfer, at the magnitudes of hundreds of milliseconds.

Table 1 summarizes important notation for ease of reference.

TABLE 1
Table of notations

Symbol	Definition
$\mathbb{J}$	The set of jobs to be completed within time interval $[0, \Gamma]$
$\mathbb{J}(\tau)$	The set of jobs arrived at time τ
$\mathbb{J}(\tau^-)$	The set of unfinished, previously accepted job by time τ
$\mathbb{U}_J$	The weight of job J
t_J	The earliest time slot for transmission of J
T_J	The maximum number of time slots allowed for transmission of J
S_J	The source datacenter where J originates
D_J	The destination datacenter where J is to be migrated to
$\mathbb{W}_J$	The chunk set of job J
$\mathbb{N}$	The set of all datacenters
$B_{m,n}$	The available bandwidth from datacenter m to n
I_J	Whether job J is accepted
$x_{m,n}^{(w)}(t)$	Whether transmit chunk w from datacenter m to n at time t

3.3 The Optimal Chunk Routing Problem

We formulate the problem to an optimization framework to derive the job acceptance decisions $I_J, \forall J \in \mathbb{J}$, and chunk routing decisions $x_{m,n}^{(w)}(t)$, $\forall w \in W_J, \forall t \in [t_J, t_J + T_J - 1], \forall J \in \mathbb{J}, \forall m, n \in \mathbb{N}, m \neq n$.

$$\max \sum_{J \in \mathbb{J}} \mathbb{U}_J \times I_J \tag{1}$$

subject to:

$$(a) \quad \sum_{t=t_J}^{t_J+T_J-1} \sum_{m \in \mathbb{N}, m \neq S_J} (x_{m,S_J}^{(w)}(t) - x_{S_J,m}^{(w)}(t)) = -I_J, \quad \forall w \in \mathbb{W}_J, \forall J \in \mathbb{J};$$

$$(b) \quad \sum_{t=t_J}^{t_J+T_J-1} \sum_{m \in \mathbb{N}, m \neq D_J} (x_{m,D_J}^{(w)}(t) - x_{D_J,m}^{(w)}(t)) = I_J, \quad \forall w \in \mathbb{W}_J, \forall J \in \mathbb{J};$$

$$(c) \quad \sum_{t=t_J}^{t_J+T_J-1} \sum_{m \in \mathbb{N}, m \neq n} (x_{m,n}^{(w)}(t) - x_{n,m}^{(w)}(t)) = 0, \quad \forall n \in \mathbb{N}/\{S_J, D_J\}, \forall w \in \mathbb{W}_J, \forall J \in \mathbb{J};$$

$$(d) \quad \sum_{t=t_J}^{T_0} \sum_{m \in \mathbb{N}, m \neq n} x_{m,n}^{(w)}(t) \geq \sum_{t=T_0+1}^{t_J+T_J-1} \sum_{k \in \mathbb{N}, k \neq n} x_{n,k}^{(w)}(t), \quad \forall w \in \mathbb{W}_J, \forall n \in \mathbb{N}/S_J, \forall T_0 \in [t_J, t_J + T_J - 2], \forall J \in \mathbb{J};$$

$$(e) \quad \sum_{J \in \mathbb{J}} \sum_{w \in \mathbb{W}_J} x_{m,n}^{(w)}(t) \leq B_{m,n}, \quad \forall m, n \in \mathbb{N}, m \neq n, \forall t \in [0, \Gamma];$$

$$(f) \quad x_{m,n}^{(w)}(t) \in \{0, 1\}, \quad \forall m, n \in \mathbb{N}, m \neq n, \forall t \in [t_J, t_J + T_J - 1], \forall w \in \mathbb{W}_J, \forall J \in \mathbb{J};$$

$$(g) \quad x_{m,n}^{(w)}(t) = 0, \quad \forall m, n \in \mathbb{N}, m \neq n, \forall t \in [0, t_J) \cup (t_J + T_J - 1, \Gamma], \forall w \in \mathbb{W}_J, \forall J \in \mathbb{J}.$$

The objective function maximizes the overall weight of all the jobs to be accepted. A special case where $\mathbb{U}_J(\cdot) = 1$ $(\forall J \in \mathbb{J})$ implies the maximization of the total number of accepted jobs. Constraint (a) states that for each chunk w in each job J, it should be sent out from the source datacenter S_J at one time slot within $[t_J, t_J + T_J - 1]$ (*i.e.*, the valid transmission interval of the job), if it is accepted for transfer at all in the system (*i.e.*, if I_J=1); on the other hand, the chunk should arrive at the destination datacenter D_J via one of D_J's neighboring datacenters within $[t_J, t_J+T_J]$ as well, as specified by Constraint (b). Constraint (c) enforces that at any intermediate datacenter n other than the source and destination of chunk w, if it receives the chunk at all in one time slot within the valid transmission interval of the job, it should send the chunk out as well within the interval.

With constraint (d), we ensure that a chunk should arrive at a datacenter earlier before it can be forwarded from the datacenter, *i.e.*, considering any time slot T_0 within the valid transmission interval $[t_J, t_J + T_J - 1]$ of job J, a datacenter n may send out chunk w in a time slot after T_0 (*i.e.*, $\sum_{t=T_0+1}^{t_J+T_J-1} \sum_{k \in \mathbb{N}, k \neq n} x_{n,k}^{(w)}(t) = 1$), only if it has received it earlier by T_0 (*i.e.*, $\sum_{t=t_J}^{T_0} \sum_{m \in \mathbb{N}, m \neq n} x_{m,n}^{(w)}(t) = 1$).

Constraint (e) specifies that the total number of chunks from all jobs to be delivered from datacenter m to n in any time slot t, should not exceed the bandwidth capacity of the connection between m and n. Routing decisions for a chunk w, $x_{m,n}^{(w)}(t)$'s, are binary, *i.e.*, either sent along connection from m to n in t ($x_{m,n}^{(w)}(t) = 1$) or not ($x_{m,n}^{(w)}(t) = 0$), and valid only within the valid transmission interval of the corresponding job, as given by constraints (f) and (g).

The solutions of our optimization framework translate to reliable routing decisions in the sense that any accepted job

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2389842, IEEE Transactions on Cloud Computing

will be delivered to the destination within the corresponding deadline. Those rejected jobs are dropped immediately at the beginning to save bandwidth, but users may resubmit the jobs at later times.

The structure of the optimization problem is similar to that in a max flow or min-cost flow problem [25], but the difference is apparent as well: we model routing of distinct chunks which can be stored at intermediate datacenters and forwarded in a later time slot, instead of continuous end-to-end flows; therefore, the time dimension is carefully involved to specify the transfer times of the chunks, which does not appear in a max/min-cost flow problem.

The optimization model in (1) is an offline optimization problem in nature. Given any job arrival pattern within $[0, \Gamma]$, it decides whether each job should be accepted for transfer under bandwidth constraints, and derives the best paths for chunks in accepted jobs, along which the chunks can reach their destinations within the respective deadlines. In practice when the transfer jobs are arriving one after another, an online algorithm to make timely decisions on job admission control and routing scheduling is desirable, which we will investigate in the next section.

4 Dynamic Algorithms

We present three practical algorithms which make job acceptance and chunk routing decisions in each time slot, and achieve different levels of optimality and scalability.

4.1 The Bandwidth-Reserving Algorithm

The first algorithm honors decisions made in previous time slots, and reserves bandwidth along the network links for scheduled chunk transmissions of previously accepted jobs in its routing computation for newly arrived jobs. Let $\mathbb{J}(\tau)$ be the set consisting of only the latest data transfer requests arrived in time slot τ. Define $B_{m,n}(t)$ as the residual bandwidth on each connection (m, n) in time slot $t \in [\tau+1, \Gamma]$, excluding bandwidth needed for the remaining chunk transfers of accepted jobs arrived before τ. In each time slot τ, the algorithm solves optimization (1) with job set $\mathbb{J}(\tau)$ and bandwidth $B_{m,n}(t)$'s for duration $[\tau+1, \Gamma]$, and derives admission control decisions for jobs arrived in this time slot, as well as their chunk transfer schedules before their respective deadlines.

Theorem 1 states the NP-hardness of optimization problem in (1) (with detailed proof in Appendix A). Nevertheless, such a linear integer program may still be solved in reasonable time at a typical scale of the problem (*e.g.*, tens of datacenters in the system), using an optimization tool such as CPLEX [26]. To cater for larger scale problems, we also propose a highly efficient heuristic in Sec. 4.3. More detailed discussions of the solution time follow in Sec. 6.3.

Theorem 1: The optimal chunk routing problem in 1 is NP-hard.

4.2 The Dynamically-Adjusting Algorithm

The second algorithm retains job acceptance decisions made in previous time slots, but adjusts routing schedules for chunks of accepted jobs, which have not reached their respective destinations, together with the admission control and routing computation of newly arrived jobs. Let $\mathbb{J}(\tau)$ be the set of data transfer requests arrived in time slot τ, and $\mathbb{J}(\tau^-)$ represent the set of unfinished, previously accepted jobs by time slot τ. In each time slot τ, the algorithm solves a modified version of optimization (1), as follows:

- The set of jobs involved in the computation is $\mathbb{J}(\tau) \cup \mathbb{J}(\tau^-)$.
- The optimization decisions to make include: (i) acceptance of newly arrived jobs, *i.e.*, $I_J, \forall J \in \mathbb{J}(\tau)$; (ii) routing schedules for chunks in newly arrived jobs, *i.e.*, $x^{(w)}_{m,n}(t), \forall m, n \in \mathbb{N}, m \neq n$, $\forall t \in [\tau+1, \Gamma]$, $\forall w \in W_J$ where $J \in \mathbb{J}(\tau)$; (iii) routing schedules for chunks in previously accepted jobs that have not reached their destinations, *i.e.*, $x^{(w)}_{m,n}(t), \forall m, n \in \mathbb{N}, m \neq n$, $\forall t \in [\tau+1, \Gamma]$, $\forall w \in W'_J$, where $J \in \mathbb{J}(\tau^-)$ and W'_J denotes the set of chunks in J which have not reached the destination D_J.
- For each previously accepted job $J \in \mathbb{J}(\tau^-)$, we set $I_J = 1$ wherever it appears in the constraints, such that the remaining chunks in these jobs are guaranteed to arrive at their destinations before the deadlines, even after their routing adjustments.
- Constraints related to chunks in previously accepted jobs, which have reached their destinations, are removed from the optimization problem.
- For each remaining job $J \in \mathbb{J}(\tau^-)$, its corresponding t_J and T_J used in the constraints should be replaced by $t'_J = \tau + 1$ and $T'_J = t_J + T_J - 1 - \tau$, given that the decision interval of the optimization problem has been shifted to $[\tau+1, \Gamma]$.
- For a chunk w in W'_J, it may have been transferred to an intermediate datacenter by time slot τ, and hence multiple datacenters may have cached a copy of this chunk. Let $\Omega(w)$ be the set of datacenters which retain a copy of chunk w. The following optimal routing path of this chunk can originate from any of the copies. Therefore, in constraints (a)(c)(d) on chunk w, we replace S_J by $\Omega(w)$, *e.g.*, constraint (a) on chunk w is modified to

$$\sum_{t=t'_J}^{t'_J+T'_J-1} \sum_{n\in\Omega(w)} \sum_{m\in\mathbb{N}, m\neq n} (x^{(w)}_{m,n}(t) - x^{(w)}_{n,m}(t)) = -I_J, \quad \forall w \in \mathbb{W}'_J, \forall J \in \mathbb{J}(\tau) \cup \mathbb{J}(\tau^-). \tag{2}$$

The detailed formulation of the modified optimization problem is given in (3).

$$max \sum_{J\in\mathbb{J}(\tau)\cup\mathbb{J}(\tau^-)} \mathbb{U}_J \times I_J \tag{3}$$

subject to:

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2389842, IEEE Transactions on Cloud Computing

$$(a) \quad \sum_{t=t'_J}^{t'_J+T'_J-1} \sum_{n\in\Omega(w)} \sum_{m\in\mathbb{N},m\neq n} (x^{(w)}_{m,n}(t) - x^{(w)}_{n,m}(t)) = -I_J, \quad \forall w \in \mathbb{W}'_J, \forall J \in \mathbb{J}(\tau) \cup \mathbb{J}(\tau^-);$$

$$(b) \quad \sum_{t=t'_J}^{t'_J+T'_J-1} \sum_{m\in\mathbb{N},m\neq D_J} (x^{(w)}_{m,D_J}(t) - x^{(w)}_{D_J,m}(t)) = I_J, \quad \forall w \in \mathbb{W}'_J, \forall J \in \mathbb{J}(\tau) \cup \mathbb{J}(\tau^-);$$

$$(c) \quad \sum_{t=t'_J}^{t'_J+T'_J-1} \sum_{m\in\mathbb{N},m\neq n} (x^{(w)}_{m,n}(t) - x^{(w)}_{n,m}(t)) = 0, \quad \forall n \in \mathbb{N}/\{\Omega(w), D_J\}, \forall w \in \mathbb{W}'_J, \forall J \in \mathbb{J}(\tau) \cup \mathbb{J}(\tau^-);$$

$$(d) \quad \sum_{t=t'_J}^{T_0} \sum_{m\in\mathbb{N},m\neq n} x^{(w)}_{m,n}(t) \geq \sum_{t=T_0+1}^{t'_J+T'_J-1} \sum_{k\in\mathbb{N},k\neq n} x^{(w)}_{n,k}(t), \quad \forall w \in \mathbb{W}'_J, \forall n \in \mathbb{N}/\Omega(w), \forall T_0 \in [t'_J, t'_J+T'_J-2], \forall J \in \mathbb{J}(\tau) \cup \mathbb{J}(\tau^-);$$

$$(e) \quad \sum_{J\in\mathbb{J}(\tau)\cup\mathbb{J}(\tau^-)} \sum_{w\in\mathbb{W}'_J} x^{(w)}_{m,n}(t) \leq B_{m,n} \quad \forall m,n \in \mathbb{N}, m \neq n, \forall t \in [\tau+1, \Gamma];$$

$$(f) \quad x^{(w)}_{m,n}(t) = \{0,1\} \quad \forall m,n \in \mathbb{N}, m \neq n, \forall t \in [t'_J, t'_J+T'_J-1], \forall w \in \mathbb{W}'_J, \forall J \in \mathbb{J}(\tau) \cup \mathbb{J}(\tau^-);$$

$$(g) \quad x^{(w)}_{m,n}(t) = 0, \quad \forall m,n \in \mathbb{N}, m \neq n, \forall t \in [\tau+1, t'_J) \cup (t'_J+T'_J-1, \Gamma], \quad \forall w \in \mathbb{W}'_J, \forall J \in \mathbb{J}(\tau) \cup \mathbb{J}(\tau^-);$$

$$(h) \quad I(J) = 1, \quad \forall J \in \mathbb{J}(\tau^-).$$

Here in the constraints, for newly arrived jobs in $\mathbb{J}(\tau)$, we set $t'_J = t_J$ and $T'_J = T_J$.

This second algorithm is more aggressive than the first one in computing the best routing paths for all remaining chunks in the system, both from the newly arrived jobs and from old, unfinished jobs. More computation is involved. Nonetheless, we will show in Sec. 6.3 that the solution can still be derived in a reasonable amount of time under practical setting with heavy data transfer traffic. It's worthy to note that the optimal solution of of the two optimization problems (1)(3) may not be unique, and we randomly select one for the routing decision. It could be interesting to study which optimal solution is even better in an online fashion, if the job request pattern is known beforehand or can be predicted. This will be part of our future work.

4.3 The Future-Demand-Friendly Heuristic

We further propose a simple but efficient heuristic to make job acceptance and chunk routing decisions in each time slot, with polynomial-time computational complexity, suitable for systems with larger scales. Similar to the first algorithm, the heuristic retains routing decisions computed earlier for chunks of already accepted jobs, but only makes decisions for jobs received in this time slot using the remaining bandwidth. On the other hand, it is more future demand friendly than the first algorithm, by postponing the transmission of accepted jobs as much as possible, to save bandwidth available in the immediate future in case more urgent transmission jobs may arrive.

Let $\mathbb{J}(\tau)$ be the set of latest data transfer requests arrived in time slot τ. The heuristic is given in Alg. 1. At the job level, the algorithm preferably handles data transfer requests with higher weights and smaller sizes (line 1), *i.e.*, larger weight per unit bandwidth consumption. For each chunk in job J, the algorithm chooses a transfer path with the fewest number of hops, that has available bandwidth to forward the chunk from the source to the destination before the deadline (line 4). The rationale is that a path with fewer hops consumes less overall bandwidth to transfer a chunk and with higher probability to meet the deadline requirement. We compute the allowed time window for the transfer of a chunk to happen at the i-th hop along its potential path, by dividing the allowed overall transfer time T_J evenly among the h hops of the path. As shown in Fig. 2, the transfer of the chunk from the source to the first-hop intermediate datacenter should happen within time window $[t_J, t_J + \frac{T_J}{h} - 1]$, and the transfer at the second hop should happen within time window $[t_J+\frac{T_J}{h}, t_J+\frac{2T_J}{h}-1]$, and so on, in order to guarantee the chunk's arrival at the destination before the deadline of t_J+T_J. The path should be one such that there is at least one unit available bandwidth at each hop within at least one time slot in the allowed transmission window of this hop (line 4). If such a path does not exist, the job should be rejected (line 6). Otherwise, the algorithm computes the chunk transfer schedule at each hop along the selected path, by choosing one time slot when available bandwidth exists on the link, within the allowed transmission window of that hop. There can be multiple time slots satisfying these requirements, and the latest time slot among them is selected with the highest probability (line 10). If the transfer schedules for all chunks in the job can be successfully made, the job is accepted (line 13).

Algorithm 1 The Future-Demand-Friendly Heuristic at Time Slot τ

1: Sort requests in $\mathbb{J}(\tau)$ by $\frac{\mathbb{U}_J}{|\mathbb{W}_J|}$ in descending order.
2: **for** each job J in sorted list $\mathbb{J}(\tau)$ **do**
3: **for** each chunk $w \in \mathbb{W}_J$ **do**
4: Find a shortest path from S_J to D_J that satisfies the following (suppose the path includes h hops): there is one unit bandwidth available at the i-th hop link ($1 \leq i \leq h$) in at least one time slot within the time frame $[t_J + (i-1) \times \frac{T_J}{h}, t_J + i \times \frac{T_J}{h} - 1]$. List all the time slots in the frame when there is one unit available bandwidth along the i-th hop link as $\tau^i_1, \tau^i_2, \ldots, \tau^i_L$.
5: **if** such a path does not exist **then**
6: Reject J, *i.e.*, set $I_J = 0$, and clear the transmission schedules made for other chunks in J;
7: **break**;
8: **end if**
9: **for** each hop (m, n) along the shortest path **do**
10: suppose it is the i-th hop; choose the r-th time slot in the list $\tau^i_1, \tau^i_2, \ldots, \tau^i_L$ with probability $\frac{r}{\sum_{p=1}^{L} p}$; set $x^w_{m,n}(\tau^i_r) = 1$ and $x^w_{m,n}(t) = 0$, $\forall t \neq \tau^i_r$ (*i.e.*, transfer chunk w from m to n at time slot τ^i_r).
11: **end for**
12: **end for**
13: Accept J, *i.e.*, set $I_J = 1$.
14: **end for**

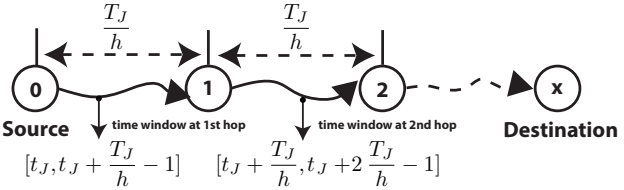


Fig. 2. Assigning allowed time window for a chunk's transfer at each hop along its transmission path.

Alg. 1 tends to be conservative in chunk forwarding by

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2389842, IEEE Transactions on Cloud Computing

scheduling the transmissions at a later time possible, with the rationale to save bandwidth for more urgent transfer requests that could arrive in the immediate future. The downside is that it may leave links idle at earlier times. We further design a supplementary algorithm to expedite chunk transfer in such scenarios, as given in Alg. 2. The idea is simple: the algorithm transfers chunks with pending future transmission schedules along a link, to fully utilize the available link bandwidth at each time slot. Such opportunistic forward-shifts of scheduled transmissions reduce future bandwidth consumption in the system, such that potentially more job transfer requests can be accommodated.

Algorithm 2 Opportunistic Chunk Transfer in Each Time Slot

```
for each link (m, n) in the system do
    while there is available bandwidth on (m, n) do
        if there exists a chunk w which has been received at datacenter m and is scheduled to be forwarded to datacenter n at a later time then
            Send w from m to n.
        else
            break
        end if
    end while
end for
```

Theorem 2 guarantees that the proposed practical heuristic has a polynomial-time complexity (with detailed proof in Appendix B). We also study the performance of this heuristic in Sec. 6, and compare it with the other optimization based solutions.

Theorem 2: The practical heuristic described in Alg. 1 and Alg. 2 has polynomial-time complexity.

5 System Implementation

We have implemented a prototype bulk data transfer (BDT) system based on the OpenFlow framework. The BDT system resides above the transport layer in the network stack, with no special requirements on the lower transport layer and network layer protocols, where the standard TCP/IP stack is adopted. Due to the clean design that will be introduced in this section, the BDT system is implemented with only 7K lines of Java code and around 2K lines of Python code, apart from the codes of the configuration files, the GUI control interface, etc.

5.1 The Key Components

Fig. 3 depicts the main modules of the system, consisting of both the central Controller and the distributed Gateway Servers. Common in all SDN-based systems, the centralized controller could become the performance bottleneck of the system. To realize fine-grained dynamic traffic scheduling, it needs to keep track of global state information, which translates to stressful resource consumption hindering horizontal scalability, in terms of both memory footprints and processing cycles. We therefore follow the common design philosophy by splitting part of the sophistications to the end system, *i.e.*, the gateway server, while keeping only necessary tasks to the core system, *i.e.*, the controller.

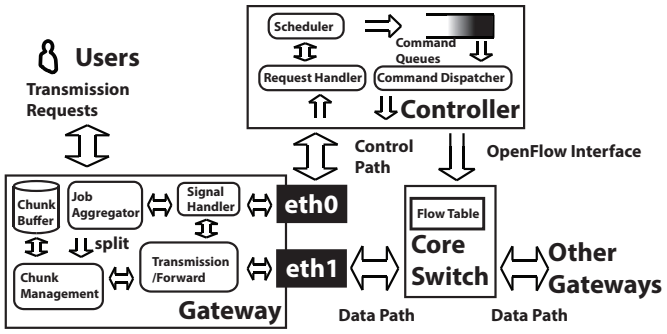


Fig. 3. The main components in the BDT system.

▷ **Controller**: The controller is built based on the Beacon framework [11], capitalizing its rich features including cross-platform portability and multi-thread support. The *Request Handler* listens to transmission requests from the gateway servers at different datacenters, each of which is responsible for collecting data transfer requests from the datacenter it resides in. For each incoming request, a unique global "job id" is assigned and returned to the corresponding gateway server. The "job id" will be reclaimed later once either the request is rejected or the data transfer is completed. The *Scheduler* periodically retrieves the received requests and computes the chunk routing schedules, following one dynamic algorithm we design. Each accepted chunk is labelled with a timestamp indicating when its transmission should begin, and the transmission command is pushed to a command queue. The *Command Dispatcher* further periodically retrieves the commands with timestamps no later than the current time from the command queue, and sends them to the corresponding gateway servers and core switches. We implement the *Command Dispatcher* in the "Adaptor" design pattern, which can be easily customized to be compliant with different versions of OpenFlow specifications. Currently, our system is compliant with OpenFlow version 1.0.0, supported by most off-the-shelf commodity switch products.

▷ **Gateway Server**: Each gateway server is attached to two separate networks, one (eth0) is for control signaling with the controller, and the other (eth1) is for data transmission. Traffic along the control path is significantly less than that along the data path, since all signaling is via plain light-weight JSON messages. The data path corresponds exactly to the dedicated links between the datacenters. The *Job aggregator* collects all the incoming data transfer requests from within the same datacenter, and forwards them to the controller to schedule. *Buffer Management* plays the role of managing all received chunks at the gateway server, including those destined to this datacenter and those temporarily stored while waiting to be forwarded by the *Transmission/Forward* module. The *Transmission/Forward* module functions by following the transmission instructions issued from the controller.

For each transmission instruction, a new thread is created to transmit the corresponding chunk. Therefore, multiple transmissions can happen concurrently, and the maximal number of transmission threads allowed are constrained by the link capacity. Actual data transmissions happen directly between two gateway servers, and the sender gateway server should first

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2389842, IEEE Transactions on Cloud Computing

notify the recipient gateway server of the "identity" (job id, chunk id) of the chunk transmitted at the same TCP connection to facilitate the management at the *Buffer Management* module at the latter. The *Signal Handler* module handles the signals communicated with the controller via the control path. Multiple signals are exchanged between the controller and a gateway server. For instance, the controller instructs the gateway server which chunks to transmit to which datacenters as the next hop at the current moment, and the gateway server may notify the controller of any transmission failure due to network failures.

▷ **Core Switch**: The core switch wires the co-located gateway server with the other gateway servers through the dedicated links. The controller reconfigures the flow tables inside each core switch via standard OpenFlow APIs, according to the calculated scheduling decisions.

Fig. 4 shows a sequence diagram describing the primary control signals between different modules at the controller and a gateway server. Two important background threads are constantly running (marked with "loop" in Fig. 4) at the controller: (1) the *scheduler* thread periodically collects all the received requests during the last time slot to calculate the scheduling decisions, and inserts the results to the command queue; (2) The *command dispatcher* thread periodically retrieves the generated routing decisions and forwards them to the corresponding gateway servers and core switches.

5.2 Other Design Highlights

Dynamic component model. The modules in our system are integrated into the Java OSGi framework [27] as independent bundles, which can be deployed and upgraded at runtime, without shutting down the controller. More specifically, we select Equinox [28] as both our development and deployment platform, which is a certified implementation of OSGi R4.x core framework specification.

Feasibility in the production network. By integrating FlowVisor[29] as the transparent proxy between the controller and the core switch, we can logically "slice" dedicated bandwidths out of the physical links for transmissions, achieving a rigorous performance isolation from the ordinary traffic in the production network. By carefully designing the "flow spaces", users can specify whether their traffic will go through the BDT network or the traditional "best effort" one.

Easy deployment of new scheduling algorithm. Our controller implementation features a simple interface *BDT_Schedulable* for incorporation of different scheduling algorithms. Due to space limit, readers are referred to Appendix C for more highlights of our system design. For instance, our future-demand-friendly heuristic is implemented with less than 600 lines of code.

Efficient handling of overwhelming numbers of TCP connections. A standard GNU/Linux distribution (e.g., Gentoo in our cases) aims to optimize TCP performance in a wide range of environments, which may not be optimal in our system deployment. Therefore, we have carefully configured the TCP buffer and tuned the kernel performance parameters to optimize the bulk transfer performance in our system. Some key parameters are listed in Table 2.

net.ipv4.tcp_congestion_control	cubic
net.core.somaxconn	8192
net.ipv4.tcp_tw_recycle	1
net.ipv4.tcp_tw_reuse	1
net.ipv4.tcp_fack	1

TABLE 2
Key TCP parameters to configure

6 Performance Evaluation

6.1 Experimental Setting

Referring to a series of surveys (*e.g.*, [30]) on the scale of commercial cloud systems, we emulate a geo-distributed cloud with 10 datacenters, each of which is emulated with a high-end IBM BladeCenter HS23 cluster [31]. Each gateway server is implemented using a mounted blade server with a 16-core Intel Xeon E5-2600 processor and 80GB RAM, where only 4 cores and 4GB RAM are dedicatedly reserved for the gateway server functionality. Hooked to an OpenFlow-enabled HP3500-series core switch [32] (equipped with crossbar switching fabric of up to 153.6 Gbps), each gateway server is connected to 4 other gateway servers via CAT6 Ethernet cables (10 Gbps throughput). To emulate limited bandwidth on dedicated links between datacenters, we configure the rate limits of each core switch on a per-port basis. To make the experimental environment more challenging, the controller module is deployed on a laptop, with a 4-core Intel Core i7 and 16GB RAM, and a maximal 10GB memory allocated to the Java virtual machine.

A number of user robots are emulated on the blade servers to submit data transfer requests to the respective gateway server. In each second, the number of requests generated in each datacenter is randomly selected from the range of $[0, 10]$, with the size of the respective bulk data ranging from 100 MB to 100 GB (real random files are generated and transferred in our system). The destination of the data transfer is randomly selected among the other datacenters. The data transfer requests can be categorized into two types according to their deadlines: (1) urgent transmissions with harsh deadlines (*i.e.*, a deadline randomly generated between 10 seconds and 80 seconds), and (2) less urgent transmissions with relative loose deadlines (*i.e.*, 80 seconds to 640 seconds). We define α as the percentage of the less urgent transmission tasks generated in the system at each time. The chunk size is 100 MB, reasonable for bulk data transfer with sizes up to tens of GB. The length of each time slot is 10 seconds, and the unit bandwidth is hence 80 Mbps. Without loss of generality, the weights of jobs $\mathbb{U}_J$ are assigned values between 1.0 and 10.0, indicating the importance of the jobs. Unless stated otherwise, the evaluation results presented in this section are based on collected logs (around 12GB) after a continuous run of 40 minutes of the system, during which around 12.7 TB transfer traffic is incurred when $\alpha = 10\%$.

We have implemented four chunk scheduling algorithms and compared their performance under the above general setup:

▷ *RES* – the bandwidth-reserving algorithm given in Sec. 4.1.

▷ *INC* – the dynamically-adjusting algorithm given in

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2389842, IEEE Transactions on Cloud Computing

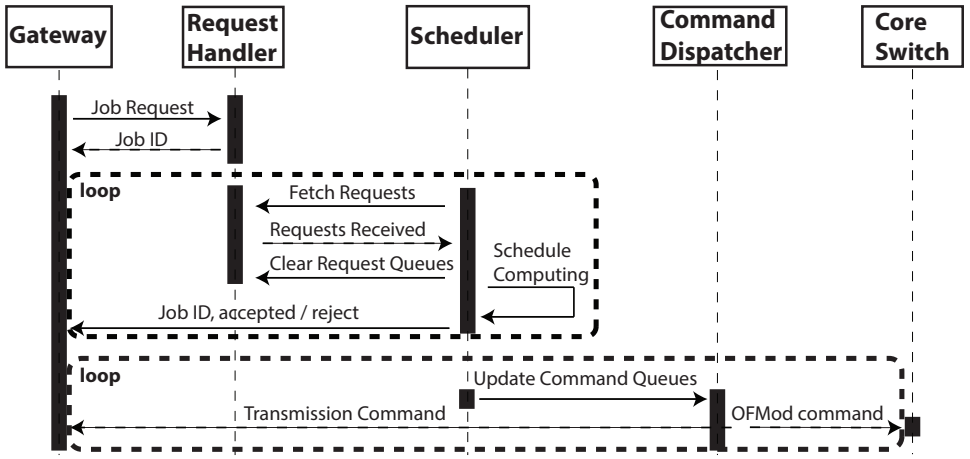


Fig. 4. The control logic between the controller and a gateway server.

Sec. 4.2.

▷ *HEU* – our future-demand-friendly heuristic given in Alg. 1 and Alg. 2 in Sec. 4.3.

▷ *RSF* – a random store-and-forward algorithm, in which when and where a chunk is to be forwarded are randomly decided.

Most existing work on inter-datacenter data transfer either assume known or predictable request arrival patterns [16] (*e.g.*, diurnal patterns) or enforce routing path reservation for continuous network flows [12], while our algorithms do not require so. Hence it is difficult to make direct comparisons with those work. Nevertheless, we will establish special experiment setup in Sec. 6.3 and Sec. 6.5 (to make the algorithms comparable), and compare our algorithms with Postcard [17] and NetStitcher [16], respectively.

6.2 Aggregate Weight and Job Acceptance

Fig. 5 plots the aggregate weight of accepted jobs in the entire system over time. For *RSF*, we treat the jobs that finish transfer before their deadlines as "accepted". We can clearly see that *INC* performs the best, benefiting from its dynamic chunk routing adjustment. *RSF* performs the worst, necessitating a more efficient deadline-aware scheduling algorithm. An interesting observation is that *HEU* outperforms *RES*, though the former is a heuristic and the latter is the optimal solution of the optimal chunk scheduling problem. We believe this is credited to the "future demand friendliness" implemented in *HEU*: *RES* optimally schedules transfer requests received in the latest time slot but does not consider any subsequently arriving requests; *HEU* is conservative in its bandwidth occupancy during its chunk routing scheduling, leaving more available bandwidth for potential, subsequent urgent jobs.

Fig. 7 plots the number of accepted jobs in the system over time. *INC*, *RES* and *HEU* accept a similar number of jobs over time, while *RSF* accepts the least. Comparing to Fig. 5, this result reveals that *INC* is able to admit more important jobs than the other algorithms do, and performs the best in supporting service differentiation among jobs of different levels of practical importance.

We further verify if similar observations hold when the percentage of less urgent jobs, α, varies. Fig. 6 and Fig. 8 show similar trends at different values of α.

6.3 Computation complexity of the algorithms

We examine the computation complexity of different algorithms, in terms of the time the control spends on calculating the chunk routing schedules using each of the algorithms, referred to as *the scheduling delay*. Note that an important requirement is that the scheduling delay of an algorithm should be less than the length of a time slot, *i.e.*, 10 seconds.

In Fig. 9, y axis represents the scheduling delay of the algorithms at the central controller in logarithm scale. We can see that *INC* and *RES* consume more computation time as expected, due to solving an integer optimization problem. However, as we claimed earlier, both algorithms are still efficient enough under our practical settings, with scheduling delays much less than the scheduling interval length (10 seconds). *HEU* incurs similar computation overhead to *RSF*, implying the applicability of *HEU* to systems at larger scales.

Although *Postcard* [17] targets a scenario different from ours (*i.e.*, to minimize the operational cost), we still implement it to investigate the time that the controller needs to run its scheduling algorithm, under a similar setting to that of the other algorithms: we assign an identical cost per traffic unit transferred on each link ($a_{i,j}$ in [17]), and use the same job arrival series with $\alpha = 10\%$. We can see in Fig. 9 that the scheduling delay of *Postcard* is much larger (sometimes more than 15 minutes), due to a much more complicated optimization problem formulated.

Fig. 10 plots the average scheduling delays of the algorithms over time, at different values of α. The scheduling delays by *INC* and *RES* increase as α grows, since both the number of chunks and the lifetime of a job are larger with larger α, which contribute to the complexity of the integer optimization problems. On the other hand, the scheduling delays of *HEU* remain at similar values with the increase of system scale, which reveals the good scalability of our practical heuristic.

Similarly, we also evaluate the maximum number of concurrent job requests that can be handled within the scheduling

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2389842, IEEE Transactions on Cloud Computing

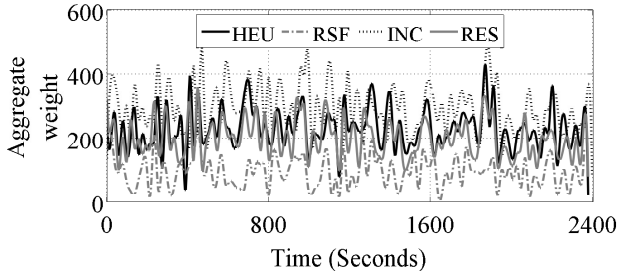


Fig. 5. Aggregate weight of accepted jobs: $\alpha = 10\%$.

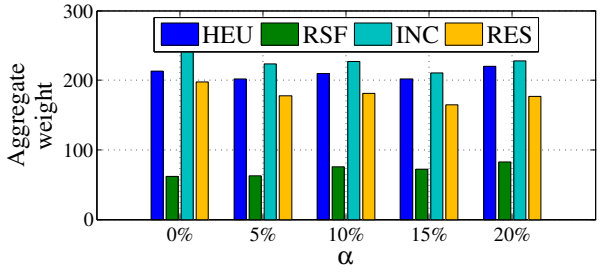


Fig. 6. Aggregate weight of accepted jobs at different percentages of less urgent jobs.

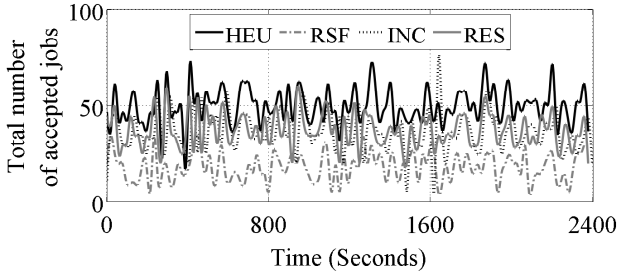


Fig. 7. Total number of accepted jobs: $\alpha = 10\%$.

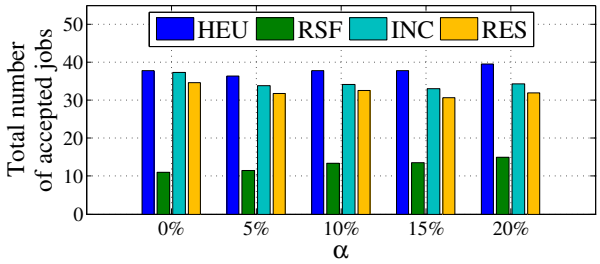


Fig. 8. Total number of accepted jobs at different percentages of less urgent jobs.

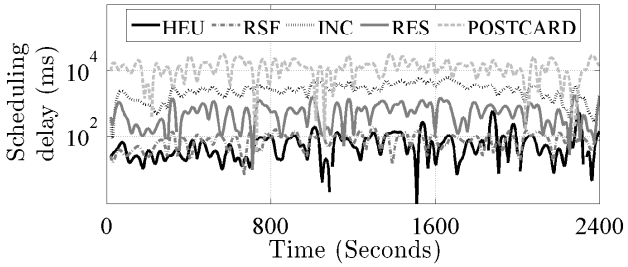


Fig. 9. Scheduling delay: $\alpha = 10\%$.

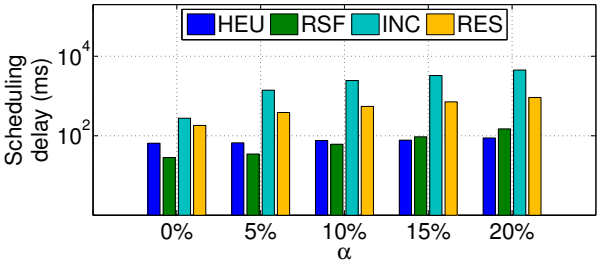


Fig. 10. Scheduling delay at different percentages of less urgent jobs.

delay constraint, *i.e.*, 10 seconds. Fig. 17 plots the average maximum job rate for three different algorithms. We can see that *INC* and *RES* can only accomodate quite a limited job rate, *i.e.*, 40.8 and 182.4 respectively, as opposed to a job rate of 1633.2 achieved by *HEU*. In our experimental setting, each transfer job has an average of 500 chunks (50 GB) to transmit, and thus the job rate can be greatly increased if the job size is reasonably constrained since the complexity of solving the integer problems mainly depends on the number of chunks and the lifetime of the jobs (specified by the deadline).

6.4 Resource Consumption at Gateway Servers

Next we investigate the resource consumption on the gateway servers in Fig. 11 to Fig. 16, due to handling data transmissions and control signaling. The results shown are averages of the corresponding measurements on the 10 gateway servers. The average CPU consumption per gateway server given in Fig. 11 mostly represents the CPU usage for handling control signaling with the controller. We can see that the CPU consumption is similar when each of the four algorithms is deployed, which increases slowly as more transmission tasks accumulate in the system, to be scheduled, over time. The memory consumption in Fig. 13 follows the similar trend. The bandwidth consumption in Fig. 15 represents the average data volumes transmitted on each dedicated link. We see that the peak bandwidth consumption of *RSF* is much higher than that of the other three algorithms, which shows that the "random" behavior of *RSF* leads to high bandwidth consumption with a low aggregate weight achieved, while the other three, especially *INC*, incur less bandwidth consumption while achieving a better aggregate weight (see Fig. 5). Fig. 12, Fig. 14, and Fig. 16 further verify the better performance of our proposed online algorithms (*i.e.*, *INC*, *RES*, *HEU*) as compared to *RSF*, with much slower growth rates of resource consumption with the increase of system load. As compared

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2389842, IEEE Transactions on Cloud Computing

to CPU consumption, memory consumption and bandwidth consumption are relatively dominating on the gateway servers where a large number of TCP connections are established.

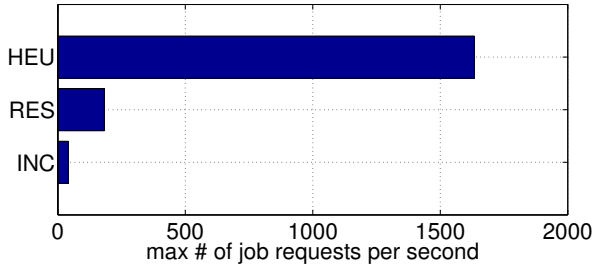


Fig. 17. Maximum job rates accommodated by three different algorithms.

6.5 Link Utilization

As mentioned earlier, the only real-world inter-datacenter bulk data transfer system we are aware of is *NetStitcher* [16]. Different from ours, *NetStitcher* relies on *a priori* knowledge of the traffic pattern on the network links over time, with a scheduling goal of minimizing the transfer time of a given volume of data. It applies the time expansion graph technique which is later adopted by *Postcard* [17], whose computation overhead prevents the scheduling from happening as frequently as ours. Therefore, to make a "fair" comparison, we set the scheduling interval in this set of experiments to 30 minutes, and the chunk size is configured to 18 GB, accordingly. We consider data transfer requests arising from the same source datacenter and destined to the same destination datacenter, with the rest 8 datacenters as potential intermediate store-and-forward nodes. An average of 100 transfer requests (180 GB, identical weights) are issued at the source datacenter every 30 minutes in a 12-hour span, with the deadlines configured to the end of the timespan (used only when running our algorithm). The link bandwidths are configured similarly to our previous experiments. Fig. 18 presents the size of data arriving at the destination by our *INC* algorithm and *NetStitcher* during the 12-hour span. We can see that the size of data transmitted by *NetStitcher* decreases gradually when the links become saturated, whereas *INC* performs better with a stable throughput over time. We believe the reason is as follows: although *NetStitcher* allows inactive replicas of chunks cached at the intermediate nodes to tackle the extreme situations (*e.g.*, "a node can go off-line, or its uplink can be substantially reduced"), only the active replica can be scheduled when the routing paths of a data transfer job need to be recomputed. Differently, *INC* can better exploit the available bandwidth in the system as any intermediate datacenter received a specific chunk can serve as the source afterwards, achieving higher link utilization.

7 Conclusion

This paper presents our efforts to tackle an arising challenge in geo-distributed datacenters, *i.e.*, deadline-aware bulk data transfers. Inspired by the emerging Software Defined Networking (SDN) initiative that is well suited to deployment of an efficient scheduling algorithm with the global view of the network, we propose a reliable and efficient underlying bulk data transfer service in an inter-datacenter network, featuring optimal routing for distinct chunks over time, which can be temporarily stored at intermediate datacenters and forwarded at carefully computed times. For practical application of the optimization framework, we derive three dynamic algorithms, targeting at different levels of optimality and scalability. We also present the design and implementation of our Bulk Data Transfer (BDT) system, based on the Beacon platform and OpenFlow APIs. Experiments with realistic settings verify the practicality of the design and the efficiency of the three algorithms, based on extensive comparisons with schemes in the literature.

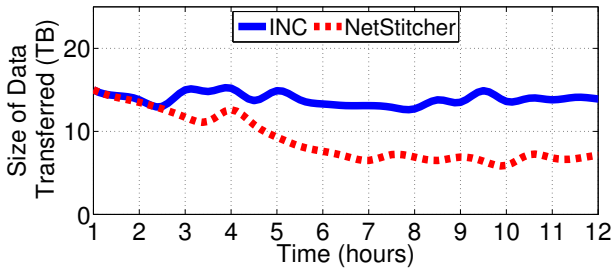


Fig. 18. Size of data transferred over time.

Appendix A
Proof of Theorem 1

Proof: First, the inequality sign in constraint (d) ($\geq$) can be more rigorously specified as the equality sign ($=$), enforced by constraints (a), (b) and (c). For ease of reference, we denote the revised problem as P_{orig}.

Next, we construct a new graph (Fig. 19). Let t_{min} be the minimal value among the earliest time slots for the job transmissions to happen, *i.e.*, $t_{min} = \min_{J \in \mathbb{J}} t_J$. Similarly, we define t_{max} as the latest deadline of all the job transmissions, *i.e.*, $t_{max} = \max_{J \in \mathbb{J}} \{t_J + T_J\}$. Then, the detailed construction of the new graph can be carried out as follows:

- The topology contains $N \times (t_{max} - t_{min} + 1)$ nodes ($N = |\mathbb{N}|$), with nodes at each row i represented as $n_{i,t_{min}}, n_{i,t_{min}+1}, \ldots, n_{i,t_{max}}$.
- Between each pair of nodes $n_{r_1,j}, n_{r_2,j+1}$ ($r_1, r_2 \in [1, N], j \in [t_{min}, t_{max}-1]$) in two neighbouring columns, add a directed link $n_{r_1,j} \rightarrow n_{r_2,j+1}$, with a bandwidth B_{r_1,r_2} if $r_1 \neq r_2$, or $+\infty$ if $r_1 = r_2$.
- The source and destination nodes for each chunk $w \in \mathbb{W}_J$ belonging to $J \in \mathbb{J}$ correspond to nodes n_{S_J,t_J} and n_{D_J,t_J+T_J} in Fig. 19, respectively.

We consider a new optimization problem, denoted as P_{new}, which computes job acceptance and routing paths of each chunks of each job in the newly-constructed graph, such

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2389842, IEEE Transactions on Cloud Computing

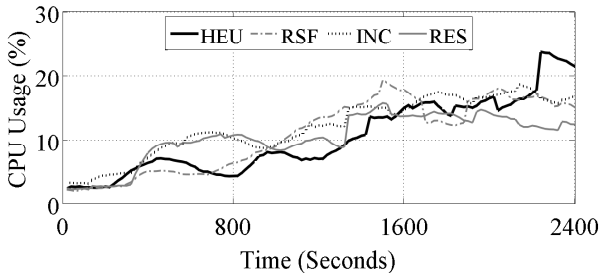


Fig. 11. CPU performance: $\alpha = 10\%$.

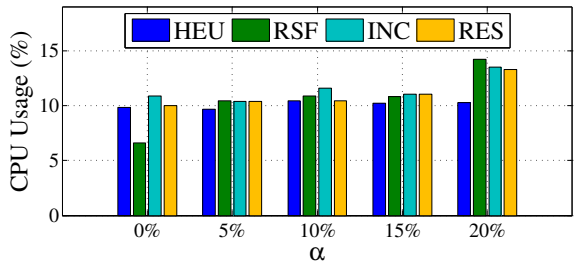


Fig. 12. CPU performance at different percentages of less urgent jobs.

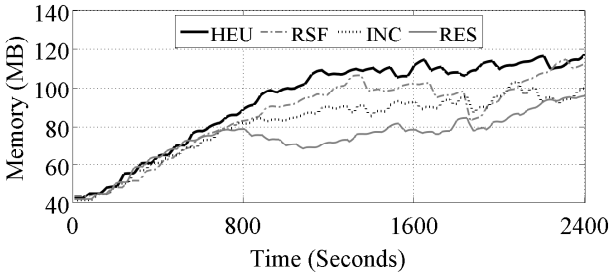


Fig. 13. Memory performance: $\alpha = 10\%$

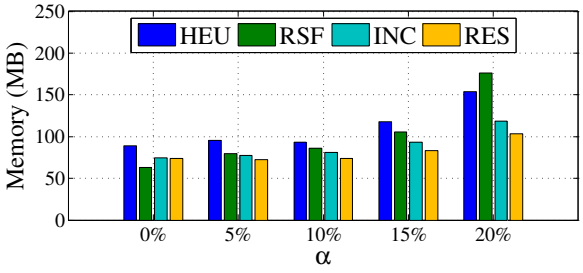


Fig. 14. Memory performance at different percentages of less urgent jobs.

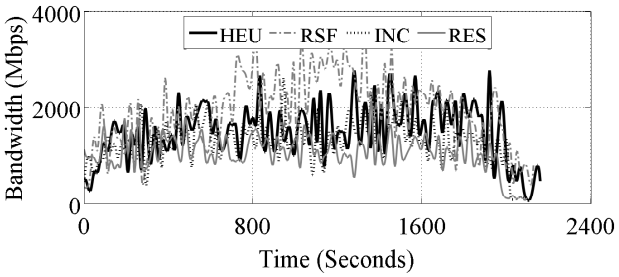


Fig. 15. Bandwidth consumption: $\alpha = 10\%$

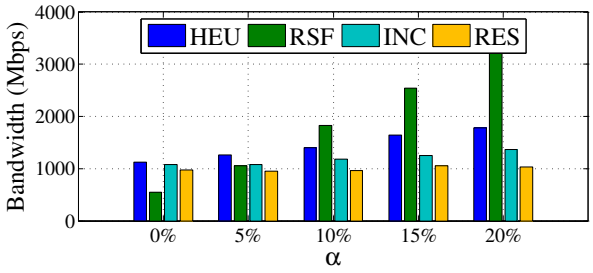


Fig. 16. Bandwidth consumption at different percentages of less urgent jobs.

that the aggregate weight of all accepted jobs is maximized. Especially, if any transmission occurs from r_1 to r_2 at time j, it corresponds to a link $n_{r_1,j} \rightarrow n_{r_2,j+1}$ in Fig. 19. If each job is considered as an individual commodity, problem P_{new} is a maximum multi-commodity flow problem which has been proven to be NP-Complete [33]. It is easy to see that any job $J \in \mathbb{J}$ is accepted in P_{new} if and only if J is accepted in P_{orig}. On the other hand, it takes only polynomial time to reduce problem P_{new} to problem P_{orig} by consolidating all the nodes in a single row as well as the associated links with the following detailed steps:

- For all the nodes in the r-th row, *i.e.*, $n_{r,t_{min}}, n_{r,t_{min}+1}, \ldots, n_{r,t_{max}}$, create a single node n_r.

- For each pair of links between any two neighbouring columns at different rows, *e.g.*, $n_{r_1,j} \rightarrow n_{r_2,j+1}, j \in [t_{min}, t_{max} - 1]$, create a new link between the two newly-created nodes n_{r_1} and n_{r_2} with a bandwidth B_{r_1,r_2}.

- Remove all the links between any two neighbouring columns at the same rows.

- Remove all the original nodes and links.

Hence P_{new} is *polynomial-time reducible* to P_{orig}, which implies, P_{new} is no harder than P_{orig}. Based on the reduction theorem (Lemma 34.8 in [34]), we can derive that P_{orig} is NP hard. The original problem in (1) is hence also NP hard.

□

APPENDIX B
PROOF OF THEOREM 2

Proof: Let's first check the complexity of Alg. 1. Assume N_J represents the number of jobs to be scheduled, max_w

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2389842, IEEE Transactions on Cloud Computing

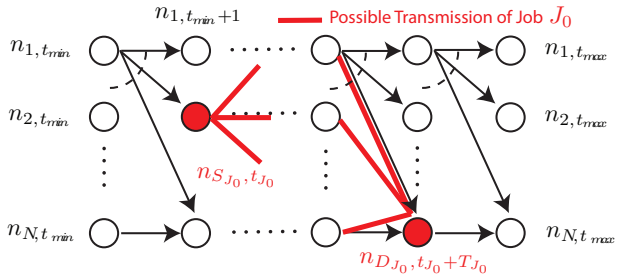


Fig. 19. The newly-constructed network for P_{new}

represents the maximal number of chunks in a single job, T_{max} represents the maximally allowed number of time slots for transmission of a single job, and N represents the number of datacenters. A detailed complexity analysis of individual statements in the algorithm is as follows:

1) The complexity of sorting the requests (line 1) is $O(N_J \times log(N_J))$.

2) The complexity of searching for a shortest path is $O(N^2 \times log(N))$ [35]. Once the path is fixed, it takes at most $O(N \times T_{max})$ steps to verify whether the path is valid, as the path has at most N hops and at most T_{max} time slots need checking for each hop.

3) In Line 6, the transmission schedules, $x^w_{m,n}(t)$, of at most max_w chunks need to be clearly, and the complexity is therefore $O(N^2 \times T_{max} \times max_w)$.

4) The complexities of line 9 and 10 is $O(N \times T_{max})$.

As a result, the overall complexity of Alg. 1 is therefore $O(N_J \times log(N_J) + N_J \times max_w \times (N^2 \times log(N) + N \times T_{max} + N^2 \times T_{max} \times max_w + N \times T_{max}))$, which is polynomial. Similarly, Alg. 2 can be proved to have polynomial-time complexity, with easier efforts. Then, Theorem 2 can be readily proved. □

APPENDIX C
THE PROGRAMMING INTERFACE OF BDT

To-be-scheduled jobs are automatically updated by the framework via Class *BDT_Job_Set*, and the singleton object *CommandQueue* is used to obtain the command queue to save the scheduling commands that will be triggered at the specified time. The example code snippets are listed as follows.

```
//BDT_Schedulable.java
public interface BDT_Schedulable {
   public void schedule();
}
```

```
//BDT_Job_Set.java
/**
 * @param clear whether to remove the
     previously accepted jobs
 * The set contains only the latest jobs if
     clear is true;
 * The set contains both latest jobs and
     previously accepted ones if clear is false
*/
public static void initialize (boolean clear)
    {...}
```

```
//Some concrete scheduler
CommandQueue cmd_queue =
    CommandQueue.getQueue();
cmd_queue.add_command(cmd.pre_fetch_time,
    cmd.time_to_invoke, cmd.cmd_to_switch,
    cmd.src_ip, cmd.dest_ip, cmd.job_id,
    cmd.chunk_index);
```

ACKNOWLEDGEMENT

The research was supported in part by Wedge Network Inc., and grants from RGC under the contract HKU717812 and HKU 718513.

REFERENCES

[1] *Data Center Map*, http://www.datacentermap.com/datacenters.html.
[2] K. K. Ramakrishnan, P. Shenoy, and J. Van der Merwe, "Live Data Center Migration across WANs: A Robust Cooperative Context Aware Approach," in *Proceedings of the 2007 SIGCOMM workshop on Internet network management*, ser. INM '07, New York, NY, USA, 2007, pp. 262–267.
[3] Y. Wu, C. Wu, B. Li, L. Zhang, Z. Li, and F. C. M. Lau, "Scaling Social Media Applications into Geo-Distributed Clouds," in *INFOCOM*, 2012.
[4] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.
[5] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "A Clean Slate 4D Approach to Network Control and Management," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 5, pp. 41–54, 2005.
[6] *SDN*, https://www.opennetworking.org/sdn-resources/sdn-definition.
[7] N. McKeown, T. Anderson, H. Balakrishnan, G. M. Parulkar, L. L. Peterson, J. Rexford, S. Shenker, and J. S. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.
[8] U. Hoelzle, "Openflow@ google," *Open Networking Summit*, 2012.
[9] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, "B4: Experience with a Globally-deployed Software Defined WAN," in *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*. ACM, 2013, pp. 3–14.
[10] S. J. Vaughan-Nichols, "OpenFlow: The Next Generation of the Network?" *Computer*, vol. 44, no. 8, pp. 13–15, 2011.
[11] *Beacon Home*, https://openflow.stanford.edu/display/Beacon/Home.
[12] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better Never than Late: Meeting Deadlines in Datacenter Networks," in *Proceedings of the ACM SIGCOMM*, New York, NY, USA, 2011, pp. 50–61.
[13] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-aware Datacenter TCP (d2tcp)," in *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*. ACM, 2012, pp. 115–126.
[14] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pFabric: Minimal Near-optimal Datacenter Transport," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: ACM, 2013, pp. 435–446.
[15] Y. Chen, S. Jain, V. K. Adhikari, Z.-L. Zhang, and K. Xu, "A First Look at Inter-data Center Traffic Characteristics via Yahoo! Datasets." in *INFOCOM*, 2011.
[16] N. Laoutaris, M. Sirivianos, X. Yang, and P. Rodriguez, "Inter-datacenter Bulk Transfers with Netstitcher," in *Proceedings of the ACM SIGCOMM 2011 conference*, New York, NY, USA, 2011, pp. 74–85.
[17] Y. Feng, B. Li, and B. Li, "Postcard: Minimizing Costs on Inter-Datacenter Traffic with Store-and-Forward," in *Proceedings of the 2012 32nd International Conference on Distributed Computing Systems Workshops*, ser. ICDCSW '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 43–50.
[18] E. Khler, K. Langkau, and M. Skutella, "Time-Expanded Graphs for Flow-Dependent Transit Times," in *Proc. 10th Annual European Symposium on Algorithms*. Springer, 2002, pp. 599–611.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2015.2389842, IEEE Transactions on Cloud Computing

[19] B. B. Chen and P. V.-B. Primet, "Scheduling Deadline-constrained Bulk Data Transfers to Minimize Network Congestion," in *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, 2007, pp. 410–417.

[20] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "ElasticTree: Saving Energy in Data Center Networks." in *NSDI*, vol. 3, 2010, pp. 19–21.

[21] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving High Utilization with Software-Driven WAN," in *Proceedings of the ACM SIGCOMM*, Hong Kong, China, 2013.

[22] M. A. Rodriguez and R. Buyya, "Deadline Based Resource Provisioning and Scheduling Algorithm for Scientific Workflows on Clouds," *IEEE Transactions on Cloud Computing*, vol. 2, no. 2, April 2014.

[23] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling Flow Management for High-performance Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 254–265, Aug. 2011.

[24] H. Liu, C. F. Lam, and C. Johnson, "Scaling Optical Interconnects in Datacenter Networks Opportunities and Challenges for wdm," in *High Performance Interconnects (HOTI), 2010 IEEE 18th Annual Symposium on*. IEEE, 2010, pp. 113–116.

[25] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Englewood Cliffs, NJ: Prentice Hall, 1993.

[26] *CPLEX Optimizer*, http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/.

[27] *OSGi Alliance*, http://www.osgi.org/Main/HomePage.

[28] *Equinox*, http://eclipse.org/equinox/.

[29] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A Network Virtualization Layer," *OpenFlow Switch Consortium, Tech. Rep*, 2009.

[30] *Google Data Center FAQ*, http://www.datacenterknowledge.com/archives/2012/05/15/google-data-center-faq/.

[31] *IBM BladeCenter HS23*, http://http://www-03.ibm.com/systems/bladecenter/hardware/servers/hs23/specs.html.

[32] *HP 3500 and 3500 yl Switch Series*, http://h17007.www1.hp.com/us/en/networking/products/switches/HP_3500_and_3500_yl_Switch_Series/index.aspx#.Uezkh2TIydM.

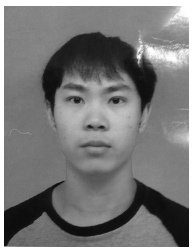
[33] S. Even, A. Itai, and A. Shamir, "On the Complexity of Time Table and Multi-commodity Flow Problems," in *Proceedings of the 16th Annual Symposium on Foundations of Computer Science*. Washington, DC, USA: IEEE Computer Society, 1975, pp. 184–193.

[34] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*, 2nd ed. McGraw-Hill Higher Education, 2001.

[35] R. Bellman, "On a Routing Problem," *Quarterly of Applied Mathematics*, vol. 16, pp. 87–90, 1958.

Yu Wu received the B.E. and M.E. degrees in computer science and technology from Tsinghua University, Beijing, China, in 2006 and 2009, respectively, and the Ph.D. degree in computer science from the Univeristy of Hong Kong, Hong Kong, in 2013.

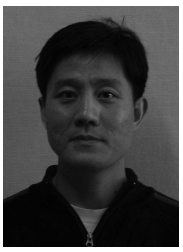
He is currently a Postdoctoral Scholar with the Department of Electrical, Computer, and Energy Engineering, Arizona State university, Tempe, AZ, USA. His research interests include cloud computing, mobile computing, network virtualization and content-centric network.



Zhizhong Zhang received his B.Sc. degree in 2011 from the Department of Computer Science, Sun Yat-sen University, China. He is currently a Ph.D. student in the Department of Computer Science, The University of Hong Kong, Hong Kong. His research interests include networks and systems.

Chuan Wu received her B.E. and M.E. degrees in 2000 and 2002 from Department of Computer Science and Technology, Tsinghua University, China, and her Ph.D. degree in 2008 from the Department of Electrical and Computer Engineering, University of Toronto, Canada. She is currently an associate professor in the Department of Computer Science, The University of Hong Kong, China. Her research interests include cloud computing, peer-to-peer networks and online/mobile social network. She is a member of IEEE and ACM.

Chuanxiong Guo (M'03) received the Ph.D. degree in communications and information systems from Nanjing Institute of Communications Engineering, Nanjing, China, in 2000. He is a Senior Researcher with the Wireless and Networking Group, Microsoft Research Asia, Beijing, China. His research interests include network systems design and analysis, data-center networking, data-centric networking, network security, networking support for operating systems, and cloud computing.



Zongpeng Li received his B.E. degree in Computer Science and Technology from Tsinghua University (Beijing) in 1999, his M.S. degree in Computer Science from University of Toronto in 2001, and his Ph.D. degree in Electrical and Computer Engineering from University of Toronto in 2005. Since August 2005, he has been with the Department of Computer Science in the University of Calgary. Zongpeng's research interests are in computer networks and network coding.

Francis C.M. Lau (SM, IEEE) received his Ph.D. in Computer Science from the University of Waterloo, Canada. He has been a faculty member in the Department of Computer Science, The University of Hong Kong, since 1987, where he served as the department head from 2000 to 2006. His research interests include networking, parallel and distributed computing, algorithms, and application of computing to art. He is the editor-in-chief of the Journal of Interconnection Networks.