Competitive Analysis for Online Scheduling in Software-Defined Optical WAN

Su Jia*, Xin Jin†, Golnaz Ghasemiesfeh‡, Jiaxin Ding‡, Jie Gao‡

*Department of Applied Math and Statistics, Stony Brook University. su.jia@stonybrook.edu
†Department of Computer Science, Johns Hopkins University. xinjin@cs.jhu.edu
‡Department of Computer Science, Stony Brook University. {gghasemiesfe, jiading, jgao}@cs.stonybrook.edu

***Abstract*—Modern planetary-scale online services have massive data to transfer over the wide area network (WAN). Due to the tremendous cost of building WANs and the stringent timing requirement of distributed applications, it is critical for network operators to make efficient use of network resources to optimize data transfers. By leveraging software-defined networking (SDN) and reconfigurable optical devices, recent solutions design centralized systems to jointly control the network layer and the optical layer. While these solutions show it is promising to significantly reduce data transfer times by centralized cross-layer control, they do not have any theoretical guarantees on the proposed algorithms. This paper presents approximation algorithms and theoretical analysis for the online transfer scheduling problem over optical WANs. The goal of the scheduling problem is to minimize the makespan (the time to finish all transfers) or the total sum of completion times. We design and analyze various greedy, online scheduling algorithms that can achieve 3-competitive ratio for makespan, 2-competitive ratio for minimum sum completion time for jobs of unit size, and 3α-competitive ratio for jobs of arbitrary transfer size and each node having degree constraint d, where $\alpha = 1$ when $d = 1$ and $\alpha = 1.86$ when $d \geq 2$. We also evaluated the performance of these algorithms and compared the performance with prior heuristics.**

I. Introduction

Modern plenary-scale services, such as search, social networking and e-commerce, have massive amounts of data to transfer over the wide area network (WAN). The quality of these services heavily depend on how fast data can be delivered to destinations. On the other hand, it is extremely expensive to build and maintain WANs. Therefore, it is critical for network operators to make the most efficient use of WAN bandwidth, in order to accommodate as much traffic as possible and finish data transfers as quickly as possible.

Traditional traffic management solutions suffer from inefficiencies of distributed protocols that run in the network. Routers do not have a global view of network topology and traffic demand, and cannot make good network-wide routing decisions. Advancements in software-defined networking (SDN) enable network operators to build centralized control systems. These systems can obtain global topology and traffic information, and configure all routers in the network in a centralized manner. For example, Google B4 [15] and Microsoft SWAN [14] show that such centralized control can significantly improve network utilization.

For optical networks, today's optical technologies allow dynamic reconfiguration of optical devices, similar to router reconfiguration in SDN. In a modern WAN, routers are not directly connected by point-to-point links. There is an optical layer under the network layer, which consists of optical devices, such as Reconfigurable Optical Add-Drop Multiplexers (ROADMs), and fibers. Routers are connected to the optical devices; a link between two routers in the network layer is actually an optical circuit that traverses multiple optical devices in the optical layer. By reconfiguring the optical devices in the optical layer, we are able to change how the routers are connected in the network layer, i.e., the network-layer topology. In this way, we can not only change routing by reconfiguring routers, but also change network-layer topology by reconfiguring optical devices. The constraints in such reconfiguration are fixed upper bounds on the degree of each node of the network, i.e., the maximum number of optical links incident to each node. This is because each router has a limited number of router ports (e.g., 64 ports), which limits the number of links a router can have. Recent work such as Owan [16] shows that we can dramatically improve data transfers by jointly controlling the network routers and optical devices. However, the solution in Owan [16] used heuristic algorithms for scheduling and reconfiguration of the optical devices, which do not provide any mathematical analysis or theoretical guarantees.

This paper presents approximation algorithms and theoretical analysis on the data transfer problem on the WAN. Similar to previous work such as Owan, we assume a centralized controller that can dynamically reconfigure both the network routers in the network layer and the optical devices in the optical layer. We have a stream of transfer requests arriving at the system, where each request has a source, a destination, a transfer size, and a release time (the earliest time by which the transfer can start). We need to design a scheduling algorithm that decides, at each time slot, which jobs to transfer and which links to schedule. All these decisions need to respect the degree bounds from the optical layer. The goal of the optimization problem is to minimize the *makespan* (the total time to finish all the transfers) or the total *sum of completion time*. The problem has an offline version and an online version. In the offline version, the system knows all the transfer requests ahead of the time; in the online version, the system only receives the transfer requests as they show up at their release time. We are mainly interested in the online setting though we

also prove approximation ratios for the offline setting.

In the previous work [16] a variety of heuristic algorithms have been adopted such as Shortest Job First, Earliest Deadline First (when deadlines are enforced), as well as more sophisticated heuristics using simulated annealing. In this paper our goal is to provide algorithms with theoretical guarantees yet we also wish to use algorithms that are simple and practical to implement. The algorithms we design are the following, all with a greedy nature and easy to implement.

- *Greedy Scheduling:* Take the currently available jobs in any arbitrary order and schedule a job whenever the degree constraint is not violated at its source and destination. We prove this simple algorithm is 3-competitive in the online setting for minimizing makespan.

  Further, for minimizing the sum of completion time, this algorithm gives a 2-approximation if all jobs arrive at time 0 and have unit size; and is 3-competitive when jobs have unit size and arrive in an online manner.
- *Perfect Matching based Scheduling:* If the optical links is directional (i.e., a link from i to j is only for data transfer from i to j but not in the other direction), one can formulate the problem as scheduling in a bipartite graph. In this case we can schedule the requests by choosing a perfect matching from the current set of jobs, which always reduce the 'heaviest bottleneck' in the requests. We prove that this algorithm is 2-competitive for minimizing makespan.
- *Smith's Greedy Scheduling:* When the jobs have varying size and when we optimize for the sum of completion time, we augment the simple greedy algorithm by first sorting the jobs in non-decreasing size. When the jobs all arrive at time 0, this algorithm is 2-competitive.
- *SRPT-based Greedy Scheduling:* In the most general setting, when jobs have varying size and may arrive in an online manner, we propose to use the Shortest Remaining Processing Time (SRPT) to sort the jobs and propose a new greedy algorithm that is 3α-competitive for minimizing the sum of completion time, where $\alpha = 1$ when $d = 1$ and $\alpha = 1.86$ when $d \geq 2$..

In addition, we also show in a variety of lower bounds on the competitive ratios for both offline and online settings.

To summarize, this paper provides the first theoretical analysis of the data transfer problem in a reconfigurable optical WAN. This problem, as shown in the next section, is related to a variety of scheduling problems in the literature yet is distinctly different due to the online nature and the maximum degree constraints. We complement the theoretical analysis by providing an extensive set of simulation results that evaluate the performance of these algorithms.

II. Related Work

SDN Traffic Engineering. SDN decouples the control plane from the data plane. Network operators can leverage SDN to build centralized control systems that overcome many drawbacks of traditional distributed solutions. Several SDN-based systems have been designed, implemented, and deployed in recent years that can improve network throughput [14], [15], allocate capacity based on service priority and the incremental value of additional allocation [18], tolerate data plane and control plane failures [20], enforce policy-based routing [13], and jointly manage routers, proxies, load balancers, and DNS servers [21]. Besides these, there is a growing interest to go beyond network-level objectives such as network throughput and focus on fine-grained transfer-level objectives such as transfer completion time. Recent solutions have shown that by leveraging SDN we can significantly reduce transfer completion time [5], [17], [19], [24], [27]. Owan goes even another layer down the stack to the optical layer, and shows how to jointly control network routers and optical devices to reduce transfer completion time [16]. However, as we have pointed out, Owan does not have any mathematical analysis and theoretical guarantees on the proposed algorithms.

Scheduling Algorithms. Scheduling is a well-studied class of problems. The scheduling problem with a set of dependent jobs on identical machines is known to be NP-hard even when the jobs have unit length [3]. Most of the works in literature have considered the problem where jobs are *independent* of each other. Both the preemptive/non-preemptive and offline/online versions on single/multi machines have been extensively studied.

There is a family of scheduling problems called scheduling with conflicts (called **non-clairvoyant scheduling**) [23], in which jobs could be in conflict with each other and no two jobs in conflict can be scheduled at the same time. One special case is the online graph coloring problem, we are asked to color the vertices such that no two adjacent vertices are given the same color. The vertices are revealed over time and we are asked to color them when they show up. In our problem, two jobs that share the same source or destination are also in conflict and cannot be scheduled at the same time. Our problem studies a special case of this problem, where the conflicts are determined by the source/destination of the jobs and instead of arbitrary. This makes many of the results for 'scheduling with conflicts' (especially the lower bounds) inapplicable in our setting.

The relationship between chromatic sum problem and the scheduling problem with the objective of minimizing average completion time has been explained thoroughly in [12], [22]. We refer the readers to Gandhi et al.([9]) for a summary of the results on the offline version both with or without preemptivenes, and Even et al.([8]) for minimizing makespan. More specific results for different constraints on the graph and jobs lengths could be found in [10], [1], [4]. In short, offline problems were well studied but no much has been done for the online problem.

III. Problem Statement

Given a set of nodes V, in which each node represents a site on the WAN, we compute network-layer topology and transfer schedules to optimize data delivery. Each node v_j has a maximum degree d_j (the number of router ports). A data

transfer request is denoted by the tuple (u_i, v_i, ℓ_i, r_i), in which the request i has source u_i, destination v_i, size ℓ_i and release time r_i. Without loss of generality, we assume that all links have unit capacity and all job sizes are integers, as we may always adjust the scale of a time slot. We assume that all transfers are scheduled by single-hop paths from source to destination. The challenge is to decide which edges to use (with respect to the degree constraints) and which set of data transfers to schedule on these edges. Specifically, we have the following two problems with different optimization objectives.

Problem 1 (Minimum Makespan). *Schedule the transfers such that the* maximum *completion time of all transfers is minimized.*

Problem 2 (Minimum Sum Completion Time). *Schedule the transfers such that the* sum *of the completion time of all transfers is minimized.*

For each problems, we focus on the *online* version in which transfer requests can arrive at different time slots. We aim for *competitive* algorithms of which performance is compared to the optimal offline version.

We model the transfer requests as a (multi-)graph H on the nodes V, in which each edge represents a request (u_i, v_i, ℓ_i, r_i). H is called the transfer request graph. The optical links are generally bidirectional. But we sometimes consider the special case when the links are *directional*. An undirected link, once placed, may be used both ways to transfer data. And the degree bound d_i for node i is the total number of undirected links incident to a node. In the directional setting, each link from i to j is only for data transfer from i to j, not in the opposite direction. One may formulate a bipartite graph — each node i corresponds to two nodes i and i' with $i \in V$ and $i' \in V'$. Similarly we can define a (multi-) bipartite graph H on $V \times V'$ in which each edge represents a job request and the degree bound d_i applies for the maximum possible outgoing degree for nodes in V and maximum incoming degree for nodes in V'. Sometimes we can obtain better approximation results in this special case.

IV. Minimizing Makespan

In this section, we focus on the objective of makespan. First, we consider the general case where the links are undirectional, the degree d_i of nodes in V could be different for different nodes and jobs could have different sizes. Next, we present results (with better approximation factor) for the special case where the links are directional (i.e., in a bipartite graph) and all nodes have the same degree constraints.

A. Algorithms and Upper Bounds

In the offline setting (when all jobs are available at time zero), the best approximation is 2 (by a greedy algorithm) and a lower bound of $4/3$ is shown in [7]. Here we study the online version.

a) Upper Bound for Online Non-Preemptive Setting:

Definition 4.1 (Greedy Scheduling). *At any time slot t we have a collection of available jobs represented by edges in $G(t)$. We go through this list of jobs in any arbitrary order and schedule the jobs if the degree constraints are not violated.*

Theorem 4.2. *The greedy algorithm is* 3*-competitive.*

Proof: For each job j from u to v we denote by r_j its arrival time or release time, T_j the time when it is scheduled in the greedy algorithm, and T_j^* when it is scheduled in the optimal offline algorithm. We also denote by T the makespan of our algorithm and T^* the makespan of the optimal offline solution. Obviously $T^* \geq T_j^* \geq r_j + \ell_j$.

By the greedy nature of the algorithm, for *all* the time slots after r_j (the release time of job j with source u and destination v), either all ports at u were used up (for jobs in set $N(u)$) or all ports at node v are used up (for jobs in set $N(v)$). Now we may upper bound the finishing time for job j to be $T_j \leq r_j + \sum_{i \in N(u)} \ell_i/d_u + \sum_{i \in N(v)} \ell_i/d_v + \ell_j$

On the other hand, we know that job j and the jobs in $N(u)$ share the same vertex u and thus the optimal solution has to use at least $(\sum_{i \in N(u)} \ell_i + \ell_j)/d_u$ slots to schedule them. This means $T^* \geq (\sum_{i \in N(u)} \ell_i + \ell_j)/d_u$. Similarly, $T^* \geq (\sum_{i \in N(v)} \ell_i + \ell_j)/d_v$. Put together we know $T_j \leq 3T^*$ for any j. This means the algorithm is 3-competitive. □

b) Special Case: Bipartite Graph: Here we assume that the job request graph H of nodes is a bipartite graph in which transfer jobs are from vertices in V to vertices in V'. Here we show that one can use a different algorithm for the bipartite graph when $d_i = d$ for all i and all jobs have size of 1 (the capacity of a link).

Definition 4.3 (Perfect Matching Scheduling). *At any time slot t we have a collection of available jobs represented by edges in a bipartite graph $H(t)$. We first add dummy edges to transform H into a k-regular graph. Any regular bipartite graph has a perfect matching. Remove this perfect matching and we obtain a $(k-1)$-regular graph. We iterate and obtain d perfect matchings for the next slot.*

This above algorithm is optimal when all jobs are available at time 0 and is 2-competitive in general.

If the largest degree in the transfer request graph H is k, we would need at least $\lceil k/d \rceil$ time slots to schedule all transfers by any algorithm. In the offline setting, the k perfect matchings are put into $\lceil k/d \rceil$ groups. Each group with at most d matchings. The ith group is scheduled in the ith time slot. Thus all requests are done in $\lceil k/d \rceil$ slots.

When jobs arrive in an online manner, at any time t we use the above idea to select a perfect matching (again dummy edges are added to make the graph regular). We now argue that this algorithm is 2-competitive.

To see that, suppose the last arriving jobs (i.e., the highest release time) arrive at time t. Also suppose at time t the job request graph (not including the jobs that arrive at time t) is $H^*(t)$ if we have used the optimal (offline) algorithm and

$H(t)$ if we have used the perfect matching algorithm. A node u in $H^*(t)$ has degree $\deg^*(u)$ and in $H(t)$ has degree $\deg(u)$. Clearly we have $\deg(u) \leq \deg^*(u) + td$ – in the worst case our algorithm does not schedule any jobs incident to u while the optimal algorithm always schedule d job at u at every one of the t slots since time 0.

Further, the newly arriving jobs at time t have degree $d'(u)$ for node u. Thus the makespan of the optimal algorithm T^* will be $T^* \geq t + [\deg^*(u) + d'(u)]/d$. For our algorithm we know that perfect matching is optimal after time t. Thus our algorithm can finish in time $T \leq t + \max_u[\deg(u) + d'(u)]/d \leq 2t + \max_u[\deg^*(u) + d'(u)]/d \leq 2T^*$.

Summarizing the above, we have

Theorem 4.4. *For a bipartite request graph H in which all nodes have degree bounds d, the perfect matching based scheduling algorithm is optimal when the jobs are available at time 0, and is 2-competitive in the online setting.*

B. Lower Bound

We present lower bound examples on the greedy scheduling and perfect matching based scheduling in both the offline and online settings. In our examples the requests are represented by a bipartite graph where jobs have sources in V and destinations in V'. We denote $V = \{s_1, s_2, \cdots, s_n\}$, $V' = \{s'_1, s'_2, \cdots, s'_n\}$. All nodes have the same degree constraint of $d = 1$. Each edge represents a job size 1. Figure 1 (i) and 1(ii) show that greedy scheduling and perfect matching based scheduling can be a factor 2 off from the optimal algorithm in the online setting, for minimizing makespan. This shows that our analysis of the perfect matching algorithm in Theorem 4.4 is tight.

When all jobs are available at time zero, Figure 1 (iii) shows that greedy scheduling can be a factor 1.5 off from the optimal.

V. Minimizing Sum Completion Time

Now we study the problem of minimizing the sum completion time. This section is partitioned into parts, focusing on different variants.

(A) Jobs have unit size and release time is zero.
(B) Jobs have unit size and arbitrary release time.
(C) Jobs can have arbitrary size but release time is zero.
(D) Jobs can have arbitrary size and release time.

We analyze three different greedy algorithms. The simple greedy algorithm in Definition 4.1 is 2-competitive for (A) and is 3-competitive for (B). For (C) and (D) we propose two slightly different greedy algorithms that both achieve competitive ratio of 2, but the analysis for (D) only holds when all degree constraints are 1.

Throughout the section we use d_v as the degree constraint of v, while $\deg(v)$ as the number of edges/jobs requests incident to v in job request graph H.

A. Jobs of Unit Size and Release Time Zero

We will show that greedy algorithm in Definition 4.1 gives a 2-approximation for scenario (A) with arbitrary degree constraints d_u. When all the degree constraints $d_u = 1$, this is in fact exactly the *edge chromatic sum problem*.

Definition 5.1 (Minimum Edge Chromatic Sum Problem). *Given a multi-graph H, partition its edges into matchings $\{M_t\}$, such that $\sum_t t \cdot |M_t|$ is minimized.*

The edges in the matching M_t are colored t, with a cost of t. In scheduling, this means that we schedule M_t in the t-th time slot and all the jobs in M_t have completion time of t. A related problem is the minimum vertex chromatic sum problem:

Definition 5.2 (Minimum Vertex Chromatic Sum Problem). *Given a graph G, partition its vertices into independent sets $\{I_t\}$, such that $\sum_t t \cdot |I_t|$ is minimized.*

For a graph H, define its *line graph* $L(H)$ as following: every edge e of H corresponds to a vertex v_e of $L(H)$, add an edge between v_e and $v_{e'}$ if e, e' share a common vertex in H. Clearly for any graph H, the edge chromatic sum problem is equivalent to the vertex chromatic sum problem in $G = L(H)$.

Given an edge/vertex coloring scheme, we say it is *compact*, if it is locally optimal, i.e. we can not move any edge/vertex to an matching/independent set with smaller index to be a feasible coloring.

The Minimum Edge Chromatic Sum problem is NP-hard, even when H is a bipartite graph [11]. But any compact edge-coloring is a 2-approximation [2]. In this work, we will show that the same approximation ratio can be obtained when the degree constraints d_v are arbitrary.

Let H be a job request graph and G be the line graph of H. Let OPT be the optimum for the minimum sum completion time problem for version (A) with degree constraints $\{d_u\}_{u \in V(H)}$. First we have a lower bound for OPT.

Lemma 5.3 (Lower Bounds on OPT). *$OPT \geq \frac{1}{2}(n + \frac{1}{2}\sum_{u \in V(H)} \deg(u)^2/d_u)$, where $n = |V(G)|$ and $\deg(u)$ is the degree of u in H.*

Proof: First we define the *clique labelling problem* as follows: given a complete graph Q with q nodes, and an integer d, we wish to color the nodes of Q such that each color is used at most d times, and minimize the total cost, assuming color i has cost i. We denote the optimum as $CL(Q)$. Clearly, we will have d vertices of color 1, another d vertices of color 2 until we finish with all vertices. That is, $CL(Q) = \sum_{j=1}^{\lfloor q/d \rfloor} j \cdot d + (q - d\lfloor q/d \rfloor)(d+1) \geq (q + q^2/d)/2$.

On the other hand, observe that each vertex u in H corresponds to a clique Q_u in G, containing vertices corresponding to edges incident to u in H. Any edge coloring of H – in particular, the optimal edge coloring of H – can be extended to a valid clique labeling for the cliques $\{Q_u\}$: the vertex in Q_u carries the color of its corresponding edge in H; by definition of edge coloring at most d_u edges incident to vertex u can have the same color. Note that each edge appears in exactly two cliques, so $OPT \geq \frac{1}{2}\sum_{u \in V(H)} CL(Q_u)$,

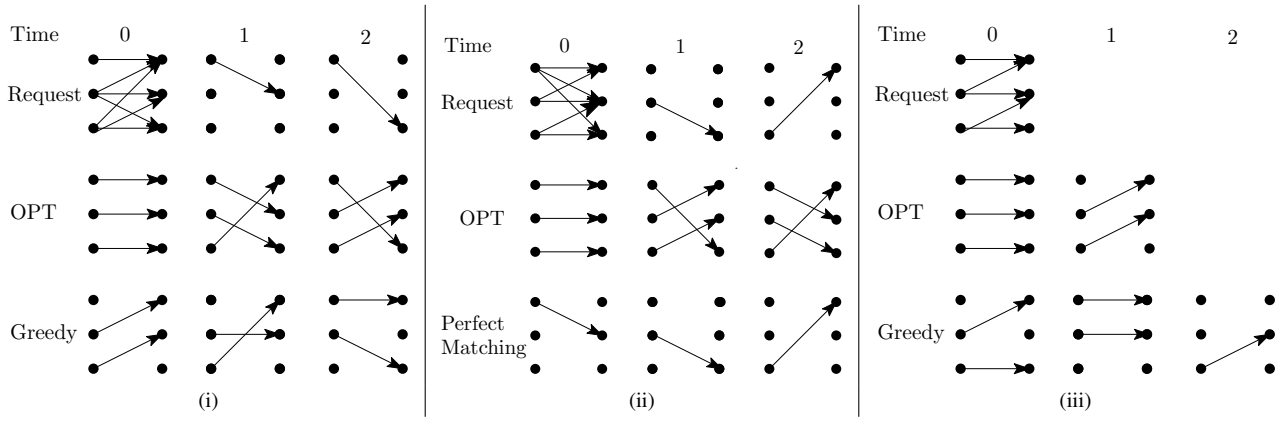


Fig. 1. (i): The lower bound for greedy scheduling algorithm in the online setting. Every node in V except node 1 is connected to all the nodes in V'. For node s_1, there is a single request (s_1, s'_1). At time i there is a request (s_1, s'_{i+1}). The optimal solution is shown in the second line, in which we can schedule n jobs in each slot and the completion time is n. While greedy scheduling (in the worst case) could have a situation such that all requests from s_1 are in conflict with other requests from time 0 to $n-2$. Thereafter, at time $n-1$, we are left with n jobs from source s_1 which requires another n slots. The completion time is $2n-1$. Thus the greedy algorithm can be a factor $2-\frac{1}{n}$ off from the optimal. (ii): The lower bound for perfect matching based scheduling algorithm in the online setting. The requests at time 0 include all the edges except the set $\{(s_j, s'_{(j+1) \mod n})|2 \le j \le n\}$. For the time i $(1 \le i \le n-1)$, there is a request $(s_i, s'_{(i+1) \mod n})$. In the optimal solution, we may schedule a perfect matching in each slot $j = 0, 1, \cdots, n-1$ as shown in the figure, which does not use any edges in the edge set $M = \{(s_j, s'_{(j+1) \mod n})|1 \le j \le n\}$ and at time $n-1$ finish M. The completion time is n. For perfect matching based algorithm, for time slot i from 0 to $n-1$, we may only end up scheduling $(s_{i+1}, s'_{(i+2) \mod n})$ in the worst case. At time n each node in V has degree of $n-1$. Therefore, the completion time is $2n-1$. (iii): The lower bound for greedy algorithm in the offline setting. For the optimal solution, two slots are used to deliver the requests; for greedy matching, three slots were used. This pattern can be repeated to show that the greedy matching can be as bad as 1.5 times the optimal makespan.

Recall that $CL(Q_u) \ge \frac{1}{2}(\deg(u) + \deg(u)^2/d_u)$. Hence, $OPT \ge \frac{1}{4}\sum_{u \in V(H)}(\deg(u) + \deg(u)^2/d_u) = \frac{1}{2}(n + \frac{1}{2}\sum_{u \in V(H)} \deg(u)^2/d_u)$ as desired. □

Lemma 5.4 (Upper Bound). *The total cost of the greedy algorithm is at most* $n + \frac{1}{2}\sum_{u \in V(H)} \deg(u)(\deg(u)-1)/d_u$.

Proof: By the definition of the greedy algorithm, when we assign color t to a job/edge $j = (u, v)$ in H, job j cannot be colored by a smaller color, i.e., in each of the slots $\tau \le t-1$, either d_u jobs incident to u are scheduled at time slot τ, or d_v jobs incident to v are scheduled at time slot τ. In the first case, we charge $1/d_u$ unit to each of those d_u edges. In the second case, we charge $1/d_v$ unit to each of the d_v edges. We also charge 1 unit on edge j itself. Clearly, the total amount of charge is exactly the cost of our greedy coloring.

Now we count the total charge in a different way. First all edges in H are charged 1 unit each. So this part of charge is $n = |V(G)|$. Now we look at the fractional charges.

The charge on an edge $j = (u, v) \in H$ is at most $1/d_u$ by each edge incident to u with colors higher than j's color *and* $1/d_v$ unit by each edge incident to v with colors higher than j's color. Now if we just look at node u and its $\deg(u)$ edges incident to u in H. We can rank them by their color from highest to lowest. Each color in this neighborhood will charge to $1/d_u$ to all the lower colors. The total charge collected on all these $\deg(u)$ edges in the neighborhood of u becomes at most $\sum_{i=1}^{\deg(u)-1} i/d_u = \frac{1}{2}\deg(u)(\deg(u)-1)/d_u$. Summing up over all vertices of H we are done. □

Theorem 5.5. *The greedy scheduling algorithm is a 2-approximation of problem (A) with arbitrary degree constraints.*

B. Jobs of Unit Size and Arbitrary Release Time

We will show that the same greedy algorithm in Definition 4.1 gives a 3-approximation for problem version (B). For simplicity, we will assume $d_u = 1$ for all $u \in V(H)$, but this result applies to general degree constraints.

Again consider the request graph H and its line graph G. Denote by $r(v)$ the release time of a vertex v in G and $x(v)$ the time slot scheduled for job v. The schedule of the requests can be modeled as using color $x(v)$ for v such that $x(v) \ge r(v)$ for any vertex v and $x(v) \ne x(w)$ if u, w are neighbors. Therefore, a job is an edge in H and a vertex in G. Its color is the time slot this job is scheduled for.

The greedy algorithm in Definition 4.1 is an online algorithm and gives a schedule/coloring scheme that is *compact*, i.e., no job can be moved to an earlier slot. We now upper bound the chromatic sum (i.e., sum of completion time).

Lemma 5.6. *Given G with n vertices and m edges, any compact vertex coloring χ satisfies $|\chi| \le m + \sum_v r(v)$, where $|\chi|$ is the vertex chromatic sum and $r(v)$ is the release time of v.*

Proof: Consider the following auxiliary graph G': for each v, add $r(v)-1$ extra dummy nodes as well as dummy edges among them to form a clique of size $r(v)$. See Figure 2. The edges in G' can be classified into three types: Type 1 includes the edges in the original graph G; Type 2 are the edges incident to exactly one dummy node; Type 3 are the edges incident to two dummy nodes. Denote by E_i the set of type i edges.

Any compact coloring of G with the release time can be completed as a compact coloring of G' – we simply give color $1, 2, \cdots, r(v)-1$ for the $r(v)-1$ dummy nodes adjacent to v. Now we use a charging scheme. When we color a node v with color j, there must be $j-1$ neighbors of it, each

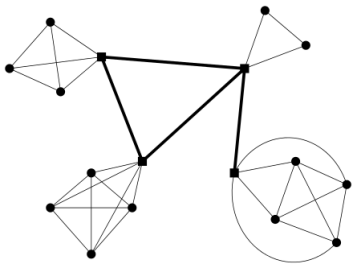


Fig. 2. An example of the dummy nodes/edges. The squares are the vertices in G with release time $4, 5, 3, 5$ respectively.

colored $1, 2, ..., j-1$ respectively. Charge 1 unit to each of these $j-1$ edges, and charge 1 unit to v itself. Note that only type 1 and type 2 edges may be charged. Hence, at the end of the algorithm, each node is charged exactly 1, and each type 1 and type 2 edge is charged at most 1. So the total charge is at most $|E_1| + |E_2| + n$. Since $|E_1| = m$ and $|E_2| = \sum_v [r(v) - 1] = \sum_v r(v) - n$, we are done. □

Lemma 5.7. *Let $G = (V, E)$ be a line graph, in which each vertex v has release time $r(v)$. Then the chromatic sum of a feasible coloring of G is no smaller than $\frac{1}{2}\lambda(m + 2n) + \frac{1}{2}(1 - \lambda)\sum r(v)$ for any $\lambda \in [0, 1]$, where $n = |V|$ and $m = |E|$.*

Proof: We make use of the property of line graph that it can be partitioned into cliques $Q_1, ..., Q_k$, s.t. each node is contained in at most two cliques, and each edge is contained in exactly one clique.

Now we consider the optimal clique labeling $CL(Q_i)$ of each clique Q_i. We have two natural lower bounds of $CL(Q_i)$:

- $CL(Q_i) \geq \binom{q_i+1}{2}$, where $q_i = |Q_i|$;
- $CL(Q_i) \geq \sum_{v \in Q_i} r(v)$.

There the linear combination of the two lower bounds is still a lower bound. $CL(Q_i) \geq \lambda\binom{q_i+1}{2} + (1-\lambda)\sum_{v \in Q_i} r(v)$, where $0 \leq \lambda \leq 1$. Summing over all cliques, we obtain

$$\begin{aligned} CL(G) &\geq (1-\lambda)\sum_{i=1}^{k}\sum_{v \in Q_i} r(v) + \lambda\sum_{i=1}^{k}\binom{q_i+1}{2} \\ &= \lambda(m + 2n) + 2(1-\lambda)\sum_{v \in V} r(v) \end{aligned}$$

On the other hand, any feasible coloring of G can be turned into a clique labeling such that the label of each vertex is exactly its color. Since each vertex belongs to at most two cliques. The chromatic sum of the optimal coloring solution is at least as big as $CL(G)/2$, since $CL(G)$ is the optimal clique labeling. □

Theorem 5.8. *The chromatic sum of any compact coloring is a 3-approximation to problem (B).*

Proof: Denote $r = \sum r(v)$. By Lemma 5.6 and 5.7

$$\frac{|\chi|}{OPT} \leq 2\frac{m + n + r}{\lambda(m + 2n) + 2(1-\lambda)r} < 2\frac{1}{\lambda\frac{1}{1+\alpha} + (1-\lambda)\frac{2}{1+\frac{1}{\alpha}}},$$

where $\alpha = \frac{r}{m+2n}$.

If $\alpha \leq 1/2$, we take $\lambda = 1$ and then $|\chi| \leq 2(1 + \alpha)OPT$. If $\alpha > 1/2$, we choose $\lambda = 0$ and then $|\chi| \leq (1 + \frac{1}{\alpha})OPT$. Either way the approximation ratio is no greater than 3. □

C. Jobs of Arbitrary Size and Release Time Zero

We use a specific greedy algorithm called the *Smith's Greedy* because its intuition comes from the well-known Smith's Rule in single machine scheduling problem [26].

Definition 5.9 (Smith's Greedy Algorithm). *We sort the jobs in non-decreasing size and schedule the jobs in a greedy manner with respect to the degree constraints. We allow preemptiveness.*

We will show that Smith's Greedy algorithm gives a 2-approximation on the sum completion time for problem (C). Denote by J the set of all jobs. We first sort and relabel all jobs/edges so that $\{l_j\}$ is non-decreasing. Consider a job j incident to a node u (which could be either a source or a destination), let $\text{rank}_u^+(j)$ ($\text{rank}_u^-(j)$) be the rank of j's completion time among all jobs incident to u, in increasing (decreasing) order.

Lemma 5.10. *The sum of completion time of the Smith's greedy algorithm is at most*

$$\sum_{j \in J}(\text{rank}^-_{src(j)}(j) + \text{rank}^-_{des(j)}(j) - 1) \cdot \ell_j.$$

Proof: Let G be the line graph of H. Build an auxiliary graph G' as follows: associate each node j in G with a clique Q_j of size l_j in G'; If $(i, j) \in E(G)$, add edges in G' between every pair of nodes (x, y) where $x \in Q_i$ and $y \in Q_j$. Consider the following charging scheme: when we color node j in G with color c, we first pick an arbitrary node u in Q_j and charge 1 to it. Then, since colors $1, 2, ..., c-1$ are already occupied in u's neighborhood in G, say by nodes $z_1, ..., z_{c-1}$, we charge 1 for each edge (u, z_k). If z_i is not in Q_j (i.e. in another clique Q_r), then we call it type I charge of j, otherwise call it type II. Then the total charge of type I for j is at most

$$\sum_{i \sim \text{src}(j), i<j} l_i + \sum_{k \sim \text{des}(j), k<j} l_k.$$

Summing over j, the total type I charge is at most

$$\begin{aligned} &\sum_j [\sum_{i \sim \text{src}(j), i<j} l_i + \sum_{k \sim \text{des}(j), k<j} l_k] \\ &\leq \sum_j [(\text{rank}^-_{\text{src}(j)} - 1) + (\text{rank}^-_{\text{des}(j)} - 1)] l_j. \end{aligned}$$

The inequality is true because each term l_j appears $(\text{rank}^-_{\text{src}(j)} - 1) + (\text{rank}^-_{\text{des}(j)} - 1)]$ times in the summation.

On the other hand, it is clear that the type II charge on each j is ℓ_j, hence the total type II charge is $\sum_j \ell_j$. Combining the upper bounds on type I and type II charges, we are done. □

By mimicking Lemma 5.3, we have

Lemma 5.11 (Lower Bound on OPT).

$$OPT \geq \frac{1}{2}(\sum_{j \in J} \text{rank}^-_{\text{src}(j)}(j) \cdot \ell_j + \sum_{j \in J} \text{rank}^-_{\text{des}(j)}(j) \cdot \ell_j)$$

Theorem 5.12. *The Smith's Greedy algorithm gives a 2-approximation for problem (C).*

D. Jobs of Arbitrary Size and Arbitrary Release Time

This is the most general setting and the methods we used before do not work for (D). In particular, the jobs of smaller size should be given higher priority in order to reduce the sum of completion time. We show an online algorithm incorporating this idea with a more complicated analysis.

Let $H = (V; E)$ be the job request graph. We assume that d_v are the same for all v in this section. Our algorithm will use the SRPT (Shortest Remaining Processing Time) algorithm as a subroutine, which is optimal for online scheduling in a single machine for minimizing the average completion time [25]. For scheduling multiple machines (an NP-hard problem), SRPT achieves an approximation factor of 1.86 which is the best approximation factor known so far for this problem [6]. In this paper we define $\alpha = 1.86$ if $d \geq 2$ and $\alpha = 1$ if $d = 1$.

Definition 5.13 (SRPT for d-machine scheduling). *At each time slot t, among all jobs that are alive (i.e. those already arrived but have not yet been completed), choose the d jobs with the smallest remaining processing time, and arbitrarily assign them to the d machines to schedule.*

Notice that the SRPT algorithm is preemptive – a job might be temporally held if a new job with smaller size arrives. We first explain how to schedule the jobs in an offline setting. Define $J(v)$ the list of data transfer requests that either starts from or ends at node v. Let us start with $d = 1$. As preprocessing, we perform SRPT scheduling on $J(v)$ for each node v and keep an SRPT list $L(v)$, indexed by time slot. The t-th position is marked j if it is used to process job j, and called a *dummy unit* if no job is processed in this time slot. A job of size ℓ will need ℓ time slots, called *job units* in $L(v)$. It can be understood as chopping this job into ℓ data blocks each taking exactly one slot to finish.

Note that each job $j = (v, w)$ should appear in exactly two lists, i.e. $L(v)$ and $L(w)$, with ℓ_j job units in each, and $2\ell_j$ job units in total. The units of j in $L(v)$ are denoted by $u^v_{j,1}...u^v_{j,\ell_j}$. We will view these $2\ell_j$ job units as distinct. But each of the ℓ_j job units of j in $L(u)$ has a *twin* in the list $L(v)$, corresponding to the same piece of data blocks. For simplicity we only state the offline version for $d = 1$, see Fig 3 for illustration.

Definition 5.14 (Offline SRPT-based SDN Scheduling). *Given a request graph H, for each v, find and store the SRPT list for $J(v)$. Then, for each time slot t, we follow the steps below to select the jobs to process at t:*

1) *Sort Job Units: denote by $A = \cup_{v\in V} L(v)$. The job units in A are sorted into a list O as follows: we first collect the job units from the head of all the lists $L(v)$ (in an arbitrary order) into A. If a list is empty, skip it. Repeat this until all job units are collected. Note that the job units in each list $L(v)$ are going to appear in the same order as in A.*
2) *Choose Job Units: Find a maximal matching $M = M^t$ as follows: for each unit in O, if it does not create conflict with other edges chosen in M, then add it into M. (A dummy unit from $L(v)$ is considered a self-loop at v.) Note that at each time slot, we choose at most one unit from each list, either a real or a dummy one.*
3) *Update the lists: Suppose unit u from job $j = (v, w)$ is chosen, and w.l.o.g suppose it is from $L(v)$, if it is not a dummy unit, then we delete it from $L(v)$, and also delete its twin u' from $L(w)$; else just remove u itself.*

For a d-machine scheduling problem for J, let L_i be the SRPT schedule on the i^{th} machine, $i = 1...d$. Let $\mathcal{L}_i = \cup_v L_i(v)$, where $L_i(v)$ is the i^{th} job list for $J(v)$. We perform step (2) for each $\mathcal{L}_i$ separately and obtain d matchings at each t (one for each i), and schedule all of them. We first analyze the offline version below.

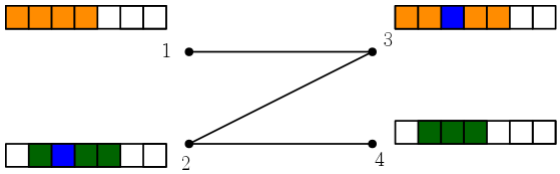


Fig. 3. A concrete example for the offline version. Suppose we have 4 jobs: $j_1 = (1, 3)$ (orange), $j_2 = (2, 3)$ (blue), $j_3 = (2, 4)$ (green), with release time $0, 2, 1$ and size $4, 1, 3$ respectively. The SRPT lists are shown in the figure, where the white units represent dummy units. Then our algorithm returns the following matchings: $M_1 = \{(1, 3)\}, M_2 = \{(1, 3), (2, 4)\}, M_3 = \{(1, 3), (2, 4)\}, M_4 = \{(1, 3), (2, 4)\}, M_5 = \{(2, 3)\}$.

For a job $j = (u, v)$, let $C^u(j)$ be the completion time of job j in the SRPT scheduling for d machines for $J(u)$. For a schedule σ for our SDN problem, define $C_\sigma(j)$ as the completion time of j.

Lemma 5.15. *If for some f, $C_\sigma(j) \leq f \cdot (C^u(j) + C^v(j))$ for any j, then σ is a $2f\alpha$-approximation.*

Proof: Let σ^* be an optimal schedule for our SDN problem. Let $\mathrm{SRPT}(u)$ denote the total completion time of the SRPT schedule for jobs incident to u and $\mathrm{TCT}(\sigma)$ to denote total completion time of a schedule σ for our SDN problem. First,

$$\begin{aligned}\mathrm{TCT}(\sigma) &\leq f\sum_{j\in J}(C^u(j) + C^v(j)) \\ &= f\sum_{u\in V}\sum_{j\sim u} C(j) = f\sum_{u\in V}\mathrm{SRPT}(u).\end{aligned}$$

On the others hand,

$$\mathrm{TCT}(\sigma^*) = \frac{1}{2}\sum_{u\in V}\sum_{j\sim u} C_{\sigma^*}(j) \geq \frac{1}{2\alpha}\sum_{u\in V}\mathrm{SRPT}(u)$$

. Combining these two inequalities and we are done. □

Given a job $j = (v, w)$ and a pair of twin units $u^v_{j,l}, u^w_{j,l}$ in $L(v)$ and $L(w)$ respectively, define $Z(j, l)$ to include the job units $u^v_{j,l}, u^w_{j,l}$ and those that are before $u^v_{j,l}, u^w_{j,l}$ in $L(v)$ and $L(w)$ respectively. Now we claim,

Lemma 5.16. *We will schedule at least one unit in $Z(j, l)$ in step (2) of our algorithm.*

Proof: If we have not selected any job unit in $Z(j, l)$, we have chosen no job incident to either v or w. Hence when we encounter either $u^v_{j,l}$ or $u^w_{j,l}$, we accept it (since it will not create conflict). □

Theorem 5.17. *For problem (D), the SRPT-based SDN scheduling gives a 2-factor when all degree constraints are one, and a 2α-factor when all degree constraints are the same.*

Proof: For a job $j = (v, w)$ with size l_j, clearly the units u^v_{j,l_j} and u^w_{j,l_j} are the $C^v(j)^{th}$ and $C^w(j)^{th}$ unit in $L(w)$ and $L(v)$ respectively. By Lemma 5.16, in at most $C^v(j)+C^w(j)$ time, one of u^w_{j,l_j} and u^v_{j,l_j} is scheduled (and hence its twin removed immediately), hence the completion time of j in our algorithm is at most $C^v(j)+C^w(j)$. Set $f = 1$ in Lemma 5.15 and we are done. □

Lemma 5.18. *Suppose for job $j = (v, w)$, $u^v_{j,i}$ and $u^w_{j,i}$ are the k^{th}, k'^{th} unit in the SRPT list for $J(v)$ and $J(w)$ respectively, w.l.o.g assume $k \le k'$. For any $t \ge k$, if $u^v_{j,l}$ is still not removed from $L(v)$ (hence $u^w_{j,l}$ is also in $L(w)$), then we will schedule at least one unit in $Z(j, i)$.*

Note that in the offline version, once we completed preprocessing, we never add new units to the lists. A major difference of the online version is, at each t, we add exactly one new unit, either real or dummy, to the tail of each $L(v)$. To be precise, if SRPT for $J(v)$ processes job j at time t, then we add a unit of j to the tail of $L(v)$.

Theorem 5.19. *For the online version of (D), there is a 3α-competitive algorithm.*

Proof: For a job $j = (v, w)$ with size l_j, the units u^v_{j,l_j} and u^w_{j,l_j} are the $C^v(j)^{th}$ and $C^w(j)^{th}$ unit in $L(w)$ and $L(v)$ respectively. By Lemma 5.18, in at most $C^v(j) + C^w(j) + \min\{C^v(j), C^w(j)\}$ time, one of u^w_{j,l_j} and u^v_{j,l_j} is scheduled, hence the completion time of j is at most $1.5(C^v(j)+C^w(j))$. Set $f = 1$ in Lemma 5.15 and we are done. □

E. Lower Bound

We take the same bipartite graph setting as in Section IV. Figure 4 provides a lower bound of 1.75 for the simple greedy scheduling in the online setting when the objective total completion time, when all job sizes are 1. Further, we also show a lower bound of 1.5 for *any* online scheduling algorithm for minimizing sum completion time in Figure 5.

VI. Evaluation

In this section, we compare the performance of three algorithms: (1) *Simple Greedy* which randomly finds a maximal matching at each time slot; (2) *Smith's Greedy* which first sorts all transfers according to their size, and then finds a maximal matching in this order at each time slot; (3) *SRPT-based Greedy* described in Algorithm 5.14. We use the following four metrics to evaluate the algorithms: makespan, 90-pct makespan (the time when 90 percent of transfers), average transfer completion time, and 90-pct transfer completion time. We use network topologies with different sizes, from 200 nodes to 2000 nodes. For traffic demand, we generate them using a wide variety of distributions. We denote $\mathrm{Exp}(2^p)$ the *truncated exponential distribution*: $P(X = 2^i) = 2^{-(i+1)}$ for $i = 0, ..., p-1$ and $P(X = 2^p) = 2^{-p}$. Denote $\mathrm{Poisson}(\mu, T)$

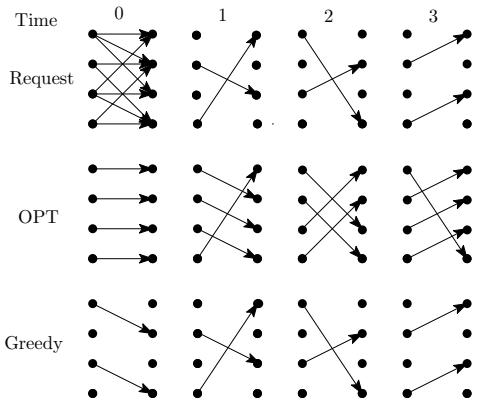


Fig. 4. Assume $|V| = |V'| = 2n$. At time 0, the requests consist of $\{(s_j, s'_{(j+2k) \mod (2n)})|1 \le j \le 2n, 1 \le k \le n\}$ and $\{(s_j, s'_{(j+1) \mod 2n})|j = 2k-1, 1 \le k \le n\}$. At time t from 1 to $2n-1$, if t is an odd number, the requests consist of $\{(s_{2j}, s'_{(2j+t) \mod 2n})|1 \le j \le n\}$; otherwise the requests consist of $\{(s_{2j-1}, s'_{(2j+t) \mod 2n})|1 \le j \le n\}$. In the optimal solution, at time $t \le 2n-1$, we schedule the edges $\{(s_j, s'_{(j+t) \mod (2n)})\}$. The sum completion time is $2n^2(2n+1)$. For greedy scheduling algorithm, in the worst case, if t is odd, we schedule $\{(s_{2j}, s'_{(2j+t) \mod 2n})|1 \le j \le n\}$; if t is even, we schedule $\{(s_{2j-1}, s'_{(2j+t) \mod 2n})|1 \le j \le n\}$. At time $2n$, each node has degree n. The average completion time is $n^2(7n+1)$. When $n \to \infty$, the competitive ratio is asymptotically 1.75.

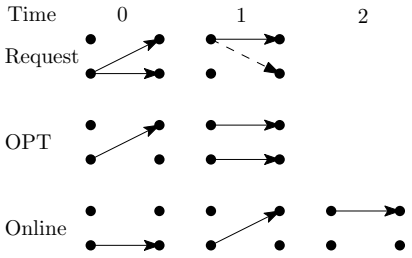
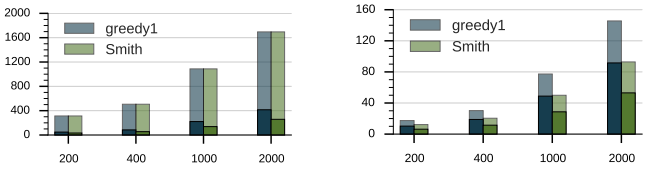
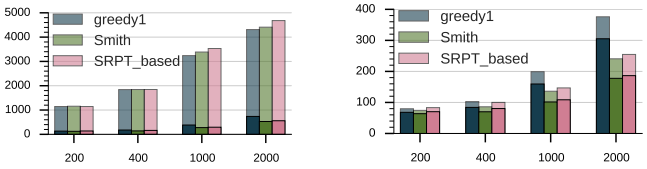


Fig. 5. The lower bound for any online algorithm to minimize sum completion time. There are two requests (s_2, s'_1) and (s_2, s'_2) at time 0. In the online algorithm if we schedule (s_2, s'_2) at the 0th slot, the request (s_1, s'_1) comes at time 1. Otherwise, the request (s_1, s'_2) comes at time 1. The optimal offline algorithm can always finish the requests in 2 time slots while any online algorithm must finish the requests in 3 time slots.



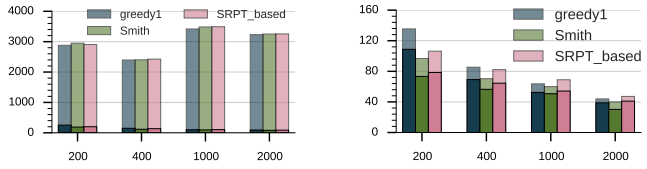
(a) Makespan. (b) Transfer completion time.

Fig. 6. Results of zero release time. Dark bars are 90-pct makespan & avg. completion time; light bars are full makespan & 90-oct completion time. The x-axis is network size, and the y-axis is normalized time. Fig. 7 and 8 have the same legends.



(a) Makespan. (b) Transfer completion time.

Fig. 7. Results of uniform release time.



(a) Makespan. (b) Transfer completion time.

Fig. 8. Results of Poisson release time.

as *discretized Poisson Process*: for each integer $t \leq T$, the number of events in $[t, t+1]$ is distributed according to $Poisson$ distribution with mean μ. Denote $\mathrm{Pow}(x_{max})$ the power law distribution truncated (and normalized) at y, formally, its probability density distribution is $f(x) \propto x^{-k}$, $0 \leq x \leq x_{max}$, with $k = 2$. Due to limited space, we report results of most representative settings as follows.

Zero Release Time. We generate traffic demand as follows. With the nodes in the topology, we randomly generate a bipartite graph with $n_1 = n_2 = \frac{1}{2}n$ nodes on each side. For each pair of nodes, we adds a data transfer between them with probability $p = 0.3$ with the size following $\mathrm{Exp}(128)$. The degree constraint of each node follows $\mathrm{Exp}(64)$. Fig 6 shows the results. We can see that Smith's greedy has smaller 90-pct makespan, average and 90-pct transfer completion times, which are consistent with our theoretical analysis.

Uniform Release Time. In this experiment, data transfers are generated as similar to the previous one except that release time follows $U(0, 128)$ and transfer size follows $\mathrm{Exp}(1024)$. The results are shown in Fig 7. Smith's greedy has smaller completion time, and the advantage becomes significant as the network size grows. This is because with larger network, the expected number of jobs incident to each job also increases and the benefits of sorting is bigger.

Poisson Release Time. This experiment changes release time to follow $\mathrm{Poisson}(3, 100)$ and transfer size to follow $\mathrm{Pow}(2048)$. The results are shown in Fig 8. The trend is opposite to Fig 7: as the network size grows, the advantage of Smith's greedy becomes less obvious. This is because in these traffic demands, the jobs incident to each node become more sparse when network size increases. The conclusion is, Smith's greedy is more effective when jobs are dense.

VII. Conclusion

In conclusion, this paper presents the first theoretical analysis of approximation algorithms for the data transfer problem in optical WANs. We prove competitive ratios of these algorithms in a variety of settings and use simulations to evaluate their performance in practice. Software-defined optical WANs are in its early stage. We hope this work can encourage future research to enhance the design and practice of optical WANs.

Acknowledgement: G. Ghasemiesfeh, J. Ding and J. Gao would like to acknowledge the support through NSF DMS-1418255, CCF-1535900, CNS-1618391 and AFOSR FA9550-14-1-0193.

References

[1] B. S. Baker and E. G. Coffman. Mutual exclusion scheduling. *Theoretical Computer Science*, 162(2):225 – 243, 1996.

[2] A. Bar-Noy, M. Bellare, M. M. Halldórsson, H. Shachnai, and T. Tamir. On chromatic sums and distributed resource allocation. *Information and Computation*, 140(2):183 – 202, 1998.

[3] H. L. Bodlaender and F. V. Fomin. Graph colorings equitable colorings of bounded treewidth graphs. *Theoretical Computer Science*, pages 22–30, 2005.

[4] H. L. Bodlaender and K. Jansen. On the complexity of scheduling incompatible jobs with unit-times. In *MFCS*, pages 291–300, 1993.

[5] B. B. Chen and P. V.-B. Primet. Scheduling deadline-constrained bulk data transfers to minimize network congestion. In *IEEE CCGRID*, May 2007.

[6] C. Chung, T. Nonner, and A. Souza. Srpt is 1.86-competitive for completion time scheduling. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 1373–1388. Society for Industrial and Applied Mathematics, 2010.

[7] J. E. G. Coffman, M. R. Garey, D. S. Johnson, and A. S. Lapaugh. Scheduling file transfers. *SIAM J. Comput.*, 14(3):744–780, 1985.

[8] G. Even, M. M. Halldórsson, L. Kaplan, and D. Ron. Scheduling with conflicts: online and offline algorithms. *Journal of Scheduling*, 12(2):199–224, 2009.

[9] R. Gandhi, M. M. Halldórsson, G. Kortsarz, and H. Shachnai. Improved bounds for scheduling conflicting jobs with minsum criteria. *ACM Trans. Algorithms*, 4(1):11:1–11:20, March 2008.

[10] M. R. Garey and D. S. Johnson. Tutorial: Hard real-time systems. chapter Complexity Results for Multiprocessor Scheduling Under Resource Constraints, pages 205–219. 1989.

[11] K. Giaro and M. Kubale. Edge-chromatic sum of trees and bounded cyclicity graphs. *Inf. Process. Lett.*, 75(1-2):65–69, July 2000.

[12] R. Graham, E. Lawler, J. Lenstra, and A. Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Discrete Optimization II*. 1979.

[13] R. Hartert et al. A declarative and expressive approach to control forwarding paths in carrier-grade networks. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 15–28, August 2015.

[14] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. Achieving high utilization with software-driven WAN. In *ACM SIGCOMM*, August 2013.

[15] S. Jain et al. B4: Experience with a globally-deployed software defined WAN. In *ACM SIGCOMM*, August 2013.

[16] X. Jin, Y. Li, D. Wei, S. Li, J. Gao, L. Xu, G. Li, W. Xu, and J. Rexford. Optimizing bulk transfer with software-defined optical wan. In *SIGCOMM*, August 2016.

[17] S. Kandula, I. Menache, R. Schwartz, and S. R. Babbula. Calendaring for wide area networks. In *ACM SIGCOMM*, volume 44, pages 515–526, August 2014.

[18] A. Kumar, S. Jain, U. Naik, A. Raghuraman, B. Carlin, M. Amarandei-Stavila, M. Robin, A. Siganporia, S. Stuart, and A. Vahdat. BwE: Flexible, hierarchical bandwidth allocation for WAN distributed computing. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 1–14, August 2015.

[19] N. Laoutaris, M. Sirivianos, X. Yang, and P. Rodriguez. Inter-datacenter bulk transfers with NetStitcher. In *ACM SIGCOMM*, August 2011.

[20] H. H. Liu, S. Kandula, R. Mahajan, M. Zhang, and D. Gelernter. Traffic engineering with forward fault correction. In *ACM SIGCOMM*, volume 44, pages 527–538. ACM, 2015.

[21] H. H. Liu, R. Viswanathan, M. Calder, A. Akella, R. Mahajan, J. Padhye, and M. Zhang. Efficiently delivering online services over integrated infrastructure. In *USENIX NSDI*, March 2016.

[22] D. Marx. Graph coloring problems and their applications in scheduling. 2003.

[23] R. Motwani, S. Phillips, and E. Torng. Non-clairvoyant scheduling. In *SODA*, 1993.

[24] K. Rajah, S. Ranka, and Y. Xia. Advance reservations and scheduling for bulk transfers in research networks. *IEEE Transactions on Parallel and Distributed Systems*, 20(11):1682–1697, 2009.

[25] L. Schrage. Letter to the editor—a proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 16(3):687–690, 1968.

[26] W. E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956.

[27] H. Zhang, K. Chen, W. Bai, D. Han, C. Tian, H. Wang, H. Guan, and M. Zhang. Guaranteeing deadlines for inter-datacenter transfers. In *EuroSys*, April 2015.