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.2015.2487964, IEEE Transactions on Cloud Computing

Inter-Datacenter Scheduling of Large Data Flows

Reuven Cohen Gleb Polevoy
Department of Computer Science
Technion–Israel Institute of Technology
Haifa 32000, Israel

***Abstract*—Inter-datacenter transfers of non-interactive but timely large flows over a private (managed) network is an important problem faced by many cloud service providers. The considered flows are non-interactive because they do not explicitly target the end users. However, most of them must be performed on a timely basis and are associated with a deadline. We propose to schedule these flows by a centralized controller, which determines when to transmit each flow and which path to use. Two scheduling models are presented in this paper. In the first, the controller also determines the rate of each flow, while in the second bandwidth is assigned by the network according to the TCP rules. We develop scheduling algorithms for both models and compare their complexity and performance.**

I. Introduction

Cloud services continue to grow rapidly, and major cloud service providers connect many geographically dispersed datacenters to form geo-distributed cloud networks [1], [13], [16]. Astronomical amounts of data are transferred over this network of datacenters for a myriad of reasons, including:

- Data replication to ensure disaster recovery.
- Data relocation in order to shut off some sites during off-peak hours to save resources or for maintenance.
- Replication and/or relocation of data in order to bring it closer to customers in different geographic locations.

Such inter-datacenter transfers are non-interactive, because they do not explicitly target the end users. Nevertheless, they are performed on a timely basis and each is associated with a rough target deadline. We address the problem of timely transfer of these large non-interactive data flows between datacenters, not over the public Internet but over a well-managed private network.

The flows that are handled by the considered scheduler fulfill the following requirements: (a) their starting time can be controlled; (b) their bandwidth demand is very large and known (i.e., we schedule elephants, not mice); (c) each of them has a target deadline. Other flows are not handled by our scheduling algorithms.

We assume that the control logic has the following information for each to-be-scheduled data flow: its size, its source-destination pair, its time of release and required delivery time, and a utility function that indicates the "profit to the system" for delivering the flow on time. In addition, the control logic knows the set of paths over which each flow can be routed. The controller needs to schedule the transmission of each flow such that the total utility is maximized and the network bandwidth resources are not exceeded.

Our network model is similar in many aspects to the one considered in [12]. The bandwidth of the considered private network is divided into two classes: best-efforts and guaranteed services. The first class is for spontaneous best-effort connections that are not targeted by the controller. The other class is for the considered scheduled flows. Thus, the scheduled flows do not encounter congestion due to the non-scheduled flows.

We believe that the main application of the considered model is in the offline context, when all the flows are given to the controller before the scheduling algorithm is invoked. However, we will also present online algorithms, which can be used if the controller receives the flows one by one when they are first ready for scheduling. In both cases, the decision when to admit each flow, which path to use (when multiple paths are given for each flow), and how much bandwidth to allocate to each flow, if bandwidth is not assigned by the network, is made solely by the controller.

As an application example, consider the Globally-Deployed Software Defined WAN of Google, called B4. The characteristics of this network, as described by [13], make it perfectly suited to the model and problem considered in this paper: a private WAN is used for synchronizing large data sets across sites; the network operator can enforce relative application priorities and control bursts at the network edge, rather than through overprovisioning or complex functionality in the WAN; capacity targets and growth rate led to unsustainable cost projections, which render traditional WAN overprovisioning approaches impractical.

When addressing the problem of inter-datacenter flow scheduling, it is crucial to determine how congestion control will be performed. We believe there are two possible answers:

- **Network Assigned Bandwidth (NAB).** In this model, all the scheduled flows are transmitted using TCP. The bandwidth assigned to every flow is allocated by the network according to the TCP rules. The controller needs to determine the starting time and the routing path (if flows are not limited to using their shortest paths) for each flow, and to estimate when the transmission of every flow will finish.
- **Controller Assigned Bandwidth (CAB).** In this model, the bandwidth assigned to every flow is allocated by the controller whose role is to prevent congestion in the network and to ensure that flows are delivered on time. Thus, the controller determines the starting time, the allocated rate, and the routing path (if flows are not

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.2015.2487964, IEEE Transactions on Cloud Computing

routed only over their shortest paths) for each scheduled flow. In this model, end-to-end reliability is decoupled from congestion control. The protocol used for end-to-end transmission is referred to as $\mathbf{TCP}^{-}$. This protocol is similar to TCP, but it does not implement TCP congestion control, because the transmission rates of the senders are determined in advance by the controller. The $\mathbf{TCP}^{-}$ protocol is not addressed in this paper, but its implementation is relatively simple.

Scheduling-based congestion control is not a new idea. For example, see [2], [10] and references therein. $\mathbf{TCP}^{-}$ can be viewed as the opposite of the Datagram Congestion Control Protocol (DCCP) [15], because $\mathbf{TCP}^{-}$ provides reliability without congestion control whereas DCCP provides congestion control without reliability. There were some efforts in the past to define $\mathbf{TCP}^{-}$ under the name "reliable UDP[1]."

As already indicated, in our problem definition, the scheduler chooses a routing path for each flow, but it does not compute these paths. In a network that supports only shortest path routing, this set will contain only shortest paths, while in a network that supports traffic engineering, this set may contain other paths as well. The problem of determining the set of routing paths to be given to the scheduler as an input is orthogonal to the scheduling algorithms proposed in this paper.

NAB-scheduling is usually impractical, because every new flow changes the termination time of existing flows, and because it is very difficult for the controller to estimate the bandwidth to be acquired by each flow even if the algorithms proposed in [11], [20] are used. However, studying the NAB model allows us to better understand the potential trade-off between NAB's performance and CAB's practicality.

Our goal in this paper is three-fold. First, we want to develop efficient algorithms for solving the scheduling problem for both NAB and CAB. Our second goal is to compare the complexity and performance of CAB-scheduling to that of NAB-scheduling. The purpose of this comparison is to see if the added complexity of NAB-scheduling is translated into significant performance gain. Finally, it is intuitive that flow elasticity allows the scheduler to perform better. However, the strength of the correlation between elasticity and performance is not clear. Our third goal is thus to study this correlation both for CAB- and NAB-scheduling.

The rest of the paper is organized as follows. In Section II we present related work. In Section III we define the CAB-scheduling problem, prove that it is NP-hard, and present an approximation algorithm for solving it. In Section IV we define the NAB-scheduling problem, which is also NP-hard, and present two classes of algorithms for solving it. In Section V we compare CAB and NAB from a theoretical perspective, and in Section VI we compare their practical performance. Finally, we conclude in Section VII.

[1]While there is no document that can be cited, there was an IETF draft that expired in 2000.

II. Related Work

Inter-datacenter networking has been attracting a lot of attention lately. The need for a scheduling logic that determines when to transmit each data flow is implicitly or explicitly indicated in [3], [7], [9], [12], [13], [16], [19].

The two most relevant papers are probably [3] and [12]. In [12], the authors study a model similar to ours in that it uses (1) global coordination of the sending rates of services and (2) central allocation of data paths. The goal of their proposed system is high efficiency while meeting certain policy goals. They consider three priority classes: active, elastic, and background. The transmission of the active and elastic flows is scheduled by the controller. The focus of [12] is on computing bandwidth allocations, updating forwarding state, handling failures and prototype implementation.

In [3], the authors propose a scheduling model for datacenter routing. Their paper focuses on finding an efficient path inside the datacenter for each flow in order to maximize the throughput. The controller collects flow information from constituent switches, computes non-conflicting paths for flows, and instructs switches to re-route traffic accordingly. The goal is to maximize aggregate network utilization with minimal scheduler overhead or impact on active flows.

In [16], the authors show how to rescue unutilized bandwidth across multiple datacenters and backbone networks and use it for non-real-time applications, such as backups, propagation of bulky updates, and migration of data. While the problem solved in [16] is different from the problem we consider, in both cases the scheduler takes advantage of the fact that the transmission of a data flow can often be postponed to non-peak hours.

In [7], the authors study and analyze inter-datacenter traffic characteristics using the anonymized NetFlow datasets collected at the border routers of five major Yahoo! datacenters. Their study reveals that Yahoo! uses a hierarchical deployment of datacenters, with US backbone datacenters and several satellite datacenters distributed in other countries. In [13], the authors present the design, implementation, and evaluation of a private WAN connecting Google's datacenters across the planet. This network is shown to have a number of unique characteristics, such as massive bandwidth requirements deployed to a modest number of sites, elastic traffic demand that seeks to maximize average bandwidth, and full control over the edge servers and network. These characteristics led to a Software Defined Networking architecture using OpenFlow to control relatively simple switches built from merchant silicon. B4 phase 3 employs centralized traffic engineering, and optimized routing based on 7 application-level priorities. In addition, an external copy scheduler interacts with the OpenFlow controller to implement deadline scheduling for large data copies[2].

In [9], the authors seek to minimize operational costs on inter-datacenter traffic with store-and-forward at intermediate

[2]In http://opennetsummit.org/archives/apr12/hoelzle-tue-openflow.pdf, these properties are mentioned with no further details.

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.2015.2487964, IEEE Transactions on Cloud Computing

nodes, by choosing routing paths, flow assignments, and scheduling strategies for each source-destination traffic pair. In [19], the authors propose a globally reconfigurable intelligent photonic network that offers bandwidth-on-demand service in the core network for efficient inter-datacenter communication. The proposed solution is motivated by the variability in traffic demands for communication across datacenters: non-interactive bulk data transfers between datacenters have different patterns than interactive end-user driven traffic.

In [17], the authors propose and evaluate a framework for optimization-based request distribution, which enables services to manage their energy consumption and costs. In [18], the authors indicate the importance of cross-datacenter macro-resource management, and propose a management layer to make coordination decisions across applications and across physical facilities.

III. The Controller Assigned Bandwidth (CAB) Scheduling Problem

A. Problem Formulation and Hardness

We start by defining the CAB-scheduling problem, where bandwidth is allocated to flows by the scheduler. Each flow is served by a $\mathbf{TCP}^-$ connection, which uses TCP-like rules for guaranteeing end-to-end reliability, but does not perform congestion control.

Problem 1 (CAB)

Input: A communication network, represented by a directed graph. Each directed edge e has a transmission capacity $c(e) > 0$. There is a set of data flows waiting to be transmitted, each attributed with:

1) A pair of source/destination nodes and a positive size (number of bytes).
2) A finite set of possible transmission windows. Each window is associated with: (a) a description of a routing path from the source to the destination; this can be the default (sometimes known as "shortest") path, or any other path; (b) a release time, which indicates when the flow is ready for transmission; (c) a deadline, which indicates the time when the flow must be completely delivered in order to be useful; (d) a profit/utility, which is explained below.
3) A set of possible bandwidth rates that can be allocated to this flow (e.g., 1 Mbps, 10 Mbps, 100 Mbps).

Objective: Find a maximum profit, feasible schedule with at most one *scheduling instance* for every flow. Each scheduling instance of a flow is characterized by: (a) a routing path and a starting time from one of the possible transmission windows of this flow, say w; (b) a fixed rate from the set of possible rates. The rate is allocated to the flow along the routing path associated with w until the flow finishes. The time it takes for an instance to finish is equal to the flow size divided by allocated bandwidth. A schedule is said to be feasible if the total bandwidth allocated to all instances at every time t on every link e is $\leq c(e)$. If a flow starts after the release time of a window w and finishes before the deadline of the same window w, it acquires the profit/utility corresponding to that window.

By using a generic utility/profit function, we allow the controller to decide which parameter has to be optimized. For example, by assigning the same profit to each flow, our optimization function would maximize the number of flows delivered on time. If the assigned profit is proportional to the size of each flow, the optimization function would maximize the amount of data delivered on time. It is also possible to assign to each flow the priority of the originating user or application; for example, a flow whose aim is to back up a database will be assigned a lower profit than one whose aim is to migrate a virtual machine. Moreover, since the profit is associated with a transmission window, and not with a flow, the same flow may have different priorities for different possible transmission windows. As an example, one can assign to a flow that is routed over one of its shortest paths a bigger profit than the profit assigned to the same flow when it is routed over a longer path.

CAB is related to the Throughput Maximization Problem (TMP) [4], defined as follows. Let $\{\mathcal{A}_i : i = 1, \ldots, n\}$ be a set of activities, where each activity consists of a set of instances $\mathcal{I}$. Every instance has a profit $p(\mathcal{I})$, a width $w(\mathcal{I})$, a start time $b(\mathcal{I})$, and a finish time $f(\mathcal{I})$. For every $\mathcal{I} \in \mathcal{A}_i$, there exists a Boolean $x_{\mathcal{I}}$. The problem is

$$\max \sum_{\text{All } \mathcal{I}} p(\mathcal{I}) x_{\mathcal{I}},$$

subject to:

$$\sum_{\mathcal{I}: b(\mathcal{I}) \leq t < f(\mathcal{I})} w(\mathcal{I}) x_{\mathcal{I}} \leq 1 \text{ for every time } t$$

and

$$\sum_{\mathcal{I} \in \mathcal{A}_i} x_{\mathcal{I}} \leq 1 \text{ for each activity } \mathcal{A}_i.$$

CAB generalizes TMP in two important ways:

1) While in TMP the bandwidth allocated to each flow is determined in advance and is given as a part of the input, in CAB it is determined by the controller.
2) While in TMP all the flows run on one common link, in CAB they run on an arbitrary network, and for each flow a routing path is chosen.

CAB is not only NP-hard, but it also does not admit a PTAS (polynomial-time approximation scheme). Namely, there exists an $\epsilon > 0$ such that there is no polynomial time approximation algorithm whose approximation ratio is ϵ to this problem. We prove this by showing that CAB is MAX SNP-hard, which implies that it does not admit a PTAS if P $\neq$ NP.

Theorem 1. *CAB is MAX SNP-hard even in the special case where the network has a single edge of unit capacity, all the flows have the same size, there are 2 windows per flow that do not intersect (in time), and the profit of each window is 1.*

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.2015.2487964, IEEE Transactions on Cloud Computing

Proof: The unweighted job interval scheduling problem with k intervals per job is defined as follows. Its input is a finite set of jobs, each of which is a k-tuple of real intervals. Its objective is to select a non-intersecting subset of intervals, such that at most one interval is selected for each job and the total number of selected intervals is maximized. This problem is MAX SNP-hard even if each job has only 2 intervals, the intervals of the same job are non-intersecting, and all the intervals are of the same length L [21]. This problem is a special case of CAB, in that it maps each job to a flow of size L, converts each interval to a transmission window from the beginning of the interval to its end, and assigns a profit 1 to each window. ■

B. An Approximation Algorithm for CAB

Although there is no PTAS for CAB, we are still able to develop a polynomial time approximation algorithm with a guaranteed approximation ratio. To this end, consider a flow to which we have already allocated some bandwidth. This implies that the time it takes to finish the flow is fixed, and it is equal to the flow size divided by the allocated bandwidth. If we are able to determine in advance the allocated bandwidth to each flow, we know the transmission time ("length") of this flow and we only need to decide when to start it.

Given a transmission window of a flow and the bandwidth allocated to it, one can easily determine the feasible sub-window for starting this flow. Consider a flow of β bytes to which a bandwidth of B bytes/sec is allocated. Let $[t_s, t_f]$ be one of the windows during which this flow can start and finish. The time it takes for this flow to finish is β/B sec. Thus, assuming that $\beta/B \leq t_f - t_s$, the flow should start during $[t_s, t_f - \beta/B]$. This interval is now referred to as *the feasible sub-window for starting the flow.*

In [4], it is shown how to use an approximation algorithm for TMP with a finite set of possible instances per flow in order to approximate the original version with continuous feasible starting sub-windows (each containing an infinite set of possible instances). This technique works for CAB as well. Therefore, we start with a finite set of possible instances given explicitly for every flow, a problem referred to as discrete CAB and solved by our Algorithm 1. Then, we present Algorithm 2, which extends Algorithm 1 and solves the general (continuous) CAB problem.

Throughout the paper we assume that each input instance is feasible if taken alone. To ensure the validity of this assumption, all the non-feasible instances are discarded before the algorithm commences.

We start by extending the TMP algorithm proposed in [4] to solve the discrete version of CAB. Both problems consider a finite number of possible instances for every flow. The algorithm from [4] uses the local-ratio technique [5], [6], outlined as follows. Let F be a set of constraints and let $p(), p_1(), p_2()$ be profit functions such that $p() = p_1() + p_2()$. Then, if x is an α-approximate solution with respect to $(F, p_1())$ and with respect to $(F, p_2())$, it is also an α-approximate solution with respect to $(F, p())$. The proof in [5] is very simple. Let x^*, x_1^* and x_2^* be optimal solutions for $(F, p())$, $(F, p_1())$, and $(F, p_2())$ respectively. Then, $p(x) = p_1(x) + p_2(x) \geq \alpha \cdot p_1(x_1^*) + \alpha \cdot p_2(x_2^*) \geq \alpha \cdot (p_1(x^*) + p_2(x^*)) = \alpha \cdot p(x^*)$.

Algorithm 1. *(solve the discrete CAB problem)*
Input: a set $\mathcal{R}$ of all possible instances for all flows, and a profit function p.

1) *Delete from $\mathcal{R}$ all the instances with a non-positive profit.*
2) *If $\mathcal{R} = \emptyset$, return an empty schedule.*
3) *Let $\tilde{i}$ be an instance with the earliest ending time in $\mathcal{R}$. Using the value of $p(\tilde{i})$, split the profit function p into p_1 and $p_2 = p - p_1$.*
4) *Run recursively on the input $(\mathcal{R}, p_2)$. Let S' be the returned schedule.*
5) *If $S' \cup \{\tilde{i}\}$ is a feasible schedule, return $S = S' \cup \{\tilde{i}\}$. Otherwise, return $S = S'$.*

The most important part of Algorithm 1 is the development of a p_1 function in step (3). To this end, define a feasible schedule S to be i-maximal if either $i \in S$, or $i \notin S$ and $S \cup \{i\}$ is infeasible. Function p_1 is chosen such that $p_2(\tilde{i}) = 0$, and for a certain $r > 0$ every $\tilde{i}$-maximal schedule is an r-approximation with respect to p_1. Using the local ratio technique, as described above, such a selection of p_1 guarantees that Algorithm 1 will return an r-approximation. The condition $p_2(\tilde{i}) = 0$ ensures that the algorithm terminates after $O(|\{\text{all the instances}\}|)$ recursive invocations, since it discards at least 1 instance during every recursive call.

Let $w(i)$ be the normalized bandwidth assigned to instance i, i.e., the assigned bandwidth divided by the maximum link bandwidth in the network. To define the p_1 function for the discrete CAB problem, we distinguish between the case where every instance $i \in \mathcal{R}$ is **"wide,"** i.e., $\forall i \in \mathcal{R} : w(i) > \rho$ for some fixed $\rho \in (0, 0.5]$, and the case where every instance is **"narrow,"** i.e., $\forall i \in \mathcal{R} : w(i) \leq \rho$. We split the set of instances into a subset of narrow instances and a subset of wide instances, and invoke Algorithm 1 with a different p_1 function on each subset. Then, we choose the best solution returned by the two independent executions of Algorithm 1 as a solution for the general case.

Let $\mathcal{A}(i)$ denote the set of all instances of the flow to which instance i belongs; these instances can never be taken together with i. Let $\mathcal{I}(i)$ be the set of all instances not in $\mathcal{A}(i)$ that intersect i in time and have at least one common edge with i. Intuitively, we can see that these are the instances that have capacity conflict with i. We view an instance of a flow also as the set of edges along its route. Recall that our problem's definition allows different instances of the same flow to use different routes.

Let E be the edge set of the network. The capacity of an edge $e \in E$ is denoted $c(e)$. Again, we normalize the capacities, such that $\forall e : c(e) \in (0, 1]$. For every $X \subseteq E$, let $c(X) = \min\{c(e) | e \in X\}$ while $c(\emptyset) \triangleq \infty$. Additionally, let $\mathrm{c_{min}}(i) = \min\{c(i \cap i') | i' \in \mathcal{I}(i)\}$, where $\min(\emptyset) \triangleq \infty$, and $\mathrm{c_{max}}(i) = \max\{c(i \cap i') | i' \in \mathcal{I}(i)\}$, where $\max(\emptyset) \triangleq 0$.

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.2015.2487964, IEEE Transactions on Cloud Computing

Finally, define $\mathrm{intersct}(i)$ as the number of edges of an instance i that also belong to an instance in $\mathcal{I}(i)$.

As explained above, to obtain a local ratio approximation, the weight function p_1 should fulfill the condition that for a certain $r > 0$ every $\tilde{i}$-maximal schedule is an r-approximation with respect to p_1. To this end, we have to give every instance a profit that will be close to the profit it would acquire in an arbitrary $\tilde{i}$-maximal schedule. For example, every instance in a flow to which $\tilde{i}$ belongs has the same profit, since there is at most one such instance in any feasible solution. The instances that intersect $\tilde{i}$ have profit that is inversely proportional to the bottleneck capacity, since this is close to the number of such instances in an $\tilde{i}$-maximal schedule. When dealing with narrow instances, we also consider the weight of an instance in their profit, while for the wide instances the weight is taken as the minimum relevant capacity. This profit definition ensures that every $\tilde{i}$-maximal schedule is a good approximation of the optimum, with respect to p_1. The proof of Theorem 2 below formalizes this intuition.

We now define the p_1 function for discrete CAB. The definition is based on $p(\tilde{i})$, where $\tilde{i}$ is the instance chosen in step (3) of Algorithm 1:

- When the algorithm runs on wide instances, for each instance i

$$p_1(i) = p(\tilde{i}) \cdot \begin{cases} 1 & i \in \mathcal{A}(\tilde{i}) \\ \frac{1}{c(i \cap \tilde{i})} & i \in \mathcal{I}(\tilde{i}) \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

- When the algorithm runs on narrow instances, for each instance i

$$p_1(i) = p(\tilde{i}) \cdot \begin{cases} 1 & i \in \mathcal{A}(\tilde{i}) \\ \frac{\frac{1}{c(\tilde{i})-\rho} w(i)}{c(i \cap \tilde{i})} & i \in \mathcal{I}(\tilde{i}) \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

Let $\mathrm{c_{min}} \triangleq \min_i \{\mathrm{c_{min}}(i)\}$ and $\mathrm{intersct} \triangleq \max_i \{\mathrm{intersct}(i)\}$. Then,

Theorem 2. *Let $\rho < c(E)$; i.e., ρ is smaller than the minimum link capacity of the network.*
(a) *When Algorithm 1 runs only on wide instances, the obtained profit is at least*

$$\frac{1}{1 + \frac{(\lceil \frac{1}{\rho} \rceil - 1)\,\mathrm{intersct}}{\mathrm{c_{min}}}} = \frac{\mathrm{c_{min}}}{\mathrm{c_{min}} + (\lceil \frac{1}{\rho} \rceil - 1)\,\mathrm{intersct}}$$

of the optimum profit.
(b) *When Algorithm 1 runs only on narrow instances, the obtained profit is at least*

$$\frac{1}{1 + \frac{\mathrm{intersct}}{(c(E)-\rho)\,\mathrm{c_{min}}}} = \frac{\mathrm{c_{min}}}{\mathrm{c_{min}} + \frac{1}{c(E)-\rho}\,\mathrm{intersct}}$$

of the optimum profit.
(c) *When Algorithm 1 runs only on narrow instances and then only on the wide instances, and the schedule with the larger profit is chosen, the approximation ratio of the algorithm is*

$$\frac{1}{2 + \frac{(\frac{4}{c(E)})\,\mathrm{intersct}}{\mathrm{c_{min}}}} = \frac{\mathrm{c_{min}}}{2\,\mathrm{c_{min}} + (\frac{4}{c(E)})\,\mathrm{intersct}}.$$

The proof can be found in the Appendix.

Next, we extend Algorithm 1 to address the general (continuous) CAB. The generalization applies Algorithm 1 on feasible starting sub-windows, rather than on individual instances. Throughout the profit decomposition, Algorithm 2 maintains the invariant that all the instances represented by a feasible starting sub-window have the same profit. This invariant requires the algorithm to split some of the feasible starting sub-windows. To guarantee polynomial running time, the algorithm deletes feasible starting sub-windows whose profit becomes smaller than an ϵ-fraction of their original one. This results in a loss of up to an ϵ factor in the approximation ratio.

Before invoking Algorithm 2, we need to convert all the windows into a set of feasible starting sub-windows for starting each flow. This is done by considering each possible bandwidth, and creating a feasible starting sub-window for each bandwidth and each possible window $[t_s, t_f]$.

Algorithm 2. *(A continuous version of Algorithm 1 for the continuous CAB)*
Input: a set $\mathcal{R}$ of all feasible sub-windows for starting each flow, a profit function p, and the desired approximation ratio ϵ.

1) *Delete from $\mathcal{R}$ all the feasible starting sub-windows with a non-positive profit.*
2) *If $\mathcal{R} = \emptyset$, return an empty schedule.*
3) *Find the instance with the earliest ending time in $\mathcal{R}$; let this instance be $\tilde{i}$.*
   If $p(\tilde{i}) < \epsilon \cdot [$ the original profit of $\tilde{i}]$, delete the feasible starting sub-window that contains $\tilde{i}$ and go to step (2). Else, using the value of $p(\tilde{i})$, determine the profit function p_1 as discussed later, and let $p_2 = p - p_1$. To guarantee that all the instances of the same feasible starting sub-window have the same profit (relative to p_2) even after updating $\mathcal{I}(\tilde{i})$, split the feasible starting sub-window into two parts: one that contains the instances that start before $\tilde{i}$ ends, and another that contains the rest of the instances.
4) *Run recursively on the input $(\mathcal{R}, p_2, \epsilon)$. Let S' be the returned schedule.*
5) *If $S' \cup \{\tilde{i}\}$ is a feasible schedule, return $S = S' \cup \{\tilde{i}\}$. Otherwise, return $S = S'$.*

The removal of the feasible starting sub-window that contains $\tilde{i}$, whose profit is smaller than ϵ times the original profit, may reduce the approximation factor by ϵ, as indicated in the following theorem.

Theorem 3. *Let n be the total number of feasible sub-windows for starting a flow, each associated with a starting time and a bandwidth. For any $\epsilon > 0$, Algorithm 2 guarantees a*

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.2015.2487964, IEEE Transactions on Cloud Computing

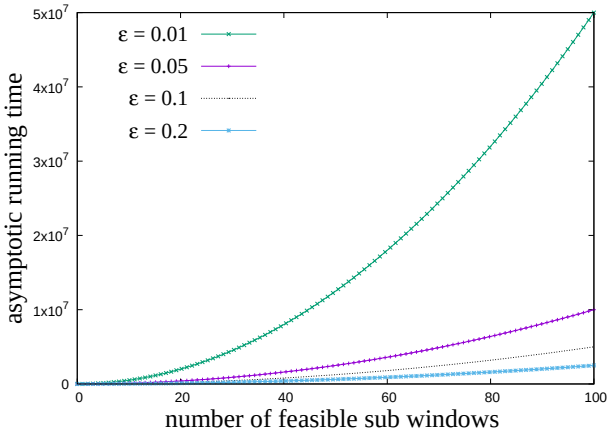


Fig. 1. Asymptotic bound on the running time of Algorithm 2

$\frac{1}{1+\frac{(\frac{2}{c(E)})\,\text{intersct}}{\text{c}_{\min}}} - \epsilon = \frac{\text{c}_{\min}}{\text{c}_{\min}+(\frac{2}{c(E)})\,\text{intersct}} - \epsilon$ *approximation for the wide instances and* $\frac{1}{1+\frac{2\,\text{intersct}}{c(E)\,\text{c}_{\min}}} - \epsilon = \frac{\text{c}_{\min}}{\text{c}_{\min}+2\,\text{intersct}\,/c(E)} - \epsilon$ *for the narrow instances. Thus, by choosing a schedule with a larger profit, we get a* $\frac{\text{c}_{\min}}{2\,\text{c}_{\min}+(\frac{4}{c(E)})\,\text{intersct}} - \epsilon$*-approximation. In addition, the running time of the algorithm is* $O(\frac{n^2\cdot|E|}{\epsilon})$.

The proof can be found in the Appendix.

To get a better idea of how the time complexity of Algorithm 2 grows, in Figure 1 we plot the asymptotic bound on the running time, $\frac{n^2\cdot|E|}{\epsilon}$, as a function of the number of feasible sub-windows for starting a flow. Each curve corresponds to a certain ϵ value. In this graph, we use $E = 50$, and the maximum number of feasible sub-windows is 1,000.

IV. The Network Assigned Bandwidth (NAB) Scheduling Problem

We now consider the scheduling problem for the NAB model, where bandwidth is allocated to flows by the network. To make the discussion general, we assume that there exists a bandwidth allocation function f, such as max-min fairness, which determines the bandwidth allocated to each flow at every time, and guarantees that the total bandwidth allocated to all the flows that share the same link at a given time does not exceed the link capacity.

Problem 2 (NAB)

Input: The same as in CAB (Problem 1), except that possible bandwidth rates are not given. Instead, the scheduler is given a dynamic bandwidth allocation function f.

Objective: Determine a routing path and a transmission time for each flow, such that the total profit is maximized, assuming that bandwidth allocation is based on f. A flow is said to acquire its corresponding profit if it starts after the release time of a window and finishes before the deadline of the same window.

There are two important differences between CAB and NAB:

1) In NAB, the bandwidth allocated to a flow is not necessarily fixed for the entire lifetime of a flow.
2) In NAB, bandwidth allocation is not determined by the controller.

Like CAB, NAB is NP-hard and does not admit a PTAS if P $\neq$ NP.

Theorem 4. *NAB is MAX SNP-hard even in the special case where the network has a single edge of unit capacity, all the flows have the same size, there are 2 windows per flow that do not intersect (in time), and the profits of all windows are unitary.*

Proof: The proof is similar to that of Theorem 1. ■

We start with an online greedy algorithm for NAB:

Algorithm 3. *(An online greedy algorithm for NAB)*

1) *When the release time t of a window w of an unscheduled flow is reached: schedule this flow into the network at t over the path of w if the following two conditions hold:*
   - (c_1) *the bandwidth share to be acquired by this flow on the path of w allows the flow to finish before the deadline associated with w.*
   - (c_2) *the bandwidth acquired by this flow does not prevent any running flow from finishing on time.*
2) *When a flow finishes at time t, let W be the set of windows for which the following holds (a) every $w \in W$ belongs to a non-scheduled flow; (b) the release time of every $w \in W$ is $\leq t$ and the deadline is $\geq t$. Try to schedule the windows of W at t, one by one, in the order of their release times. A window w can be scheduled if conditions (c_1) and (c_2) above hold for it. If a window is scheduled, all the other windows associated with the same flow are removed from W.*

Note that if there are multiple relevant windows in step (1) of the algorithm, as would be the case if flows have multiple possible paths, these windows can be examined in any arbitrary order.

Algorithm 3 is an online algorithm. Thus, the scheduler can make a scheduling decision without knowing in advance the future flows. To use this algorithm offline, the controller needs to sort all the windows in ascending order of their release times.

Let m be the total number of flows and let n be the total number of windows of all the flows. The running time of Algorithm 3 is $O(m\log(m)nt(m,E))$, where $t(m,E))$ is the time needed to verify that a certain flow can be admitted without violating the termination time of previously admitted flows. For instance, if bandwidth is allocated by the network according to max-min fairness, $t(m,E) = m^2\,|E|$ and the total running time is $O(m^3n\,|E|)$.

Lemma 1. *Regardless of how flows are chosen in step (2), for every dynamic allocation f and for any $\epsilon > 0$, there exists an instance of NAB for which the profit of the schedule returned by Algorithm 3 is $< \epsilon \times$ OPT, where* OPT *is the optimum profit.*

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.2015.2487964, IEEE Transactions on Cloud Computing

Proof: Consider a network with a single edge whose capacity is 1. Suppose there are only two flows to schedule. The first flow is released at $t = 0$, has a deadline at $t = 1$, a size of 1, and a profit of $\epsilon/2$. The second flow is released at $t = 0.1$, has a deadline at 1.1, a size of 1, and a profit of 1. Obviously, only one flow can be scheduled. For $\epsilon < 2$, the optimal solution is to schedule the second flow at its release time, which yields $\text{OPT} = 1$. However, Algorithm 3 schedules the first flow, thereby obtaining a profit of $\epsilon/2 < \epsilon \times \text{OPT}$. ■

Next, we modify Algorithm 3 into a 2-phase offline greedy algorithm. In the new algorithm the transmission flows are preordered according to some criterion, such as release time, profit, etc. Then, the algorithm goes over the ordered list and considers one flow at a time. For each considered flow, the algorithm checks whether it can be scheduled into the network upon its release time without disturbing already scheduled flows. If so, the flow is scheduled. In the second phase, the algorithm tries to schedule each non-scheduled flow.

Algorithm 4. *(An offline greedy algorithm for NAB)*

1) *Sort all the windows of all the flows in advance, according to some criterion to be discussed later. Let the sorted list be U.*
2) *Go over the sorted list. For each window w, perform as follows*
   - *Schedule w into the network at its release time t if the following two conditions hold:*
     - (c_1) *the bandwidth share to be acquired by this flow on the path of w allows the flow to finish before the deadline associated with w.*
     - (c_2) *scheduling the flow of w into the network at t does not prevent flows that have already been scheduled from finishing on time;*
   - *When a flow of a window w is scheduled, remove from U all the windows associated with the same flow, and add the instance to the list S.*
   - *Update the termination time of each instance in S and sort the flows in this list in ascending order of their termination times.*
3) *While S is not empty do:*
   a) *Let t be the termination time of the first instance in S. Since S is sorted in ascending order of instance termination times, this instance is the first to terminate from all the instances in S. Remove this instance from S and try to schedule unscheduled flows in the following way.*
   b) *Try to schedule the windows of U at t, one by one, in their sorted order. A window w can be scheduled if conditions (c_1) and (c_2) above hold for it. If a window is scheduled, remove this window and all the other windows associated with the same flow from W. In addition, update the termination time of each instance in S and sort this list again.*

In practice, the running time of Algorithm 4 is shorter than of Algorithm 3. However, their asymptotic running times are equal if we ignore the sorting phase of Algorithm 4. Recall that when Algorithm 3 is executed offline, its input windows must also be sorted according to their starting times.

Lemma 2. *Suppose that the U list in Algorithm 4 is sorted in a decreasing order of $\frac{\alpha\beta\gamma}{\delta}$, where α, β, γ and δ are defined for every window w as follows: (a) α is either 1 for all the windows, or equal to $p(w)$; (b) β is either 1 for all the windows, or equal to the minimum capacity on the path of w; (c) γ is either 1 for all the windows or equal to $d(w) - r(w)$, where $r(w)$ and $d(w)$ are the release time and deadline of w; and (d) δ is the size of the corresponding flow. Then, for every dynamic allocation f and for any $\epsilon > 0$ there exists an instance of NAB with one window per flow for which the profit of the schedule returned by Algorithm 4 is $< \epsilon \times \text{OPT}$, where OPT is the optimum profit.*

Proof: If $\alpha = \beta = \gamma = \delta = 1$, then the scheduling is arbitrary and the claim holds. We next prove the lemma for the case where $\delta \neq 1$. Consider a network with a single edge of a unitary capacity and two flows. The first flow is released at $t = 0$, has a deadline at $t = 1$, a size of $\epsilon/3$, and a profit of $\epsilon/2$. The second flow is released at $t = 0$, has a deadline at 1, a size of 1, and a profit of 1. Obviously, only one flow can be scheduled. For every $0 < \epsilon < 2$, the optimal solution is to schedule the second flow at its release time, which yields $\text{OPT} = 1$. However, Algorithm 4 schedules the first flow at its release time, thereby obtaining a profit of only $\epsilon/2 < \epsilon \times \text{OPT}$.

Next, we prove the lemma for the case where $\alpha \neq 1$ and $\delta = 1$. Consider a network with a single edge of a unitary capacity and $n + 1$ flows. The first flow is released at $t = 0$, has a deadline at $t = 1$, a size of 1, and a profit of 1. Each of the remaining n flows is released at $t = 0$, has a deadline at 1, a size of $\frac{1}{n}$, and a profit of 0.9. The algorithm schedules only the first flow, thereby obtaining a profit of 1, while the optimum solution is to schedule the other n flows, obtaining a profit of $0.9n$. For a large enough n, $1 < \epsilon \times \text{OPT} = \epsilon \times 0.9n$.

Finally, we prove the lemma for the case where $\beta \cdot \gamma \neq 1$ (i.e., $\beta \neq 1$ or $\gamma \neq 1$). Consider a network with a single edge of a unitary capacity and two flows. The first flow is released at $t = 0$, has a deadline at $t = 1.1$, a size of 1, and a profit of $\epsilon/2$. The second flow is released at $t = 0$, has a deadline at 1, a size of 1, and a profit of 1. Obviously, only one flow can be scheduled. Therefore, for every $0 < \epsilon < 2$, the optimal solution is to schedule the second flow at its release time, obtaining $\text{OPT} = 1$, while Algorithm 4 schedules the first flow at its release time, obtaining a profit of only $\epsilon/2 < \epsilon \times \text{OPT}$. ■

V. CAB vs. NAB: Some Special Cases

In this section we compare the optimal solution for CAB to the optimal solution for NAB. The results of such a comparison cannot be easily predicted because each model has its own advantages and disadvantages. CAB is more flexible because its allocation is not restricted by the "external" dynamic allocation rule. On the other hand, CAB's allocated bandwidth is fixed for the whole lifetime of an instance

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.2015.2487964, IEEE Transactions on Cloud Computing

whereas in NAB the bandwidth is dynamically adapted to the actual load and availability.

In the following we show that in the special case where every flow has a single window, all the flows share the same route, and they all have the same release time, then the two optimal values are equal, provided that the possible bandwidth rates include $c(J)$ for each flow J. This result holds regardless of how bandwidth is allocated by the network to the scheduled flows in the NAB model.

Given an instance Π of CAB and NAB, denote the profit of its optimum solution by $\text{OPT}(\text{CAB})(\Pi)$ and $\text{OPT}(\text{NAB})(\Pi)$ respectively.

Lemma 3. *When every flow has a single window, all the flows share the same route, all the flows have the same release time, and for each flow J the bandwidth rate $c(J)$ can be allocated by CAB, the following holds:*

$$\text{OPT}(\textit{CAB})(\Pi) = \text{OPT}(\textit{NAB})(\Pi).$$

Proof: Recall that $c(J)$ is the bottleneck of the path of flow J. This path is unique in the considered case because J has only one window. We decompose the proof into two parts. In the the first part we show that under the conditions above, $\text{OPT}(\text{NAB}) \geq \text{OPT}(\text{CAB})$ holds. The second part shows the other direction.

Let the release time of all the flows be $t = 0$, and let e be the edge with the minimum capacity along the shared path. Let the sequence of instances $i_1, i_2, \ldots, i_n$ be a feasible CAB schedule. Each i_k starts after the release time (0) and finishes before the deadline of the considered flow. We now construct a schedule for NAB using the following procedure. We start with $t = 0$ and go over the scheduled instances in a non-decreasing order of their termination times. Let this order be $i_1, i_2, \ldots, i_n$. Then, for $k = 1, 2, \ldots, n$ we perform the following two steps. First, we denote the flow to which i_k belongs as J_k, and schedule it at time t. Second, we set t to be the finishing time of J_k if this flow was running alone.

We now show that when the bandwidth is allocated according to this procedure, all the flows finish by their deadlines. We show this by induction on the iterations of the procedure. Before rescheduling J_1, all the rescheduled flows $(\emptyset)$ have finished by their deadlines. Thus, the induction basis holds. Assuming the claim holds for the first k rescheduled flows, we now prove it for the $(k+1)$th flow. Suppose, on the contrary, that the claim does not hold for this flow; namely, that just before rescheduling J_{k+1} at time t, $d(J_{k+1}) - t < \frac{s(J_{k+1})}{c(e)}$ holds, where $s(J)$ is the size of flow J. In other words, the time until the deadline of J_{k+1} is not enough for the completion of this flow even if it was the only flow in the network. However, since $t = \sum_{l=1,\ldots,k} \frac{s(J_l)}{c(e)}$, we get $d(J_{k+1}) < \frac{\sum_{l=1,\ldots,k+1} s(J_l)}{c(e)}$, which contradicts the assumption that $i_1, i_2, \ldots, i_n$ is a feasible schedule in the CAB model, taken in the non-decreasing order of the flows' finishing times, such that all of them start after the release time and finish not later than by their respective deadlines.

The proof of the other direction, i.e., that $\text{OPT}(\text{NAB}) \leq \text{OPT}(\text{CAB})$, is similar, except for the following changes:

- We are given a schedule of NAB where every scheduled flow starts after its release time (0) and finishes not later than its deadline. We construct a schedule for CAB by a procedure that is completely analogous to the one described earlier.
- We go over the scheduled flows in a non-decreasing order of their finishing times. Each flow J is rescheduled to run using the bandwidth of $c(J)$, which is the maximum bandwidth it can use. ■

We next show that under a different set of conditions, we can state that $\text{OPT}(\text{CAB})(\Pi) \geq \text{OPT}(\text{NAB})(\Pi)$. For this set of conditions, it can be shown that $\text{OPT}(\text{CAB})(\Pi) \leq \text{OPT}(\text{NAB})(\Pi)$ does not necessarily hold.

Lemma 4. *When every flow has a single window, all the flows have the same release time and deadline, and CAB allows every bandwidth to be allocated,* $\text{OPT}(\textit{CAB})(\Pi) \geq \text{OPT}(\textit{NAB})(\Pi)$ *even if flows do not necessarily share the same route.*

Proof: Suppose that all the release times are at $t = 0$ and all the deadlines are at $t = 1$. Let $t_1, t_2, \ldots, t_n$ be a feasible NAB schedule. In this schedule, $t_k > 0$ is the starting time of flow J_k and every flow finishes before $t = 1$. We now construct a CAB schedule by the following procedure. For each $k = 1, 2, \ldots, n$, allocate to the instances of J_k that begin at $t = 0$ a bandwidth of $\frac{s(J_k)}{1-0} = s(J_k)$, where $s(J)$ is the size of flow J. This instance will finish exactly by $t = 1$.

It is left to prove that the obtained CAB schedule is feasible. Suppose, to the contrary, that there exists an edge e whose capacity is exceeded, i.e., $\sum_{J|e\in J} s(J) > c(e)$, or equivalently, $\frac{\sum_{J|e\in J} s(J)}{c(e)} > 1$. But this contradicts the assumption that $i_1, i_2, \ldots, i_n$ is a feasible NAB schedule. ■

VI. Simulation Study

A. The Simulated Algorithms

In this section we present a simulation study, in order to address the three challenges first raised in Section I:

- Deciding what are the best algorithms for CAB and NAB.
- Deciding whether there is a significant performance advantage to NAB, although we know that their running time renders them impractical.
- Understanding the correlation between the width of the flows' windows and the performance of the scheduler.

The algorithms simulated in this section are as follows:

Alg-2 (for CAB): This algorithm implements Algorithm 2 on the wide instances and on the narrow instances. Then, it chooses the schedule with the largest profit to guarantee the approximation ratio stated in Theorem 3. After choosing either the narrow or the wide instances, the algorithm tries to improve the solution by adding non-selected instances. To this end, the algorithm sorts the non-selected instances in descending order of the ratio between their profit and size. Recall that each such an instance is already associated with a specific bandwidth allocation.

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.2015.2487964, IEEE Transactions on Cloud Computing

Alg-3 and Alg-4 (for NAB): These algorithms implement Algorithm 3 and Algorithm 4, respectively. In Alg-4, the windows in U are sorted in a decreasing order of

$$\frac{p(w)c(w)(d(w)-r(w))}{s(w)},$$

where for every window w, $p(w)$ is the window's profit, $c(w)$ is the minimum capacity on its path, $r(w)$ and $d(w)$ are its release time and deadline, and $s(w)$ is the size of the flow of w. The intuition is to generalize the well-known greedy algorithm for the Knapsack problem, which aims at maximizing the profit of the items chosen for a knapsack with a given capacity [14]. In the Knapsack problem each item J only has a profit $p(J)$ and a size $s(J)$, and the items are sorted according to $p(J)/s(J)$, which indicates the profit per size unit. In our problem we want to take into account two additional parameters: the width of each window $d(w)-r(w)$ and the availability of resources along the path, which is represented by $c(w)$. In general, windows whose $d(w)-r(w)$ is wider and whose $c(w)$ is bigger can be more easily scheduled, and therefore their "overall scheduling cost" is smaller. Thus, we put both factors in the numerator of our sorting criterion.

Even if we assume that bandwidth is shared according to max-min fairness, in which case $t(m,E) = m^2\,|E|$ holds, the running time of both algorithms renders them impractical also for the simulations. To speed up these algorithms, we slightly change them as follows. First, the rescheduling phase is executed only until the first time it succeeds. Second, when a flow terminates, we invoke the rescheduling phase only if sufficient time has elapsed since the previous rescheduling attempt. Third, a rescheduling is attempted only for flows whose path shares at least one edge with the path of the terminating flow. While these enhancements do not improve the asymptotic running time of the algorithms, they improve the actual time and make them feasible for our simulation study.

Alg-5 and Alg-6 (for CAB): These algorithms are the CAB equivalent of Alg-3 and Alg-4 respectively, without the speedup enhancements mentioned above. In contrast to Alg-3 and Alg-4, Alg-5 and Alg-6 need to determine the bandwidth allocated to each scheduled flow. This bandwidth is chosen to be the widest possible bandwidth available along the path from the source to the destination. We will also discuss a version of Alg-5 and of Alg-6 where the allocated bandwidth is the narrowest that allows the flow to finish on time.

B. Simulation Model

Because we have to make many decisions about the simulation model, and it is impossible to present and discuss the results for all possible combinations, we focus here on datacenters connected by a backbone and assume that the backbone is the bottleneck. This is a reasonable assumption because the backbone consists of long-distance, expensive links. Thus, we abstract each datacenter as a node in the backbone graph. We build the backbone graph using Waxman's model [22] in the following way[3]. We first randomly place 30 network nodes in a $1.0{*}1.0$ plane, uniformly and independently. The probability to have an edge with a unit capacity between every pair of nodes is $a \cdot \exp(\frac{-d}{b \cdot L})$, where d is the Euclidean distance between the two nodes, L is the maximum Euclidean distance between any two nodes, $a = 0.25$, and $b = 0.5$. With these parameters, the average number of edges in each simulated graph is ≈ 90. If the resulting graph is not connected, it is ignored and a new graph is drawn.

We start with the case where the controller is given only one window for each flow, and then study the case where every flow has multiple windows, each with different times and/or paths. The release time for this possible window is randomly chosen according to the uniform distribution between 0 and 100, namely, $U[0,100]$. As already indicated, the width of the possible window is an important performance parameter of the various algorithms. Therefore, we will show several graphs with different widths: $U[5,30]$ (i.e., the width is randomly chosen between 5 and 30 using uniform distribution), $U[5,55]$, and $U[5,15]$. We use two approaches to assign profit to flows. In the first, the profit is randomly chosen according to $U[0,10]$. In the second, the profit is a product of a random number chosen from $U[0,10]$ and the flow's size.

Each point on each graph is an average of 20 different experiments with the same parameters. In all the graphs we show the "normalized profit" as a function of the "normalized load." To compute the normalized profit, we calculate the total profit obtained by each algorithm and divide this by the total profit of all the flows. To determine the normalized load, we first compute the load imposed by each flow as the flow's size multiplied by the length of the shortest path from the flow's source and destination. We then sum up the loads of all the flows and divide the result by the network capacity. The latter is defined as the aggregated bandwidth of all the network links multiplied by the time elapsed between the first release time and the last deadline. To increase the normalized load, we either increase the number of flows or increase their average size. The decision does not affect the results.

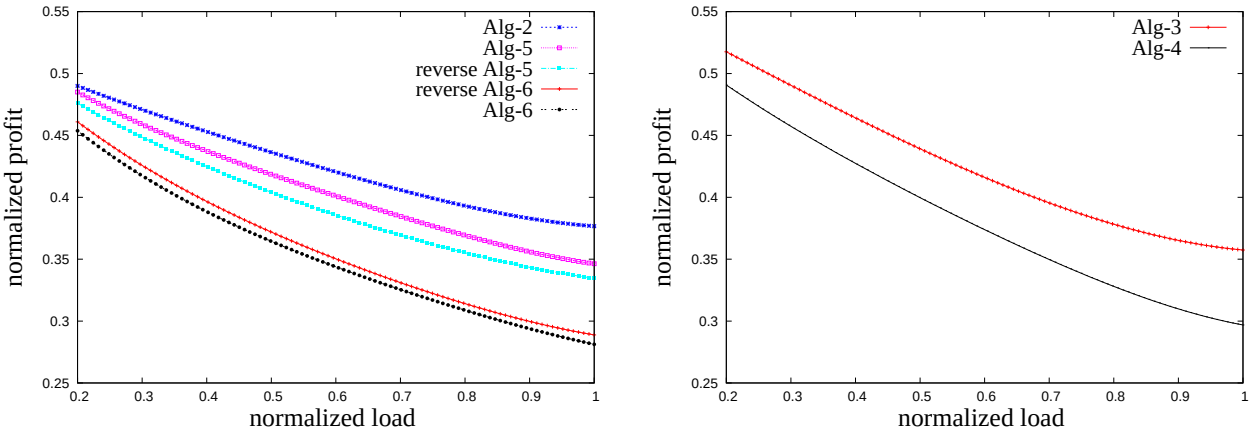
C. Simulation Results

As indicated earlier, one of our goals is to understand the strength of the correlation between the scheduler's flexibility and performance. The scheduler's flexibility is represented by the ratio between the average width of the possible windows and the total time. This ratio is referred to as the "flexibility ratio." Obviously, the scheduler has more flexibility when this ratio is closer to 1.

We start with the case where the possible window is randomly chosen for each flow according to $U[5,30]$, and the total time is 130. This implies that the flexibility ratio is $0.5 \cdot (30+5)/130 = 0.13$. This case is presented in the two graphs of Figure 2. As expected, for all the algorithms, an increase in load results in a decrease in normalized profit.

[3]Waxman's model is very often used to abstract network backbones. However, when we used other models, we did not notice important differences.

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.2015.2487964, IEEE Transactions on Cloud Computing



(a) CAB algorithms

(b) NAB algorithms

Fig. 2. Performance of the various algorithms for flexibility ratio= 0.13

This is simply because we try to accommodate more and more flows during the same interval and therefore using the same capacity. Note that when the load increases, the total profit also increases (this graph is not shown) because the scheduler has more flows to choose from. However, the fraction of scheduled flows decreases. It is also evident that the relative performance of the various algorithms is not affected by the load.

The performance of the CAB algorithms is shown in Figure 2(a), and of the NAB algorithms in Figure 2(b). For the CAB algorithms, the best performance is obtained by Alg-2, which is based on the algorithms developed in Section III. The second best is Alg-5, which also performs better than Alg-6. The superior performance of Alg-5 over Alg-6 and, equivalently, of Alg-3 over Alg-4 in Figure 2(b), might come as a surprise because Alg-3 and Alg-5 are online algorithms while Alg-4 and Alg-6 are offline algorithms. However, the advantage of Alg-3 and Alg-5 can be attributed to the fact that they receive the flows sorted in ascending order of release times. Thus, both algorithms try to schedule every flow from the moment this flow is ready for transmission.

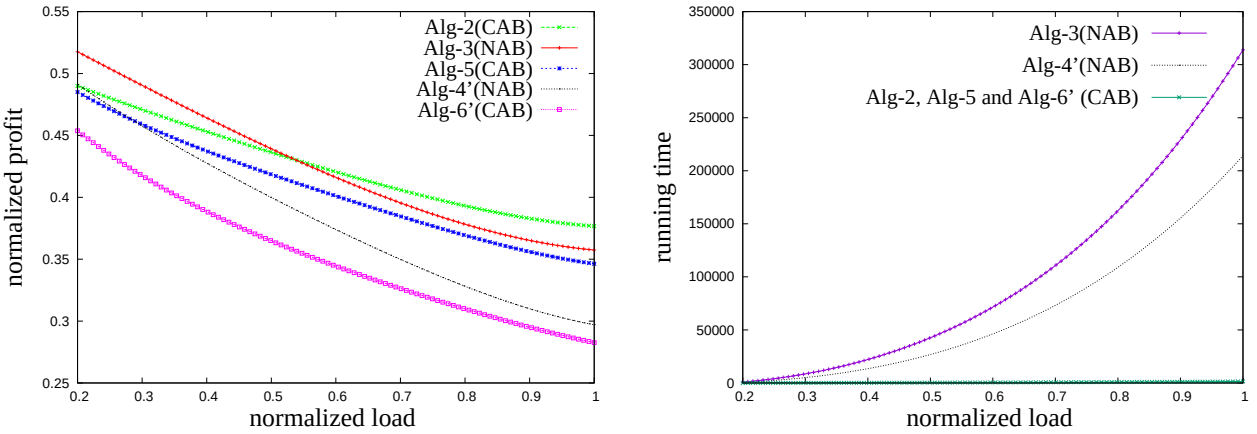
Figure 2(a) also shows the performance of two algorithms that have not been mentioned earlier: reverse Alg-5 and reverse Alg-6. Recall that when Alg-5 and Alg-6 sort the flows, they need to determine the bandwidth allocated to each scheduled flow. This bandwidth is chosen to be the widest possible bandwidth available along the path from the source to the destination. In contrast, the reverse versions of these algorithms choose the narrowest bandwidth that allows the flow to finish on time.

Figure 3(a) shows the same curves of Alg-2, Alg-3 and Alg-5 from Figure 2(a) and (b), in order to compare the performance of CAB vs. NAB for flexibility ratio of 0.13. We can see that the best CAB algorithm (Alg-2) performs better than the best NAB algorithm (Alg-3) for high loads (> 0.5), whereas for low loads the situation is reversed. The graph shows two new curves: of Alg-4' and Alg-6'. These two algorithms are similar to Alg-4 and Alg-6 respectively, with the only change that in their second phase these two algorithms sort the instances according to their release times and attempt to schedule new instances whenever a scheduled instance ends. Thus, these algorithms imitate the second phase of Alg-3 and Alg-5. We still see that the performance of Alg-3 and Alg-5 is significantly better than that of Alg-4' and Alg-6', which indicates that the advantage of these algorithms should be attributed to their first phase, were they sort the instances according to the instances' release times. To compare the actual running times of the various algorithms, we measured them (in milliseconds) on an Intel 2.8 GHz computer with Windows 7. The results are presented in Figure 3(b) when full load is obtained by 350 flows and each algorithm is executed 20 times for each input set. We observe that the running times of the NAB algorithms are significantly longer than those of the CAB algorithms.

Figure 4(a) shows the performance of the best CAB and NAB algorithms when the flexibility ratio increases to $0.5 \cdot (55+5)/155 = 0.19$. This ratio is obtained by drawing the width of the possible windows from $U(5, 55)$ and extending the total time to 155. All the other parameters are identical to those of Figure 2. When we compare Figure 4(a) to Figure 3(a), we see that the performance is indeed considerably better for all the algorithms. This indicates that by giving the scheduler enough flexibility, we can increase its performance even for light loads. Another interesting observation can be made with regard to the performance of CAB vs. NAB. While in Figure 2 we see that the best NAB algorithms perform slightly better than the best CAB algorithms, in Figure 4(a) this situation is reversed. *This interesting observation indicates that the CAB algorithms are more sensitive to small flexibility factors than their NAB counterparts.* To verify this observation, we decrease the flexibility ratio to $0.5 \cdot (15+5)/115 = 0.11$. The results are presented in Figure 4(b). Indeed, we can see that all the algorithms perform worse than in Figure 2, and that the difference in performance between the best CAB algorithm and the best NAB algorithm is now 10%, compared to less than 5% in Figure 2.

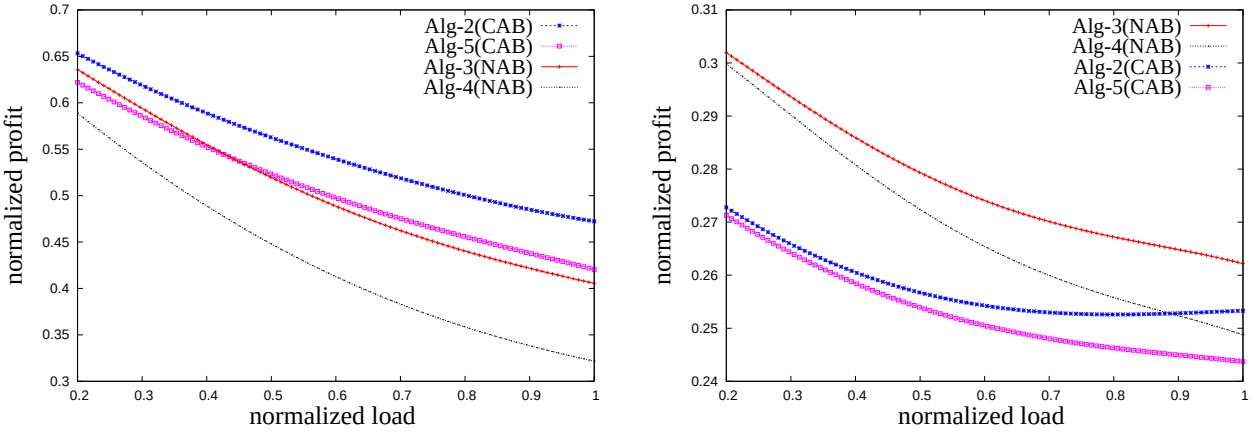
In Figure 5 we repeat the same experiment as in Figure 2,

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.2015.2487964, IEEE Transactions on Cloud Computing



(a) performance (b) running time (milliseconds)

Fig. 3. CAB vs. NAB for flexibility ratio= 0.13 (performance and running time)



(a) flexibility ratio= 0.19 (b) flexibility ratio= 0.11

Fig. 4. CAB vs. NAB for two different flexibility ratios

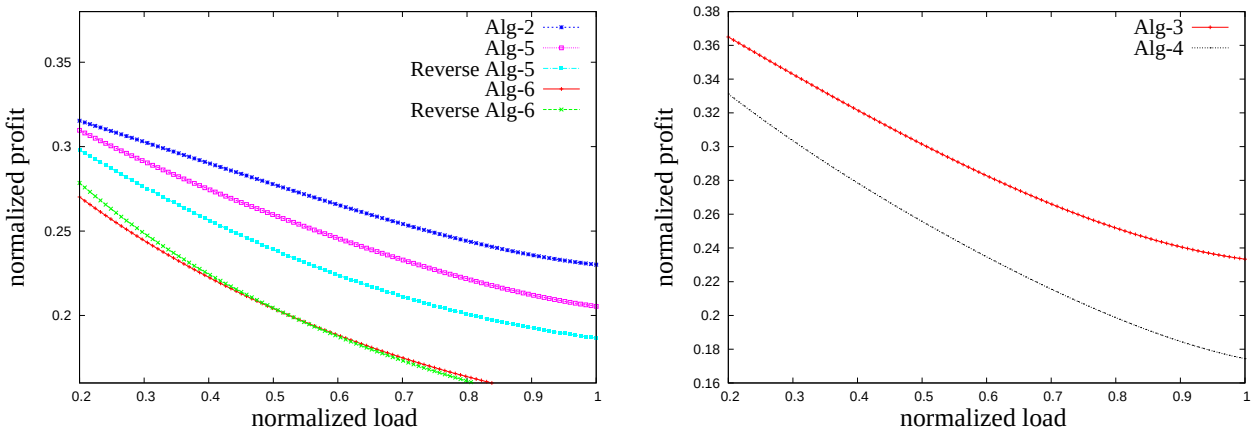
except that the profit of each flow is proportional to the flow's size. While we see a drop in the performance, the relative behavior of the various algorithms does not change compared to Figure 2. The performance drop can be explained by the fact that in this case the bigger flows are assigned more profit than in the previous case, but their possible window times do not change.

Recall that in all the simulations described so far we used only one possible transmission window for each flow. We now show simulation results for the case where each flow has multiple possible transmission windows, not necessarily with the same profit. In Figure 6(a) we consider the same simulation setting as in Figure 2, except that every flow now has three possible transmission windows, each associated with a different path. The first path is chosen to be a shortest path (as in Figure 2). For the second path, a random intermediate node v is chosen and a shortest path from the source to v is used, followed by a shortest path from v to the destination (this is the concept known as 1-hub routing [8]). A different random intermediate node v' is chosen for the third path; once again, a shortest path from the source to v' is used, followed by a shortest path from v' to the destination. We see a *substantial improvement* for both models and for all normalized load values. For example, for Alg-2 (CAB), the normalized profit increases from 0.44 to 0.75 when the normalized load is 0.2 and from 0.44 to 0.5 when the normalized load is 0.5.

The simulation setting for Figure 6(b) is similar to that in Figure 6(a), except that for each possible transmission window the profit depends on the length of its path; i.e., we multiply the flow profit by 1/path_length. This gives the scheduler motivation to prefer transmission windows whose path is short compared to other transmission windows of the same flow. By comparing Figure 6(b) to Figure 6(a), we see *an additional substantial improvement* for both CAB and NAB, for all traffic loads.

Finally, Figure 6(c) shows results for the same simulation setting as in Figure 6(a), but this time higher profit is assigned to transmission windows with earlier release time. We do this by assigning different release times to the three possible transmission windows and multiplying the profit of the window with the first release time by 1, the profit of the window with the second release time by 1/2, and the profit

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.2015.2487964, IEEE Transactions on Cloud Computing



(a) The CAB algorithms

(b) the NAB algorithms

Fig. 5. Performance of the various algorithms for flexibility ratio= 0.13 (but profit is proportional to the flow size)

of the window with the third (latest) release time by 1/3. By comparing Figure 6(c) to Figure 6(a), we see again an additional substantial improvement for both CAB and NAB, and for all traffic loads. We also see that for light loads the performance in Figure 6(c) is better than that in Figure 6(b) and vice versa.

We can summarize this section as follows:

1) Our new approximation algorithm (Algorithm 2) is the best algorithm for CAB, not only because it is faster, but also because it yields the top performance.
2) The most important factor in the performance of all the algorithms is the length of the possible windows.
3) Even if the NAB scheduler has full knowledge of the exact bandwidth allocated by the network to every flow, the NAB algorithms perform better than the CAB algorithms only for low loads and large flexibility ratio values. **This indicates that CAB scheduling is the best model for inter-datacenter flow scheduling.**
4) By allowing each flow to be scheduled over one of multiple (three) different transmission windows, we can substantially increase the performance. Moreover, by tuning the profit of each window according to attributes such as path length and release time, further substantial improvement is obtained.

VII. Conclusions

In this paper we defined and addressed a new scheduling problem, called CAB. This problem arises when many data-centers need to transfer astronomical amounts of data among themselves on a timely basis. We proved that this problem is not only NP-hard, but that it also does not admit a PTAS. Then, we developed an efficient approximation algorithm and two heuristics for solving it. We also defined a related problem, called NAB-scheduling, which differs from CAB-scheduling in the way bandwidth is assigned to flows. Our approximation algorithm was not only faster than the heuristics, but was also shown to perform better. We also showed that the NAB algorithms, whose implementation is impractical, have no performance advantage compared to the CAB algorithms. Thus, we believe that CAB is the best model for inter-datacenter scheduling, which means that a TCP$^-$ protocol should be used in the Transport layer, and bandwidth should be allocated to the scheduled flows by the controller and not by the network.

Appendix

The Proof of Theorem 2

The first two parts of the theorem are proven by first showing that every $\tilde{i}$-maximal schedule is an appropriate approximation with respect to p_1, and then applying the local ratio technique to prove the approximation ratio by induction on the algorithm's invocation.

To prove part (a) of the theorem, we start with three lemmas.

Lemma 5. *If for every instance $i : w(i) > \rho$, then at most $\left\lceil \frac{1}{\rho} \right\rceil - 1$ instances can be scheduled at the same time on the same edge.*

To prove this lemma, we schedule at least $\left\lceil \frac{1}{\rho} \right\rceil$ instances at the same time. Then, their aggregated bandwidth is larger than $\rho \frac{1}{\rho} = 1$, which is the maximum normalized capacity of an edge.

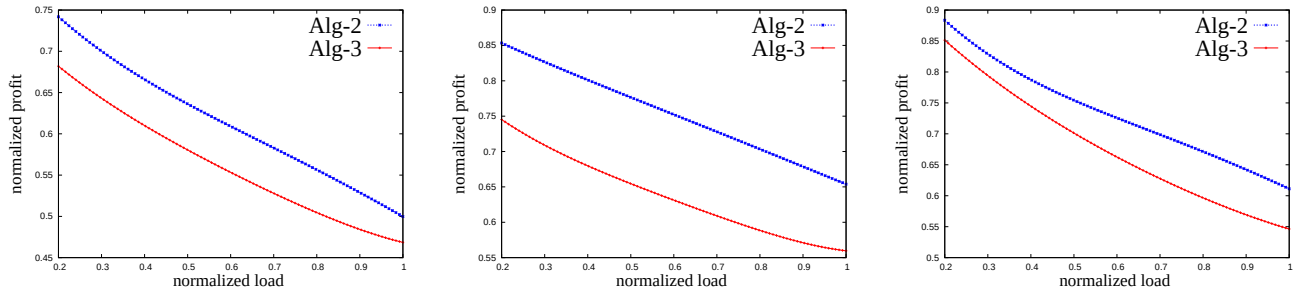
Lemma 6. $\forall S : p_1(S) \leq p_1(\tilde{i}) \cdot \left(1 + \frac{\alpha(\left\lceil \frac{1}{\rho} \right\rceil - 1) \cdot \text{intersct}(\tilde{i})}{\text{c}_{\min}(\tilde{i})}\right)$.

To prove this lemma, we note that the bound $p_1(S) \leq p_1(\tilde{i}) \cdot b_{\text{opt}}$ holds for

$$b_{\text{opt}} = 1 + \alpha(\left\lceil \frac{1}{\rho} \right\rceil - 1) \cdot \sum_{e | e \in i \cap \tilde{i} \text{ for } i \in \mathcal{I}(\tilde{i})} \frac{1}{\text{c}_{\min}(\tilde{i})}$$
$$= 1 + \frac{\alpha(\left\lceil \frac{1}{\rho} \right\rceil - 1) \cdot \text{intersct}(\tilde{i})}{\text{c}_{\min}(\tilde{i})}.$$

Obviously, we can take at most one instance from $\mathcal{A}(\tilde{i})$ in any solution. Moreover, from Lemma 5 it follows that, for

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.2015.2487964, IEEE Transactions on Cloud Computing



(a) each window has a different path (b) profit depends on the path (c) profit depends on the release time

Fig. 6. Performance of the best CAB/NAB algorithms when each flow has three possible transmission windows

every edge common to $\tilde{i}$ and to an instance in $\mathcal{I}(\tilde{i})$, we can add at most $\frac{\lceil \frac{1}{\rho} \rceil - 1}{c(i \cap \tilde{i})}$ to the profit. This lemma is relevant since by the choice of $\tilde{i}$, all the instances that intersect with $\tilde{i}$ in time also intersect with each other.

Lemma 7. *For every $\tilde{i}$-maximal S, $p_1(S) \geq p_1(\tilde{i}) \cdot \min\left\{1, \frac{1}{\mathrm{c}_{\max}(\tilde{i})}\right\}$ holds.*

This lemma holds because any $\tilde{i}$-maximal solution either contains an instance of $\mathcal{A}(\tilde{i})$ or an obstructing instance from $\mathcal{I}(\tilde{i})$.

We now prove part (a) of the theorem. From Lemmas 6 and 7 we obtain an approximation ratio of at least $\frac{1}{1+\frac{(\lceil \frac{1}{\rho} \rceil - 1)\,\mathrm{intersct}(\tilde{i})}{\mathrm{c}_{\min}(\tilde{i})}} = \frac{\mathrm{c}_{\min}(\tilde{i})}{\mathrm{c}_{\min}(\tilde{i}) + (\lceil \frac{1}{\rho} \rceil - 1)\,\mathrm{intersct}(\tilde{i})}$, which can be simplified to:

$$\frac{1}{1 + \frac{(\lceil \frac{1}{\rho} \rceil - 1)\,\mathrm{intersct}}{\mathrm{c}_{\min}}} = \frac{\mathrm{c}_{\min}}{\mathrm{c}_{\min} + (\lceil \frac{1}{\rho} \rceil - 1)\,\mathrm{intersct}}.$$

To prove part (b) of the theorem, we need the following two lemmas.

Lemma 8. *For every S, $p_1(S) \leq p_1(\tilde{i}) \cdot \left(1 + \frac{\frac{1}{c(\tilde{i}) - \rho} \cdot \mathrm{intersct}(\tilde{i})}{\mathrm{c}_{\min}(\tilde{i})}\right)$ holds.*

To prove this lemma, we note that $p_1(S) \leq b_{\text{opt}}$ holds for $b_{\text{opt}} = 1 + \frac{1}{c(\tilde{i}) - \rho} \sum_{e | e \in i \cap \tilde{i}, \text{ for } i \in \mathcal{I}(\tilde{i})} \frac{1}{\mathrm{c}_{\min}(\tilde{i})} = 1 + \frac{\frac{1}{c(\tilde{i}) - \rho} \cdot \mathrm{intersct}(\tilde{i})}{\mathrm{c}_{\min}(\tilde{i})}$, because we can take at most one instance from $\mathcal{A}(\tilde{i})$ in a feasible solution. Also, for every edge common to $\tilde{i}$ and an instance in $\mathcal{I}(\tilde{i})$, we can add at most $\frac{\frac{1}{c(\tilde{i}) - \rho}}{c(i \cap \tilde{i})}$ to the profit. This is because of the capacity constraints, and because all the instances that intersect with $\tilde{i}$ in time also intersect with each other.

Lemma 9. *For every $\tilde{i}$-maximal S, $p_1(S) \geq p_1(\tilde{i}) \cdot \min\left\{1, \frac{1}{\mathrm{c}_{\max}(\tilde{i})}\right\}$ holds.*

This lemma holds because any $\tilde{i}$-maximal solution contains an instance of $\mathcal{A}(\tilde{i})$ or some obstructing instances from $\mathcal{I}(\tilde{i})$ whose total bandwidth is more than $c(\tilde{i}) - w(\tilde{i}) \geq c(\tilde{i}) - \rho$.

We now prove part (b) of the theorem. From Lemmas 8 and 9 we obtain an approximation ratio of at least

$$\frac{1}{1 + \frac{\mathrm{intersct}(\tilde{i})}{(c(E) - \rho)\,\mathrm{c}_{\min}(\tilde{i})}} = \frac{\mathrm{c}_{\min}(\tilde{i})}{\mathrm{c}_{\min}(\tilde{i}) + \frac{1}{c(E) - \rho}\,\mathrm{intersct}(\tilde{i})} \geq \frac{1}{1 + \frac{\mathrm{intersct}}{(c(E) - \rho)\,\mathrm{c}_{\min}}} = \frac{\mathrm{c}_{\min}}{\mathrm{c}_{\min} + \frac{1}{c(E) - \rho}\,\mathrm{intersct}}.$$

To prove part (c) of the theorem, we need the following lemma, which extends a result from [4]:

Lemma 10 (Combining Approximations)**.** *Given a $\frac{1}{\alpha}$-approximation solution for a subset of the input elements and a $\frac{1}{\beta}$-approximation solution for all the remaining input elements, we obtain a $\frac{1}{\alpha+\beta}$-approximation for the whole set by choosing the solution with the larger profit.*

To prove this lemma, denote the two subsets of input elements by $\mathcal{A}_1$ and $\mathcal{A}_2$. Let the output schedule of the $\frac{1}{\alpha}$-approximation for $\mathcal{A}_1$ be $\mathcal{S}_1$, and the output schedule of the $\frac{1}{\beta}$-approximation for $\mathcal{A}_2$ be $\mathcal{S}_2$. Let $\mathcal{S}^*$ be an optimal solution for the whole input set $\mathcal{A}$. Either $p(\mathcal{S}^* \cap \mathcal{A}_1) \geq \frac{\alpha}{\alpha+\beta} p(\mathcal{S}^*)$ or $p(\mathcal{S}^* \cap \mathcal{A}_2) \geq \frac{\beta}{\alpha+\beta} p(\mathcal{S}^*)$ must hold. Therefore, either $p(\mathcal{S}_1) \geq \frac{1}{\alpha} \cdot \frac{\alpha}{\alpha+\beta} p(\mathcal{S}^*) = \frac{1}{\alpha+\beta} p(\mathcal{S}^*)$ or $p(\mathcal{S}_2) \geq \frac{1}{\beta} \cdot \frac{\beta}{\alpha+\beta} p(\mathcal{S}^*) = \frac{1}{\alpha+\beta} p(\mathcal{S}^*)$ holds. Thus, a solution with the greater profit is a $\frac{1}{\alpha+\beta}$-approximation for the whole set $\mathcal{A}$.

We now prove part (c) of the theorem. By setting $\rho = 0.5c(E)$, we obtain the ratio of $\frac{1}{1 + \frac{(\frac{2}{c(E)})\,\mathrm{intersct}}{\mathrm{c}_{\min}}} = \frac{\mathrm{c}_{\min}}{\mathrm{c}_{\min} + (\frac{2}{c(E)})\,\mathrm{intersct}}$ for the wide subset and $\frac{1}{1 + \frac{2\,\mathrm{intersct}}{c(E)\,\mathrm{c}_{\min}}} = \frac{\mathrm{c}_{\min}}{\mathrm{c}_{\min} + 2\,\mathrm{intersct}/c(E)}$ for the narrow subset. Then, part (c) follows directly from Lemma 10.

The Proof of Theorem 3

For $\epsilon = 0$, the algorithm acts as Algorithm 1 would act on an infinite number of instances. Since Theorem 2 and Lemma 10 are correct even for an infinite number of instances, the claim holds in this case. When $\epsilon > 0$, the approximation ratio of Algorithm 2 is reduced by a factor of $1 - \epsilon$. The proof in this case is similar to that proposed in [4], and is briefly presented to keep our paper self-contained.

Since we can partition the windows to the finally obtained windows before the algorithm begins, we can assume without

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.2015.2487964, IEEE Transactions on Cloud Computing

loss of generality that the algorithm never splits windows. Call a window that is deleted during the execution of step (3) of the algorithm a "bad window." When a bad window τ is selected (and deleted), let ϵ_τ be its profit. Consider now how the algorithm runs and how it would have run if we changed the profit of every bad window τ to $p(\tau) - \epsilon_\tau$ and invoked a precise algorithm (the same algorithm, but without deletions in step (3). The precise algorithm can run in the second scenario exactly as the deleting algorithm would do in the first scenario with regard to what is chosen and the final schedule, except that the profit of every bad window τ that has not yet been deleted in the first scenario will be higher by ϵ_τ than its profit in the second scenario. Let p and p' be the profit functions in both scenarios and p_{OPT} and p'_{OPT} be the optima in both scenarios, respectively. For any feasible schedule D, $p(D) \geq p'(D) \geq (1-\epsilon)p(D)$, and therefore, $p'_{\text{OPT}} \geq (1-\epsilon)p_{\text{OPT}}$. So, if the solution S of the precise algorithm is r-approximation, then $p(S) \geq p'(S) \geq rp'_{\text{OPT}} \geq r(1-\epsilon)p_{\text{OPT}}$, and the solution returned by our algorithm is $r(1-\epsilon)$-approximation.

To prove the running time, we extend the analysis of [4] (Section 3.3.1). Thus, we follow their stack-based iterative implementation. At the first phase, we push $\tilde{i}$ onto a stack and iterate till there are no more instances. At the second phase, we pop the items from the stack, adding them if they do not violate the feasibility. Since the algorithm deletes at least one window during each iteration, the number of iterations is bounded by the number of windows (we treat a split window as two windows). This number is less than $\frac{n^2}{\epsilon}$, because the number of non-empty iterations is at most $\frac{n}{\epsilon}$. Note that the instances whose weight is $< \epsilon$ of their original profit can be removed while updating the profits. Thus, their removal does not affect the asymptotic complexity. Consider a non-empty iteration. Choosing i takes $O(n)$ and updating the profits takes $O(n\,|E|)$. Thus, the first phase takes $O(\frac{n^2|E|}{\epsilon})$. The second phase unwinds a stack whose size is $O(\frac{n}{\epsilon})$, since it was built only by non-empty iterations. Checking the feasibility of adding an instance that is known to end first takes $O(|E|\,n)$. Thus, the second phase takes $O(\frac{n^2|E|}{\epsilon})$ time, and the whole algorithm takes $O(\frac{n^2|E|}{\epsilon})$ time.

REFERENCES

[1] S. Agarwal et al. Volley: Automated data placement for geo-distributed cloud services. In *NSDI'2010*.

[2] U. Akyol, M. Andrews, P. Gupta, J. D. Hobby, I. Saniee, and A. L. Stolyar. Joint scheduling and congestion control in mobile ad-hoc networks. In *INFOCOM 2008*.

[3] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: dynamic flow scheduling for data center networks. In *the 7th USENIX Conf. on Networked Systems Design and Implementation*, 2010.

[4] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *J. ACM*, 48(5):1069–1090, 2001.

[5] R. Bar-Yehuda and S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Discrete Mathematics*, 25:27–45, 1985.

[6] R. Bar-Yehuda, M. M. Halldórsson, J. Naor, H. Shachnai, and I. Shapira. Scheduling split intervals. *SIAM Journal on Computing*, 36(1), 2006.

[7] 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 *IEEE INFOCOM 2011*.

[8] R. Cohen and G. Nakibli. On the computational complexity and effectiveness of N-hub shortest-path routing. *IEEE/ACM Transactions on Networking*, 16(3), June 2008. An earlier version was presented in Infocom'2004.

[9] Y. Feng, B. Li, and B. Li. Postcard: Minimizing costs on inter-datacenter traffic with store-and-forward. In *IEEE Distributed Computing Systems Workshops (ICDCSW)*, 2012.

[10] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. In *SIGCOMM 2000*.

[11] S. Floyd, M. Handley, J. Padhye, and J. Widmer. TCP friendly rate control (TFRC): Protocol specification. *RFC 5348*, 2008.

[12] C. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. Achieving high utilization with software-driven wan. In *Sigcomm*, 2013.

[13] S. Jain et al. B4: Experience with a globally-deployed software defined wan. In *SIGCOMM*, 2013.

[14] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, Berlin, 2004.

[15] E. Kohler, M. Handley, and S. Floyd. Datagram congestion control protocol (DCCP). *RFC 4340*, 2006.

[16] N. Laoutaris, M. Sirivianos, X. Yang, and P. Rodriguez. Inter-datacenter bulk transfers with netstitcher. In *Sigcomm*, 2011.

[17] K. Le, R. Bianchini, M. Martonosi, and T. Nguyen. Cost- and energy-aware load distribution across data centers. In *HotPower*, 2009.

[18] J. Liu, F. Zhao, X. Liu, and W. He. Challenges towards elastic power management in internet data centers. In *IEEE International Conference on Distributed Computing Systems*, 2009.

[19] A. Mahimkar et al. Bandwidth on demand for inter-data center communication. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*. ACM, 2011.

[20] R. Prasad, C. Dovrolis, M. Murray, and K. Claffy. Bandwidth estimation: metrics, measurement techniques, and tools. *Network, IEEE*, 17(6), 2003.

[21] F. C. R. Spieksma. On the approximability of an interval scheduling problem. *J. Scheduling*, 2:215–227, 1999.

[22] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee. How to model an internetwork. In *INFOCOM*, 1996.

Reuven Cohen received the B.Sc., M.Sc. and Ph.D. degrees in Computer Science from the Technion - Israel Institute of Technology, completing his Ph.D. studies in 1991. From 1991 to 1993, he was with the IBM T.J. Watson Research Center, working on protocols for high speed networks. Since 1993, he has been a professor in the Department of Computer Science at the Technion. He has also been a consultant for numerous companies, mainly in the context of protocols and architectures for broadband access networks. Reuven Cohen has served as an editor of the IEEE/ACM Transactions on Networking and the ACM/Kluwer Journal on Wireless Networks (WINET). He was the co-chair of the technical program committee of Infocom 2010 and headed the Israeli chapter of the IEEE Communications Society from 2002 to 2010.

Gleb Polevoy received the B.A. (Summa Cum Laude) in Mathematics and Computer Science and M.Sc. in Computer Science from the Technion - Israel Institute of Technology, Haifa, Israel, in 2004 and 2011, respectively. From 2004–2010 he also worked as a software engineer and from 2011–2012 he worked as a researcher in the Technion. Since 2012, he is a Ph.D. student in the EEMCS Department at Delft University of Technology, working on Multi-Agent systems and game theory.