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 on 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 2-competitive ratio for jobs of arbitrary transfer size and each node having degree constraint 1. 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). Since it is extremely expensive to build and maintain WANs, WAN bandwidth is a very scarce resource. On the other hand, the quality of these services heavily depend on how fast data can be delivered to destinations. Therefore, it is critical for network operators to make 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 distributed protocols that run in the network. Routers do not have a global view of topology and 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 [17] and Microsoft SWAN [16] show that such centralized control can significantly improve network utilization.

Moreover, today’s optical technologies also 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. The optical layer 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 only constraints are degree upper bounds on 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 means a router can only have a certain number of links. Recent work such as Owan [18] shows that we can dramatically improve data transfers by jointly control the network routers and optical devices. However, the solution Owan [18] used heuristic algorithms 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 use which links. 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 their release time. We are mainly interested in the online setting though we also prove approximation ratios in the offline setting.

In the previous work [18] 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: Sort the currently available jobs in an
arbitrary order and schedule a job whenever the degree constraint is not violated. We prove it is 3-competitive in the online setting for minimizing makespan.

Further, for minimizing the sum of completion time, this algorithm is 2-competitive 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:** When the job requests form a bipartite graph, we can schedule them by choosing a perfect matching from the current set of jobs, which always reduce the ‘heaviest bottleneck’ in the requests. We prove 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 must consider the different job sizes. Thus we augment the simple greedy algorithm by first sorting the jobs in non-increasing 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 2-competitive for minimizing sum of completion time when all degree constraints are 1.

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

To summarize, this paper provides the first theoretical analysis of the data transfer problem in a reconfigurable optical WAN setting. 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 [16], [17], allocate capacity based on service priority and the incremental value of additional allocation [20], tolerate data plane and control plane failures [22], enforce policy-based routing [15], and jointly manage routers, proxies, load balancers, and DNS servers [23]. Besides these, there is a growing interest to go beyond network-level objectives like network throughput and focus on fine-grained transfer-level objectives like transfer completion time. Recent solutions have shown that by leveraging SDN we can significantly reduce transfer completion time [7], [19], [21], [26], [29]. 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 [18]. However, as we have pointed out, Owan does not have any mathematical analysis and theoretical guarantees on the proposed algorithms.

**Scheduling Algorithms.** Scheduling a well-studied class of problems. Here we only review those related to our problem. 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 [5]. Most of the works in literature have considered the problem where jobs are independent of each other. Both preemptive and non-preemptive and also offline/online versions of the problem on single/uniform parallel machines have been subject of many studies.

As a subset of problems with independent jobs, there is a family of scheduling problems called scheduling with conflicts [25], in which jobs could be in conflict with each other such that no two jobs in conflict can be scheduled at the same time (their availability could be online or offline). The conflict relationship can be described using a conflict graph. In online setting, upon arrival of a new job the conflict relationship to other jobs will be revealed. On the other hand, in 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. Once colored, the color of the vertex cannot be changed later. The relationship between graph sum multi-coloring problem and the scheduling problem with the objective of minimizing average completion time has explained thoroughly in [24], [13]. Gandhi et al. in [10] have studied the offline version with both preemptive and non-preemptive assumptions. Table I in [10] summarizes their own results and also the best known ones from [2], [3], [4], [14] very well. Even et al. in [9] and their tables I and II, have covered the known and new results on the same problem with the objective of minimizing maximum make span extensively. More specific results for different constraints on the graph and jobs lengths could be find in [11], [1], [6].

III. PROBLEM STATEMENT

Given a set of nodes \( V \), in which each node \( v_i \) represents a site on the WAN, we compute network-layer topology and transfer schedules to optimize data delivery. Each node has a degree constraint \( d_i \), which models the number of router ports at \( s_i \). It constrains \( s_i \) to have at most \( d_i \) links (optical circuits) to other nodes. A stream of transfers arrives at the system with \( (u_i, v_i, \ell_i, r_i) \) denotes the source, destination, size, release time of transfer \( i \). 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 transfers to schedule on these edges. Specifically, we have the following two variants of the problem 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 problem, we focus on the online version in which transfer requests arrive at each time slot. 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 $(u_i, v_i, t_i, r_i)$. $H$ 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. However, 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 $V'$. This special case sometimes yields better approximation results.

IV. MINIMIZE MAKESPAN

In this section, we focus on minimizing the makespan. First, we study 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. We consider preemptive schedule throughout this section.

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 [8]. 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 assume $T$ as the makespan of our algorithm and $T^*$ the optimal in offline setting. Obviously $T^* \geq T^*_j \geq r_j + r_j - \ell_j$ the best case happens when job $j$ is scheduled immediately when it is available.

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 $c$ (the capacity of a link). This new algorithm is optimal when all jobs are available at time 0 and is $2$-competitive in general.

If the largest degree is $k$, then we would need at least $\lceil k / d \rceil$ time slots to schedule all transfers by any algorithm. Now we argue that we can finish all transfers in the same number of slots, thus producing an optimal solution.

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. Obviously, 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.

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 $T^* \geq t + [\deg^*(u) + d'(u)] / d$. 
While 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 \( V = \{s_1, s_2, \ldots, s_n\} \), \( V' = \{s'_1, s'_2, \ldots, s'_n\} \). All nodes have the same degree constraint of \( d = 1 \). Each edge represents a job size 1. Figure 1 and 2 show that greedy scheduling and perfect matching based scheduling can be a factor 2 off from the optimal algorithm in the online setting, for the objective of 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 3 shows that greedy scheduling can be a factor 1.5 off from the optimal.

**V. MINIMIZE SUM COMPLETION TIME**

Now we study minimizing the sum completion time. This section is partitioned into four versions of the problem.

(A) Jobs have unit size, release time is zero.

(B) Jobs have unit size, release time can be arbitrary.

(C) Job sizes can be arbitrary, release time is zero.

(D) Job sizes can be arbitrary, release time can be arbitrary.

Note that (A) and (C) are offline settings. We analyze three different greedy algorithms. The simple greedy algorithm in

![Fig. 1. 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 \( s_i \), there is a single request \((s_i, s'_j)\). At time \( i \) there is a request \((s_i, s'_{i+1})\). 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.](image1)

![Fig. 2. 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 \( \{(s_j, s'_{j+1}) \mod n \mid 2 \leq j \leq n\} \). For the time \( i \) (1 \( \leq i \leq n - 1 \)), there is a request \((s_i, s'_{i+1}) \mod n \) in the optimal solution, we may schedule a perfect matching in each slot \( j = 0, 1, \ldots, n - 1 \) as shown in the figure, which does not use any edges in the edge set \( M = \{(s_j, s'_{j+1}) \mod n \mid 1 \leq j \leq n\} \) and at time \( n - 1 \) finish \( M \). The completion time is \( n \). Therefore, the completion time is \( 2n - 1 \).](image2)

![Fig. 3. 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.](image3)

**Definition 4.1.** 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. Unit Size Jobs, Zero Release Times**

We will show that greedy algorithm in Definition 4.1 gives a 2-approximation for scenario (A) with arbitrary degree constraints \( d_v \). When all the degree constraints \( d_v = 1 \), the special case is in fact exactly the edge chromatic sum problem, defined in the following.

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

The edges in matching \( M_t \) are colored \( t \), with a cost of \( t \). In scheduling, we schedule \( M_t \) in the \( t \)-th time slot and all
the jobs in $M_i$ 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 $I_i$, such that the vertex chromatic sum $\sum_i t \cdot |I_i|$ 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 $L(H)$, and 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, we say it is **compact**, if it is locally optimal, i.e., we can not move any edge/vertex to a matching/independent set with smaller index to obtain a new feasible edge/vertex coloring.

The Minimum Edge Chromatic Sum problem is NP-hard, even when $H$ is a bipartite graph [12]. But any compact edge-coloring is a 2-approximation [?]. In this work, we will show that the same approximation ratio can be obtained when the degree constraints $d_e$ are arbitrary. In the language of scheduling, these edges/jobs are scheduled in the $i$th time slot. Clearly the greedy scheduling algorithm in Definition 4.1 achieves a compact coloring. Thus the greedy algorithm is a 2-approximation for minimum sum completion time.

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 $\{d_e\}_{e \in E(V(H))}$. First we have a lower bound for $OPT$.

**Lemma 5.3 (Lower Bounds on OPT).** $OPT \geq \frac{1}{2}(n + \frac{1}{2} \sum_{u \in V(H)} \deg(u)^2/d_{d_e})$, 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 graphs $Q$ 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 of $\{Q_u\}$: the vertex in $Q_u$ carries the color of its corresponding edge in $H$; by definition of edge coloring at most $d_e$ edges incident to vertex $u$ can have the same color. Note that each edge appears in exactly two cliques, so $OPT \geq \frac{1}{2} \sum_{u \in V(H)} CL(Q_u)$. Recall that $CL(Q_u) \geq \frac{1}{2}(\deg(u) + \deg(u)2/d_{d_e})$. Hence, $OPT \geq \frac{1}{2} \sum_{u \in V(H)} (\deg(u) + \deg(u)2/d_{d_e}) = \frac{1}{2}(n + \frac{1}{2} \sum_{u \in V(H)} \deg(u)^2/d_{d_e})$ as desired.

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

**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 $\tau \leq t - 1$, either $d_u$ jobs incident to $u$ are scheduled at time slot $\tau$, or $d_v$ jobs incident to $v$ are scheduled at time slot $\tau$. In the first case, we charge $1/d_e$ unit to each of those $d_e$ edges. In the second case, we charge $1/d_e$ unit to each of the $d_e$ 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 by $1/d_e$ by all the edges incident to $u$ with colors higher than $j$'s color and $1/d_e$ unit by all the edges 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 $1/d_e$ to all the lower colors. The total charge collected on all these $\deg(u)$ edges in the neighborhood of $u$ becomes at most $\sum_{i=1}^{\deg(u) - 1} 1/d_e = \frac{n}{2} \deg(u)(\deg(u) - 1)/d_u$. Summing up over all vertices of $H$ we are done.

Combining the upper and lower bounds, we have:

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

**B. Unit Size Jobs, Arbitrary Release Times**

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 $d_e = 1$ 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 $x(v) \geq r(v)$ for any vertex $v$ and $x(v) \neq x(w)$ 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) in such a compact coloring scheme.

**Lemma 5.6.** Given $G$ with $n$ vertices and $m$ edges, any feasible compact vertex coloring $\chi$ satisfies $|\chi| \leq m + \sum_v r(v)$, 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 4. 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, ..., $r(v)$ – 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 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 at $|E_2| = \sum_v r(v) – 1 = \sum_v r(v) – n$, we are done. \(\square\)

**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 $q_n\lambda m + 2n + \frac{1}{2}(1 - \lambda)\sum_v r(v)$ 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 $CL(v_i)$ of each clique $Q_i$. That is, we give each vertex in $Q_i$ a color that is different from other colors in the same clique. Also the color of $v$ must be at least as big as $r(v)$. $CL(v_i)$ takes the sum of all the colors in this labeling. Then for each clique $Q_i$, we have two natural lower bounds of $CL(Q_i)$:

- $CL(Q_i) \geq \frac{q_i + 1}{2}$, 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. $CL(Q_i) \geq \lambda^{(q_i + 1)/2} + (1 - \lambda)\sum_{v \in Q_i} r(v)$, where $0 \leq \lambda \leq 1$. Summing over all cliques, we obtain

$$CL(G) \geq (1 - \lambda)\sum_{i=1}^{k} \sum_{v \in Q_i} r(v) + \lambda\sum_{i=1}^{k} \left(\frac{q_i + 1}{2}\right) = \lambda(m + 2n) + 2(1 - \lambda)\sum_{v \in V} r(v)$$

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. Therefore, we have a lower bound on any chromatic sum as stated. \(\square\)

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

**Proof:** Denote $r = \sum r(v)$. Then, from Lemma 5.6 and 5.7, we have

$$\frac{|X|}{OPT} \leq 2 \frac{m + n + r}{\lambda(m + 2n) + 2(1 - \lambda)r} < 2 \frac{1}{\lambda^{1/a} + (1 - \lambda)\frac{2}{1 + \frac{\alpha}{2}}}$$

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

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

**C. Zero Release Time, Arbitrary Size**

When all jobs are available at time zero but the job $i$ can have arbitrary size $\ell_i$, 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 [28].

**Definition 5.9 (Smith’s Greedy Algorithm).** We sort the jobs in non-increasing size. And schedule the jobs in a greedy manner with respect to the degree constraints. We allow a job to stop and re-start (i.e., using the preemptive setting).

We will show that Smith’s Greedy algorithm gives a 2-approximation on the sum completion time for problem (C).

We first sort (break ties arbitrarily) and relabel all jobs/edges so that $\{\ell_j\}$ is nondecreasing. Consider a job $j$ incident to a node $u$ (which could be source or destination), let $\text{rank}^+_u(j)$ (rank^+_v(j)) be the rank of j’s completion time among all jobs incident to u, in increasing (decreasing) order.

**Lemma 5.10.** The cost of the Smith’s greedy algorithm is at most

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

**Proof:** (Sketch) 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 $\ell_j$ in $G'$. If $(i, j) \in E(G)$, then 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, say by $z_1, ..., z_{c-1}$, we charge 1 for each edge $(u, z_k)$. Mimic the proof of charging scheme in (A), the charge for job $j$ is at most $\ell_j + \sum_{i \leq j, i \neq j} \ell_i$. Summing over all $j$ and we are done. \(\square\)

By mimicking Lemma 5.3, we have

**Lemma 5.11 (Lower Bound on OPT).**

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

Combining the 2 lemmas above, we immediately obtain

**Theorem 5.12.** The Smith’s Greedy algorithm is 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 $d_v = 1$ for all $v$ in this section. Our algorithm will use the well-known SRPT (Shortest Remaining Processing Time) algorithm as a subroutine. The SRPT schedule is optimal for single machine online scheduling with respect to the average completion time objective [27]. Note that this is not true for multiple machine version, which is NP-hard.

**Definition 5.13 (SRPT Algorithm).** For the single machine online scheduling problem, the SRPT algorithm is defined as follows: at each time slot, among all jobs that are alive (i.e. those already appeared but have not yet been completed), choose the one with the smallest remaining processing time.

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. Then we discuss how to adapt to the online setting.

Define $J(v)$ the list of data transfer requests that either starts from node $v$ or ends at node $v$. 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 $i$-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 $\ell$ will need $\ell$ time slots, called job units in $L$. Since the SRPT algorithm is preemptive, the $\ell$ job units might not be contiguous.

Note that each job $j = (v, w)$ should appear in exactly two lists, i.e. $L(v)$ and $L(w)$, with $\ell_j$ job units in each of them, and $2\ell_j$ job units in total. The units of $j$ in $L(v)$ are denoted by $u_{j,1}^v$, $u_{j,2}^v$, ..., $u_{j,\ell_j}^v$. We will view these $2\ell_j$ job units as distinct. Note that each of the $\ell_j$ job units of $j$ in $L(v)$ has a twin in the list $L(w)$.

For simplicity only state the offline version, Fig 5 for illustration.

**Definition 5.14 (Offline SRPT-based Scheduling Algorithm).**

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 union of all the job units in the SRPT lists. The job units in $A$ are sorted into a list $O$. 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. Notice that the job units in any individual list $L(v)$ appear in the same order 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 in $M$. 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 dummy or real unit.

3. Update the lists: Suppose unit $u$ from job $j = (src, des)$ is chosen, and without loss of generality suppose it is from $L(src)$, if it is a dummy unit, then just remove it; if it is not, then we delete it from $L(src)$, and also delete its twin $u'$ from $L(des)$.

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 $L(v)$. We analyze the online version below.

![Fig. 5. 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 (do not forget to delete the twins): $M_1 = \{(1, 3), (2, 1), (3, 2), (4, 3)\}$, $M_2 = \{(1, 3), (2, 4)\}$, $M_3 = \{(1, 3), (2, 4)\}$, $M_4 = \{(1, 3), (2, 4)\}$, $M_5 = \{(2, 3)\}$](image)

**Lemma 5.15.** For a job $j = (u, v)$, let $C^u(j)$ be the completion time of job $j$ in the SRPT schedule induced by the single-machine scheduling problem for all jobs incident to $u$. Similarly define $C^v(j)$. For a schedule $\sigma$ for our problem, define $C^\sigma(j)$ as the completion time of $j$. If $C^\sigma(j) \leq f \cdot (C^u(j) + C^v(j))$ for any $j$, for a factor $f$. Then, $\sigma$ is a $2f$-approximation for the min sum completion time problem.

**Proof:** Use TCT to denote total completion time. Let $\sigma^*$ be an optimal schedule for our SDN problem. Let $SRPT(u)$ denote the total completion time of the SRPT schedule for jobs incident to $u$. On the one hand,

$$OPT = TCT(\sigma^*) = \frac{1}{2} \sum_{u \in V} \sum_{j \sim u} C^\sigma(j) \geq \frac{1}{2} \sum_u SRPT(u).$$

On the other hand,

$$TCT(\sigma) \leq f \sum_{j \sim u} (C(u(j)) + C(v(j))) = f \sum_{u \in V} \sum_{j \sim u} C(j) = f \sum_{u} SRPT(u).$$

Combining these 2 inequalities and we are done. □

**Observation 5.16.** For any time $t$ and for any job $j = (v, w)$, if $L(v)$ contains unit $u_{j,h}^v$ and $u_{j,k}^w$ where $i \leq k$, then it must contain $u_{j,h}^v$ for all $h$ between $i$ and $k$. Similar claim holds...
For greedy scheduling algorithm, in the worst case, if edges \( j \) consist of Fig. 6.

The average completion time is \( n^2(7n + 1) \). When \( n \to \infty \), which is asymptotically 1.75OPT.

**Theorem 5.19.** Any online algorithm that minimizes the sum completion time is at least 1.5. off from the offline optimal in the worst case.

**Proof:** As shown in Figure 7, consider the case where we have 4 nodes in the bipartite graph, at time 0, there are requests \((s_2, s'_1)\) and \((s_2, s'_2)\). If we first finish \((s_2, s'_2)\) at time 1, the request \((s_1, s'_1)\) comes and vice versa. The optimal can finish the requests in 2 time slots while any algorithm must finish the requests in 3 time slots.

---

**Figure 7.** The lower bound for any online algorithm to minimize sum completion time.

---

**Fig. 8.** Results of zero release time. Dark bars are 90-pct makespan & avg. completion time: light bars are full makespan & 90-ct completion time. The x-axis is network size, and the y-axis is normalized time. Fig. 9 and 10 have the same legends.

---

**Fig. 9.** Results of uniform release time.

---

**Fig. 10.** Results of Poisson release time.

---

**VI. Evaluation**

In this section, we compare the performance of different algorithms on various settings with simulations. We compare the following three algorithms: (1) greedy1 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** which is 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 \( \text{exp}(2^i) \) the truncated exponential distribution: \( P(X = 2^i) = 2^{-(i+1)} \) for \( i = 0, \ldots, p - 1 \) and \( P(X = 2^p) = 2^{-p}. \) Denote Poisson\((\mu, T)\) 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 \( \mu. \) Denote 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 add a data transfer between them with probability \( p = 0.3 \) with the size following \( \text{exp}(128). \) The degree constraint of each node follows \( \text{exp}(64). \) Fig 8 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 \( \text{exp}(1024). \) The results are shown in Fig 9. 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 Poisson\((3, 100)\) and transfer size to follow Pow\((2048). \) The results are shown in Fig 10. The trend is opposite to Fig 9: 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.

**References**


