Hybrid Circuit/Packet Network Scheduling with Multiple Composite Paths

Shih-Hao Tseng*, Bo Bai†, and John C.S. Lui‡
*School of Electrical and Computer Engineering, Cornell University
†Future Network Theory Lab, 2012 Labs, Huawei Technologies, Co. Ltd.
‡Department of Computer Science and Engineering, The Chinese University of Hong Kong
Email: st688@cornell.edu, baibo8@huawei.com, cslui@cse.cuhk.edu.hk

***Abstract*—A switching/forwarding fabric with high-bandwidth one-to-one and low-bandwidth many-to-many data forwarding can be achieved by combing electronic packet switches (EPS) and optical circuit switches (OCS). This hybrid solution scales well but it is not suitable for cloud computing/datacenter applications, which typically rely on one-to-many and many-to-one communications. Recently, composite-path switching (cp-switching), which adds paths between EPS and OCS, is introduced to deal with this skewed traffic pattern. The state-of-the-art scheduling algorithm for cp-switches reduces, by heuristics, a cp-switching problem with one composite path to a hybrid switching problem without the composite path, and leverages existing scheduling techniques to tackle the latter problem. Unfortunately, the approach provides neither performance guarantee nor the support for multiple composite paths.**

In this paper, we systematically study the shortest time cp-switch scheduling problem with multiple composite paths and show that the problem can be expressed as an optimization problem with sparsity constraints. An LP-based algorithm is derived accordingly which supports multiple composite paths and more importantly, it provides performance guarantee. Simulations demonstrate that adding more composite paths can help shorten the schedule, and our approach outperforms existing methods by 30% to 70%.

I. Introduction

Electronic packet switches (EPS) can be combined with optical circuit switches (OCS) to provide low-bandwidth many-to-many and high-bandwidth one-to-one data forwarding [1], [2]. This hybrid solution achieves higher throughput with lower cost [3], which makes it an appealing architecture for data centers [4]–[6]. However, data center applications nowadays, such as MapReduce [7], Dryad [8], or more general coflows [9], require more than just one-to-one high-bandwidth forwarding. Instead, major applications involve one-to-many or many-to-one mappings, such as distributed data storage [10], [11], large-scale graph processing [12], parallel computation and partition-aggregation workflows [7], [8].

Supporting one-to-many or many-to-one traffic while offering high-bandwidth by OCS at the same time is challenging. The high-bandwith of OCS is achieved by establishing a physical circuit between the inport and the outport. To map from one inport to multiple outports, OCS needs to reconfigure the circuits multiple times to perform time division multiplexing (TDM) among the outports [13]. Each reconfiguration takes from tens of microseconds to few milliseconds [6], [13], which imposes a significant overhead when mapping to more outports and limits the use of such hybrid switching system for latency sensitive data center tasks [14].

To overcome the drawback, the idea of composite paths is proposed [15]. Since EPS nowadays supports heterogeneous port bandwidth, with many low-bandwidth ports and few high-bandwidth ports, one can connect an OCS outport to a high-bandwidth EPS inport (and vice versa) to create a composite path. Composite paths help resolve the OCS reconfiguration overhead by relying on the EPS to do multiplexing. For instance, because EPS can be reconfigured much faster, a one-to-many mapping can be established efficiently by mapping the OCS inport to a composite path through a unicast, then to the EPS, where the multicast is performed. Therefore, adding composite paths allows EPS to send more data to the outports under one-to-many/many-to-one mappings, and avoids OCS TDM but still provides higher input bandwidth for one-to-many/many-to-one scenarios.

Nevertheless, composite paths introduce new scheduling challenges. Without composite paths, EPS and OCS can be scheduled in parallel [3], [16]; while with composite paths, EPS and OCS are tangled together. The state-of-the-art approach CPSwitchSched [15] suggests some heuristics for translating the demand to be scheduled so that one can schedule switches with composite paths (cp-switches) using existing hybrid switch (h-switch) schedulers, which schedule EPS and OCS in parallel. Two technical issues of the translation based approach are as follows. Firstly, it does not provide any theoretical performance guarantee. Also, currently CPSwitchSched only works for one pair of composite paths. In general, we can have multiple composite paths especially at the system level. For example, the control plane of a software-defined network can control a set of EPS and OPS to function like a giant cp-switch [17]. To support multiple composite paths, a systematic generalization of CPSwitchSched is required.

A. Contribution and Organization

In this work, we take an alternative approach to address the two issues above. Instead of adopting problem translation philosophy, we include multiple composite paths into the model and systematically study the shortest time cp-switch scheduling problem. The main contributions of our work are the following: First of all, we show that the problem can be decoupled into several continuous-time control subproblems. These subproblems can also be formulated in the

The work was done when Shih-Hao Tseng was a summer research assistant at The Chinese University of Hong Kong, cooperating with the Future Network Theory Lab. The work of John C.S. Lui is supported in part by GRF 14200117 and Huawei's Research Grant.

 DOI: 10.1109/INFOCOM.2018.8485837.

form of mixed integer linear programs (MILP). Although the subproblems are proven NP-hard, we develop approximation algorithms with fixed approximation ratio to the original problem. By relaxing the subproblems in the form of MILP, a linear programming (LP) based approximation algorithm is developed. The algorithm uprounds the relaxed solution using maximum weight matching algorithms. We demonstrate by simulations under different loading scenarios that the proposed algorithm outperforms the existing approach.

The paper is organized as follows. In Section II, We first introduce the notation and formulate the continuous-time control problem, followed by its equivalent MILP form. The NP-hardness of the subproblems is shown in Section III-A, and the upper bound and the lower bound on the shortest schedule are given in Section III-B. The bounds help develop the LP-relaxation based algorithm in Section IV. The online version of the proposed algorithm is specified in Section IV-D. We conduct simulations in Section V to explore three main questions: benefits of multiple-composite-path adoption (Section V-B), performance under skewed demand (Section V-C), and effects of OCS reconfiguration overhead (V-D). And we conclude the paper in Section VI.

II. Formulation

We begin with the notation used in the following context in Section II-A and set up the problem in Section II-B. The continuous-time formulation of the problem is given in Section II-C, and its equivalent MILP formulation is in Section II-D.

A. Notation

Let $\mathbb{R}$ denote the the set of real numbers and $\mathbb{Z}$ the set of integers. We denote by 0 both the number zero and the all-zero square matrix. $\mathbf{1} \in \mathbb{R}^{N\times 1}$ is an all-one vector. $e_n \in \mathbb{R}^{N\times 1}$ is the unit vector with 1 at its n^{th} row. $\|\cdot\|_0$ counts the number of non-zero entries of the vector (also called l_0-norm in the literature). $[a, b]$ is an interval between a and b, while $\{a, b\}$ is a set having two elements a and b. $[a, b]_{\mathbb{Z}}$ is the set of all the integers within $[a, b]$. For short, we also write $a \in [1, A]_{\mathbb{Z}}$ as $a \in A$ when a is an integer. The name of a problem also refers to its optimal value, e.g., let $S = \min f$, symbol S can refer to either the problem itself or its optimal value ($\min f$).

B. Problem Setup

We model the network as a cp-switch consisting of N ports and P composite paths. As shown in Fig. 1, each port $n \in [1, N]_{\mathbb{Z}}$ connects to both EPS and OCS, and each composite path is a full-duplex link connecting EPS and OCS. A port can be both an inport and an outport with symmetric input/output capacities (bandwidth): Each EPS port has capacity c_E; each OCS port has capacity c_O; and each composite path is assumed to have capacity c_O as well. In general, $c_O \approx 10c_E \gg c_E$.

EPS supports many-to-many switching, while OCS functions as a crossbar switch: Each OCS inport can map to only one OCS outport and vice versa. For instance, OCS can switch from inport i to outport j and from inport j to outport k simultaneously, but we cannot send from inport i to both outports j and k at the same time.

We assume no buffer is installed at each composite path.

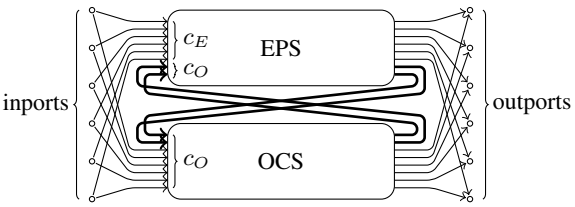


Fig. 1: A hybrid switch with composite paths (cp-switch). Composite paths are drawn in thick lines. Port capacities are marked accordingly.

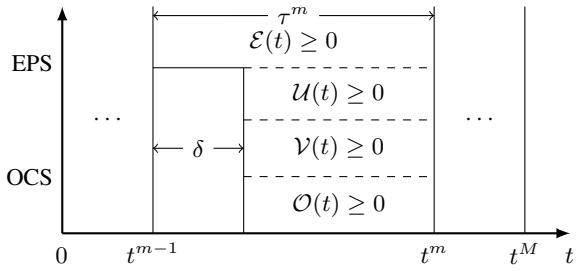


Fig. 2: The reconfiguration timeline of a step.

The demand is given by a matrix $D \in \mathbb{R}^{N\times N}$, and each entry D_{ij} refers to the amount of data that should be sent from port i to port j. By convention, we assume $D_{nn} = 0$ for all $n \in N$.

We want to find the shortest time $M + 1$ step schedule to satisfy the demand, i.e., to configure the cp-switch such that a total amount of data D_{ij} is sent from inport i to outport j for all $i, j \in N$. In step 0, only EPS is used, and the remaining M steps involve the whole cp-switch. Each step $m \in [0, M]_{\mathbb{Z}}$ is of time length τ^m. We reconfigure the cp-switch at the beginning of each step. The reconfiguration time of OCS is $\delta > 0$, and EPS does not need to be reconfigured. Instead, EPS can change the sending rate between ports freely.

C. Continuous-Time Control Formulation

Finding the shortest time schedule can be viewed as a continuous-time control problem and formulated as follows.

Without loss of generality, we assume the schedule starts at time $0 = t^{-1}$. The ending time of each step m is t^m, and $\tau^m = t^m - t^{m-1}$ for all $m \in [0, M]_{\mathbb{Z}}$ by definition.

Let $\mathcal{E}(t), \mathcal{U}(t), \mathcal{O}(t), \mathcal{V}(t) \in \mathbb{R}^{N\times N}$ be the mapping matrices at time t, which we can control. $\mathcal{E}_{ij}(t)$ is the sending rate from EPS inport i to EPS outport j at time t. Similar definitions apply to $\mathcal{U}_{ij}(t)$ mapping EPS to OCS, $\mathcal{O}_{ij}(t)$ mapping OCS to OCS, and $\mathcal{V}_{ij}(t)$ mapping OCS to EPS. The composite paths are represented implicitly by each column in $\mathcal{U}(t)$ and each row in $\mathcal{V}(t)$. Since we have only P composite paths, we can have at most P non-zero columns in $\mathcal{U}(t)$ and P non-zero rows in $\mathcal{V}(t)$.

A step comprises two phases: the reconfiguration phase and the sending phase. By definition, $\mathcal{U}(t), \mathcal{O}(t)$ and $\mathcal{V}(t)$ are all zero during step 0. During each step $m \in M$, EPS is always accessible; while $\mathcal{U}(t)$, $\mathcal{O}(t)$ and $\mathcal{V}(t)$ must be zero within time interval $[t^{m-1}, t^{m-1} + \delta]$ since the OCS is still under reconfiguration. Fig. 2 demonstrates the timeline of such steps.

With the above definitions, we can formulate the following continuous-time control problem $C(\hat{M})$ for $\hat{M} \in [0, M]_{\mathbb{Z}}$:

$$\begin{aligned}
\min \quad & \sum_{m=0}^{\hat{M}} \tau^m && (C(\hat{M})) \\
\text{s.t.} \quad & \tau^0 = t^0 \geq 0, \\
& \tau^m = t^m - t^{m-1} \geq \delta && \forall m \in \hat{M} \quad (1) \\
& \mathcal{E}(t) \geq 0, \quad \mathcal{U}(t) \geq 0, \\
& \mathcal{O}(t) \geq 0, \quad \mathcal{V}(t) \geq 0 && \forall t \geq 0 \quad (2) \\
& \int_0^{t^{\hat{M}}} \mathcal{E}(t) + \mathcal{U}(t) + \mathcal{O}(t) + \mathcal{V}(t)\ dt = D && (3) \\
& \text{capacity constraints (4)-(9)} \\
& \text{operation constraints (11)-(14)}
\end{aligned}$$

where condition (1) describes τ^m; condition (2) are the non-negativity constraints; and condition (3) is the demand satisfaction constraint. The capacity constraints (4)-(9) and the operation constraints (11)-(14) are elaborated in the following.

The capacity constraints are

$$\mathcal{U}(t) = 0, \quad \mathcal{O}(t) = 0, \quad \mathcal{V}(t) = 0 \qquad \forall t \in [0, t^0] \quad (4)$$

$$(\mathcal{E}(t) + \mathcal{U}(t))\, \mathbf{1} \leq c_E \mathbf{1} \qquad \forall t \geq 0 \quad (5)$$

$$(\mathcal{E}(t) + \mathcal{V}(t))^\top\, \mathbf{1} \leq c_E \mathbf{1} \qquad \forall t \geq 0 \quad (6)$$

$$(\mathcal{O}(t) + \mathcal{V}(t))\, \mathbf{1} \leq c_O \mathbf{1} \qquad \forall t \in [t^{m-1} + \delta, t^m], m \in \hat{M} \quad (7)$$

$$(\mathcal{O}(t) + \mathcal{U}(t))^\top\, \mathbf{1} \leq c_O \mathbf{1} \qquad \forall t \in [t^{m-1} + \delta, t^m], m \in \hat{M} \quad (8)$$

$$\mathcal{U}(t) = 0, \quad \mathcal{O}(t) = 0, \quad \mathcal{V}(t) = 0 \qquad \forall t \in [t^{m-1}, t^{m-1} + \delta], m \in \hat{M} \quad (9)$$

where condition (4) describes the requirements at step 0; conditions (5) and (6) are the EPS inport/outport capacity constraints; conditions (7) and (8) are the OCS inport/outport capacity constraints; and condition (9) requires no data transmission during OCS reconfiguration.

Define $E^m \in \mathbb{R}^{N \times N}$ as the total data sent by $\mathcal{E}(t)$ during $[t^{m-1}, t^m]$:

$$E^m = \int_{t^{m-1}}^{t^m} \mathcal{E}(t)\ dt. \quad (10)$$

Also define U^m from $\mathcal{U}$, V^m from $\mathcal{V}$, and O^m from $\mathcal{O}$ in the same way. And the operation constraints are

$$\left\|\mathbf{1}^\top U^m\right\|_0 \leq P \qquad \forall m \in \hat{M} \quad (11)$$

$$\left\|V^m \mathbf{1}\right\|_0 \leq P \qquad \forall m \in \hat{M} \quad (12)$$

$$\left\|e_n^\top V^m \mathbf{1}\right\|_0 + \left\|e_n^\top O^m\right\|_0 \leq 1 \qquad \forall m \in \hat{M}, n \in N \quad (13)$$

$$\left\|\mathbf{1}^\top U^m e_n\right\|_0 + \left\|O^m e_n\right\|_0 \leq 1 \qquad \forall m \in \hat{M}, n \in N \quad (14)$$

where conditions (11) and (12) confine the number of composite paths; condition (13) requires each OCS inport can send to only one OCS outport; similarly, condition (14) requires each OCS outport can receive from only one OCS inport.

The shortest time schedule can be found by solving $C(\hat{M})$ for $\hat{M} \in [0, M]_{\mathbb{Z}}$ and taking the shortest schedule among the solutions. Mathematically, we want to obtain

$$\text{OPT} = \min_{\hat{M} \in [0, M]_{\mathbb{Z}}} C(\hat{M})$$

and the corresponding schedule. Notice that $C(\hat{M})$ is always feasible for $\hat{M} \in [0, M]_{\mathbb{Z}}$. A trivial feasible solution is to use EPS only by setting $\mathcal{U}(t) = \mathcal{V}(t) = \mathcal{O}(t) = 0$ for all $t \geq 0$. (Section II-D shows that finding an EPS only schedule is merely an LP.) However, finding the optimal solution to $C(\hat{M})$ is not trivial as it involves continuous-time constraints (conditions (5)-(8)), integral constraints (conditions (3) and (10)), and sparsity constraints (conditions (11)-(14)). We can transform $C(\hat{M})$ into an equivalent mixed integer linear program (MILP), which allows us to propose our LP-based approximation algorithm in Section IV.

D. Mixed Integer Linear Programming Formulation

We first consider one kind of special mapping matrix:

Definition 1 (Piecewise Constant Mapping Matrix)**.** A mapping matrix $\mathcal{A}(t)$ is piecewise constant if and only if there exists a countable set $\{t^m\}_{m \in \mathbb{Z}}$ such that $\mathcal{A}(t) = \mathcal{A}(t^m)$ for all $t^m < t < t^{m+1}$, $m \in \mathbb{Z}$.

Inspired by [18], we have the following lemma which enables us to transform $C(\hat{M})$ into an MILP.

Lemma 1. *There exists a feasible solution to $C(\hat{M})$ if and only if there exists a feasible solution to $C(\hat{M})$ whose $\mathcal{E}(t), \mathcal{U}(t), \mathcal{O}(t)$ and $\mathcal{V}(t)$ are all piecewise constant.*

Proof. Similar to the proof of Proposition 1 in [18], given a solution, we can construct the corresponding piecewise constant solution to $C(\hat{M})$ by taking the average over the intervals $[0, \tau^0]$, $[\tau^{m-1}, \tau^{m-1} + \delta]$ and $[\tau^{m-1} + \delta, \tau^m]$ for all $m \in \hat{M}$. □

By Lemma 1, the constraint (3) is equivalent to

$$E^0 + \sum_{m=1}^{\hat{M}} E^m + U^m + O^m + V^m = D. \quad (15)$$

Regarding the constraints (5)-(8), let E^{mr} and E^{ms} be the data sent by $\mathcal{E}(t)$ during the OCS reconfiguration phase and sending phase of step m, which are given by

$$E^{mr} = \int_{t^{m-1}}^{t^{m-1}+\delta} \mathcal{E}(t)\ dt, \qquad E^{ms} = \int_{t^{m-1}+\delta}^{t^m} \mathcal{E}(t)\ dt.$$

We know for $m \geq 1$

$$E^m = E^{mr} + E^{ms}.$$

We can decouple U^m, O^m and V^m in the same way, and the constraint (5) can be transformed to two constraints corresponding to the OCS reconfiguration phase and sending phase:

$$\begin{aligned}
&(E^{mr} + U^{mr})\, \mathbf{1} \leq c_E \delta \mathbf{1}, \\
&(E^{ms} + U^{ms})\, \mathbf{1} \leq c_E (\tau^m - \delta) \mathbf{1}.
\end{aligned}$$

Since $U^{mr} = 0$ and $U^{ms} = U^m$, the above constraints are reduced to

$$E^{mr}\mathbf{1} \le c_E\delta\mathbf{1},$$
$$(E^{ms} + U^m)\,\mathbf{1} \le c_E(\tau^m - \delta)\mathbf{1}.$$

Similarly, we can reduce the two transformed constraints of constraint (7) to one constraint, and the constraints (5)-(8) are rewritten as

$$E^0\mathbf{1} \le c_E\tau^0\mathbf{1}, \quad E^{0\top}\mathbf{1} \le c_E\tau^0\mathbf{1}$$
$$E^{mr}\mathbf{1} \le c_E\delta\mathbf{1}, \quad E^{mr\top}\mathbf{1} \le c_E\delta\mathbf{1} \qquad \forall m \in \hat{M} \quad (16)$$
$$(E^{ms} + U^m)\,\mathbf{1} \le c_E(\tau^m - \delta)\mathbf{1} \qquad \forall m \in \hat{M} \quad (17)$$
$$(E^{ms} + V^m)^\top\,\mathbf{1} \le c_E(\tau^m - \delta)\mathbf{1} \qquad \forall m \in \hat{M} \quad (18)$$
$$(O^m + V^m)\,\mathbf{1} \le c_O(\tau^m - \delta)\mathbf{1} \qquad \forall m \in \hat{M} \quad (19)$$
$$(O^m + U^m)^\top\,\mathbf{1} \le c_O(\tau^m - \delta)\mathbf{1} \qquad \forall m \in \hat{M} \quad (20)$$

To replace the sparsity constraints (11)-(14), we introduce the indicators $\hat{U}^m \in \{0,1\}^{N\times 1}, \hat{O}^m \in \{0,1\}^{N\times N}, \hat{V}^m \in \{0,1\}^{N\times 1}$ such that

- $\hat{U}^m_{n1} = 1$ if $\mathbf{1}^\top U^m e_n > 0$.
- $\hat{O}^m_{ij} = 1$ if $O^m_{ij} > 0$.
- $\hat{V}^m_{n1} = 1$ if $e_n^\top V^m\mathbf{1} > 0$.

Since $\left\|\mathbf{1}^\top U^m\right\|_0 \le \mathbf{1}^\top\hat{U}^m$, constraint (11) is satisfied if and only if there exists $\hat{U}^m$ such that $\mathbf{1}^\top\hat{U}^m \le P$. Therefore, we can translate the sparsity constraints to

$$\overline{D}\hat{U}^m \ge U^{m\top}\mathbf{1}, \qquad \overline{D}\hat{O}^m \ge O^m, \qquad \overline{D}\hat{V}^m \ge V^m\mathbf{1},$$
$$\mathbf{1}^\top\hat{U}^m \le P, \quad \mathbf{1}^\top\hat{V}^m \le P,$$
$$\hat{V}^{m\top}e_n + \mathbf{1}^\top\hat{O}^{m\top}e_n \le 1, \quad \hat{U}^{m\top}e_n + \mathbf{1}^\top\hat{O}^m e_n \le 1 \quad (21)$$

where $\overline{D} = \sum_{i,j\in N} D_{ij}$, so that the MILP equivalent to $C(\hat{M})$ is expressed as

$$\min \quad \sum_{m=0}^{\hat{M}} \tau^m \qquad (I(\hat{M}))$$
$$\text{s.t.} \quad \text{time constraints (1)}$$
$$\text{demand constraints (15)}$$
$$\text{capacity constraints (16)-(20)}$$
$$\text{sparsity constraints (21)}$$
$$E^0 \ge 0, \quad E^m = E^{mr} + E^{ms}$$
$$E^{mr} \ge 0, \quad E^{ms} \ge 0,$$
$$U^m \ge 0, \quad O^m \ge 0, \quad V^m \ge 0,$$
$$\hat{U}^m \in \{0,1\}^{N\times 1}, \quad \hat{O}^m\{0,1\}^{N\times N}$$
$$\hat{V}^m \in \{0,1\}^{N\times 1} \qquad \forall m \in \hat{M}$$

We refer to this MILP as $I(\hat{M})$.

We name a set of feasible indicators $\hat{U}^m$, $\hat{O}^m$, and $\hat{V}^m$ an *OCS configuration* at step m, as it dictates how the OCS should be configured at step m.

III. Analysis

We show the NP-hardness of $C(\hat{M})$ in Section III-A. The upper and lower bounds are established in Section III-B, which help derive a fixed approximation ratio for the scheduling algorithms upper bounded by $L(0)$.

A. NP-hardness

Although $C(\hat{M})$ is equivalent to $I(\hat{M})$, it is still in the form of an MILP instead of a polynomial time solvable form such as an LP. The following proposition suggests that the scheduling problem is actually NP-hard.

Proposition 1. *$C(\hat{M})$ is NP-hard.*

Proof. In [19], scheduling an OCS with reconfiguration delay is shown NP-hard, which is a special case of $C(\hat{M})$ with $c_E = 0$. □

We remark that NP-hardness of $C(\hat{M})$ does not imply the problem OPT itself is NP-hard, but it does prevent us from obtaining the optimal solution by iteratively solving $C(\hat{M})$ for all $\hat{M} \in M$ (notice that $C(0)$ is a polynomial-time solvable LP). In Section III-B, we investigate the upper and lower bounds given by LP relaxations, and show how an approximation algorithm can be designed with a bounded approximation ratio.

B. Performance Bounds and Approximation Ratio

We can relax the integer indicators $\hat{U}^m, \hat{O}^m$ and $\hat{V}^m$ to take value from interval $[0,1]$, which yields the relaxed linear program $L(\hat{M})$.

$$\min \quad \sum_{m=0}^{\hat{M}} \tau^m \qquad (L(\hat{M}))$$
$$\text{s.t.} \quad \text{constraints in } I(\hat{M}) \text{ but with}$$
$$\hat{U}^m \in [0,1]^{N\times 1}, \quad \hat{O}^m[0,1]^{N\times N}$$
$$\hat{V}^m \in [0,1]^{N\times 1} \qquad \forall m \in \hat{M}$$

Some useful facts about $L(\hat{M})$ are given in Lemma 2.

Lemma 2. *The following inequalities hold:*

- $c_E L(0) \le (c_E + c_O)L(1)$.
- $L(1) \le L(k)$ *for all* $k > 1$.

Proof. To show the first inequality, we consider an optimal solution achieving $L(1)$ with the step interval lengths $\tilde{\tau}^0$ and $\tilde{\tau}^1$. It must satisfies conditions (16)-(20). Summing up those conditions and we get

$$D\mathbf{1} \le \left(c_E\tilde{\tau}^0 + (c_E + c_O)\,\tilde{\tau}^1 - c_O\delta\right)\mathbf{1} \le (c_E + c_O)L(1),$$
$$D^\top\mathbf{1} \le \left(c_E\tilde{\tau}^0 + (c_E + c_O)\,\tilde{\tau}^1 - c_O\delta\right)\mathbf{1} \le (c_E + c_O)L(1),$$

which implies that $E^0 = D$, $\tau^0 = \frac{c_E + c_O}{c_E}L(1)$ is a feasible solution to $L(0)$. Since $L(0)$ achieves the optimal value, we know $L(0) \le \frac{c_E + c_O}{c_E}L(1)$.

Second inequality can be shown in a similar way. Consider an optimal solution $\tilde{E}^m, \tilde{U}^m, \tilde{O}^m, \tilde{V}^m$ to $L(k)$ with step time

length $\tilde{\tau}^m$ for all $m \in [1,k]_{\mathbb{Z}}$. We construct a $L(1)$ solution as the following

$$E^{1s} = \sum_{m\in[1,k]_{\mathbb{Z}}} \tilde{E}^m - \tilde{E}^{1r}, \qquad U^1 = \sum_{m\in[1,k]_{\mathbb{Z}}} \tilde{U}^m,$$
$$O^1 = \sum_{m\in[1,k]_{\mathbb{Z}}} \tilde{O}^m, \qquad V^1 = \sum_{m\in[1,k]_{\mathbb{Z}}} \tilde{V}^m,$$
$$\hat{U}^1 = \frac{U^1}{\overline{D}}, \qquad \hat{O}^1 = \frac{O^1}{\overline{D}}, \qquad \hat{V}^1 = \frac{V^1}{\overline{D}}.$$

Summing up conditions (16) for step 2 to k and (17) for step 1 to k and we get

$$\left(E^{1s} + U^1\right)\mathbf{1} \le c_E\left(L(k) - \tilde{\tau}^0 - \delta\right)\mathbf{1}.$$

Also, summing up conditions (19) from step 1 to k yields

$$\begin{aligned}\left(O^1 + V^1\right)\mathbf{1} &\le c_O\left(L(k) - \tilde{\tau}^0 - k\delta\right)\mathbf{1} \\ &\le c_O\left(L(k) - \tilde{\tau}^0 - \delta\right)\mathbf{1}.\end{aligned}$$

The same computation applies to both conditions (18) and (20). Therefore, combining the steps 1 to k as one step results in a feasible solution to $L(1)$ with the value $L(k)$, which implies $L(1) \le L(k)$. □

Lemma 2 says two things:

- We can upper bound $L(0)$ by a scaled $L(1)$. Since $C(0) = I(0) = L(0)$, we know OPT is upper bounded by $L(0)$ and hence also by a scaled $L(1)$.
- $L(1)$ lower bounds $L(k)$ for all $k \ge 1$. Since $C(k) = I(k) \ge L(k)$, we can lower bound all $C(k)$ by $L(1)$.

Those properties lay the foundation for establishing the upper and lower bounds on the shortest schedule time OPT in Lemma 3 and deriving the approximation ratio in Corollary 1.

Lemma 3. *If $L(0) \le \delta$, $\mathrm{OPT} = L(0)$ and the shortest time schedule uses EPS only. Otherwise,*

$$L(0) \ge C(1) \ge \mathrm{OPT} \ge L(1).$$

Proof. Since $C(0) = I(0) = L(0)$, we know

$$L(0) \ge \mathrm{OPT} = \min_{\hat{M}\in[0,M]_{\mathbb{Z}}} C(\hat{M}).$$

If $L(0) \le \delta$, since condition (1) implies $C(\hat{M}) \ge \delta$ for all $\hat{M} \ge 1$, we have $\mathrm{OPT} = L(0)$.

If $L(0) > \delta$, we have $L(0) \ge C(1)$ since the solution to $L(0)$ is also a feasible solution to $C(1)$ with zero matrices U^1, O^1 and V^1. Also, we know $C(\hat{M}) = I(\hat{M}) \ge L(\hat{M})$ for all $\hat{M} \ge 1$. By Lemma 2, the desired inequality is derived. □

Then we have Corollary 1, which provides an approximation ratio for the algorithms upper bounded by $L(0)$.

Corollary 1. *Any cp-switch scheduling algorithm adopting $L(0)$ as an upper bound is a $\frac{c_E+c_O}{c_E}$-approximation algorithm.*

Proof. Let the solution given by the cp-switch scheduling algorithm be S. By the assumption and Lemma 3, we have

$$L(0) \ge S \ge \mathrm{OPT} \ge L(1),$$

which suggests

$$\frac{S}{\mathrm{OPT}} \le \frac{L(0)}{L(1)},$$

and the corollary follows from Lemma 2. □

Another corollary from Lemma 3 is stated below. It allows us to find a solution with shorter time, which sheds light on our algorithm design in Section IV.

Corollary 2. *Given $k \ge 0$, if there exists a feasible solution S to $C(k)$ with $\tau^0 > \delta$, we have a feasible solution S' to $C(k+1)$ such that $S' \le S$.*

Proof. We prove by construction. Let step 0 of S be E^0. We can define another $L(0)$ with the demand equals to E^0. As such, we have $L(0) = \tau^0 > \delta$. By Lemma 3, we have another $C(1)$ solution satisfying the demand with $C(1) \le L(0)$. Replacing the step 0 in S with the $C(1)$ solution and we get a feasible $C(k+1)$ solution S' with $S' \le S$. □

IV. Proposed Algorithms

We propose our LP-relaxation based approximation algorithm (Algorithm 2) guided by three design features:

- Upper bound $L(0)$.
- Maximum weight matching uprounding procedure.
- Step increment decision.

Firstly, the resulted schedule of Algorithm 2 is upper bounded by $L(0)$, and hence Corollary 1 gives the approximation ratio. Since our algorithm is based on LP relaxation, how the relaxed integer variables are uprounded decides the quality of the solution. We follow a best-effort heuristic which relies on the polynomial-time solvable maximum weight matching algorithm. The details are described in Section IV-A. In Section IV-B, we illustrate when a new step should be introduced. And Algorithm 2 is presented in Section IV-C. We also show that Algorithm 2 can be applied to the online scenario, and the online version Algorithm 3 is given in Section IV-D.

A. Maximum Weight Matching Uprounding Procedure

Upon obtaining an $L(1)$ solution, we have to determine how to uproud the integer variables $\hat{U}^1, \hat{O}^1$, and $\hat{V}^1$. Our idea is to find an OCS configuration which can send the most data required by the $L(1)$ solution.

Let $\mathbf{1}_{\{N\times P\}}$ be the all-one matrix with N rows and P columns, we define the OCS forwarding matrix H as

$$H = \begin{bmatrix} O^1 & V^1\mathbf{1}_{\{N\times P\}} \\ \mathbf{1}_{\{P\times N\}}U^1 & 0 \end{bmatrix}.$$

The meaning of H is as follows. Consider the OCS switch ports formed by the rows and the columns of H, with the demand, or the weights, between inports and outports defined by the entries of H. A perfect matching defines a possible OCS configuration, and its corresponding aggregated weight reflects the maximum amount of data that can be forwarded under the configuration. So by finding the maximum weight (demand) matching of H, we can find an OCS configuration that sends the most parts of U^1, O^1, and V^1. The uprounding algorithm is then given by Algorithm 1.

Algorithm 1: Uround(S)

Input: S: the schedule from $L(1)$ based on some demand.
Output: OCS configuration $\hat{U}^1, \hat{O}^1$, and $\hat{V}^1$.

1: Form the OCS forwarding matrix H from S.
2: $W \in \{0,1\}^{(N+P)\times(N+P)}$ = MaxWeightMatching(H).
3: **return** OCS configuration:

$$\hat{O}^1_{ij} = W_{ij},$$
$$\hat{U}^1_j = \sum_{k\in P} W_{(N+k)j}, \qquad \hat{V}^1_i = \sum_{k\in P} W_{i(N+k)},$$

for all $i, j \in N$.

We apply the classical Kuhn-Munkres Algorithm [20] as the maximum weight matching algorithm in our design. We note that lower time-complexity algorithms are also available, and we refer the reader to [21] for a survey.

B. Step Increment Decision

Corollary 2 allows us to check if we can improve a solution by adding one more step. Suppose we have a feasible $C(1)$ solution with the time length $\overline{\tau}$ and the OCS configuration $\hat{U}^1$, $\hat{O}^1$, and $\hat{V}^1$. We can consider problem Q below.

$$\begin{aligned} \max \quad & \tau^0 && (Q)\\ \text{s.t.} \quad & \text{constraints in } I(1) \text{ with given } \hat{U}^1, \hat{O}^1, \text{ and } \hat{V}^1 \\ & \tau^0 + \tau^1 \le \overline{\tau} \end{aligned}$$

Solving problem Q yields a feasible solution with the longest possible τ^0 within time $\overline{\tau}$. As suggested by Corollary 2, if $\tau^0 > \delta$, we can then further shorten step 0 by adding one more step.

C. Iterative cp-Switch Scheduling

The three features introduced above are summarized in Algorithm 2, our proposed algorithm.

Algorithm 2: Iterative cp-Switch Scheduling

1: The residual demand $D_{res} \leftarrow D$.
2: The upper bound $\overline{C} \leftarrow L(0)$.
3: The resulted schedule $R \leftarrow \emptyset$.
4: **for** $m = 1, \ldots, M$ **do**
5: **if** $\overline{C} \le \delta$ **then**
6: Break the loop.
7: **end if**
8: Solve $L(1)$ based on D_{res} and store the schedule as S.
9: Get OCS configuration $\Theta \leftarrow$ Uround(S).
10: Solve $L(1)$ based on Θ, let $\overline{\tau} \leftarrow L(1)$.
11: Solve Q based on $\overline{\tau}$ and store the schedule as S'.
12: Set $\overline{C}$ as the τ^0 in S'.
13: Assign the step 1 in S' as step m in R.
14: $D_{res} \leftarrow D_{res} - (E^m + U^m + O^m + V^m)$.
15: **end for**
16: Solve $L(0)$ based on D_{res} as step 0 in R.

Algorithm 2 keeps track of the residual demand D_{res} and the upper bound $\overline{C}$ on τ^0. In each iteration it finds a two step LP-relaxed schedule $L(1)$, and uround it to get the OCS configuration for step 1. Given the OCS configuration, $L(1)$ is solved to obtain the minimum possible time $\overline{\tau}$. Within time length $\overline{\tau}$, we find the schedule with the longest τ^0 via solving Q. The solution to Q is a feasible solution to $C(1)$, which is stored as the two step schedule S'. The step 1 in S' is then assigned as a new step of the resulted schedule, and we update D_{res} and $\overline{C}$ respectively.

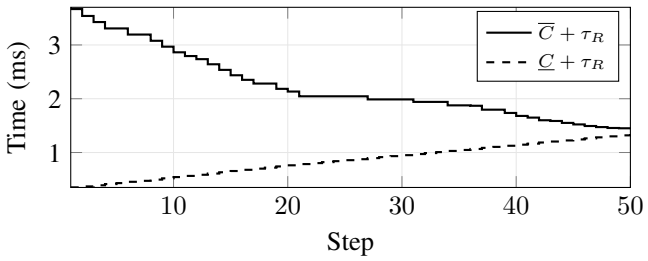


Fig. 3: Algorithm 2 shrinks the bounding interval.

Let $\underline{C}$ be the length of S and τ_R be the length of the assigned steps in R. The interval $[\underline{C}+\tau_R, \overline{C}+\tau_R]$ bounds the length of the final schedule given by Algorithm 2. Algorithm 2 actually shrinks the interval at each iteration as in Fig. 3.

Since Algorithm 2 is upper bounded by $L(0)$ in line 2, the proposition below follows from Corollary 1.

Proposition 2. *Algorithm 2 is a $\frac{c_E+c_O}{c_E}$-approximation algorithm.*

D. Online cp-Switch Scheduling

Each iteration in Algorithm 2 depends on only the current state of D_{res} and $\overline{C}$. Thus we can generalize Algorithm 2 to the online scenario, i.e., the algorithm keeps only one step and some necessary states in the memory, instead of computing the whole schedule with $M + 1$ steps and applying them sequentially.

The online version Algorithm 3 schedules whenever the switch is not IDLE, and it keeps only the current step in its memory. Notice that if the demand D is given at time 0, Algorithm 3 is equivalent to Algorithm 2.

Algorithm 3: Online cp-Switch Scheduling

1: Update D to be the data left to send.
2: **if** the switch is not IDLE **then**
3: Wait until new demand arrives.
4: **end if**
5: **if** $L(0) \le \delta$ **then**
6: Schedule E^0.
7: **else**
8: Line 8 - 11 in Algorithm 2.
9: Schedule the step 1 in S' and set the switch BUSY. The switch is set back to IDLE after time τ^1 in S'.
10: **end if**

V. Simulation

We implement and compare Algorithm 2 with the state-of-the-art scheduling algorithm CPSwitchSched [15]. Simulations

are conducted to explore three issues: the benefits of adopting multiple composite paths (Section V-B), the performance under skewed demand (Section V-C), and the effects of OCS reconfiguration overhead (Section V-D).

A. Simulation Setup

CPSwitchSched converts the cp-switch scheduling problem with one composite path to an augmented hybrid switch (h-switch) scheduling problem without the composite path. The converted problem is then solved by existing h-switch scheduling algorithms. Here we implement two h-switch scheduling algorithms to compare with: Solstice [3] and Eclipse [16].

Solstice assumes the demand matrix is sparse and skewed, and hence it stuffs the demand matrix to make each row and column sum to the same value. The stuffed demand matrix is then iteratively sliced by finding an OCS configuration that sends a large portion of the demand. Slicing involves finding a perfect matching of a bipartite graph, which is done by Hopcroft-Karp Algorithm [22] in our implementation.

Eclipse also schedules in a greedy manner but with a more sophisticated idea of matching. Instead of just finding a perfect matching, it tries to find a maximum weight matching which maximizes the average utilization of OCS. As in Algorithm 2, we use Kuhn-Munkres Algorithm [20] to find the maximum weight matching.

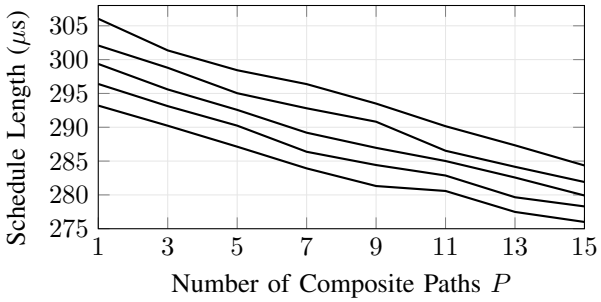
In the simulations, the cp-switch has $N = 32$ ports. The EPS port capacity c_E is set to 10 Gbps, and the OCS port capacity is 100 Gbps. The number of scheduling steps is upper bounded by $M = 15$.

The demand D is generated based on four loading conditions: meshed multicast, skewed multicast, lighter loading , and heavier loading. Under meshed multicast, we generate for each $i, j \in N$, $i \neq j$, a demand D_{ij} uniformly random over $[100, 130]$ (kB); while for skewed multicast, we pick each inport with probability 0.5 and let each picked inport i send random demand D_{ij} to the outport j with probability $\frac{1}{3}$. The skewed multicast depicts the case when the network is dominated by several one-to-many traffic. Lighter loading and heavier loading are just the meshed multicast with the demand D_{ij} distributed uniformly random over $[1, 1.3]$ (Mb) and $[100, 130]$ (Mb), respectively. As the original design in [15], CPSwitchSched operates under $R_t = 0.7N$, $B_t = 2$ (Mb) for the multicast settings and the lighter loading, and $B_t = 200$ (Mb) for the heavier loading.

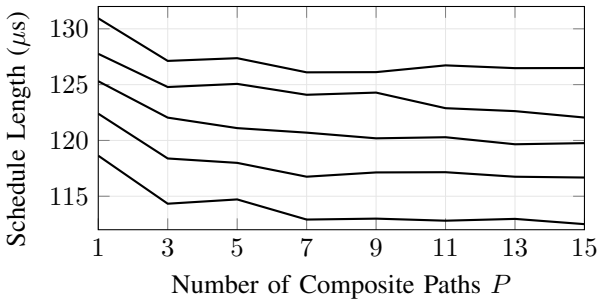
B. Benefits of Multiple-Composite-Path Adoption

We first examine if more composite paths may help to find a shorter schedule. 250 random demand D are generated under meshed multicast and skewed multicast with OCS reconfiguration time $\delta = 20$ (μs). Given each demand, we apply Algorithm 2 with $P = 1, 3, \ldots, 15$ composite paths. The results are collected and their percentiles are plotted in Fig. 4. From the bottom to the top lines represent the 30^{th} to the 70^{th} percentiles.

Fig. 4 shows that having more composite paths helps reduce the resulted schedule length. The performance improvement is more significant under meshed multicast (Fig. 4(a), the 50^{th} percentile reduces 6.5%) than under skewed multicast



(a) Meshed multicast: random demand D_{ij} is generated for each $i, j \in N$, $i \neq j$.



(b) Skewed multicast: each inport i initiates a one-to-many demand with probability 0.5.

Fig. 4: From the bottom to the top are the lines corresponding to the 30^{th}, 40^{th}, 50^{th}, 60^{th}, and 70^{th} percentiles of the schedule length given by Algorithm 2. Having more composite paths reduces the resulted schedule time.

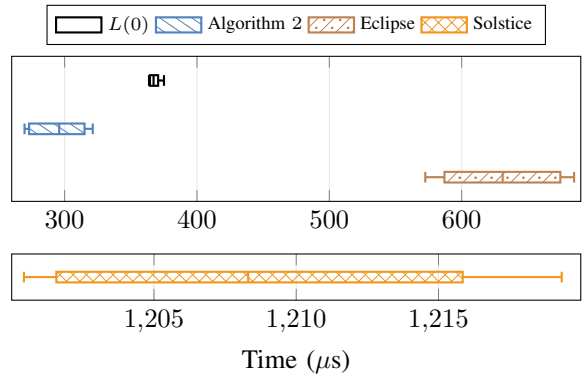
(Fig. 4(b), the 50^{th} percentile reduces 4.4%). The reason is simple: The skewed multicast has a less denser demand matrix than the meshed multicast. When the number of the composite paths exceed the number of multicast inports, adding more composite path does not help reduce the length of the schedule. We can also see the phenomenon in Fig. 4(b). The marginal improvement of adding an additional composite path decreases.

C. Performance under Skewed Demand

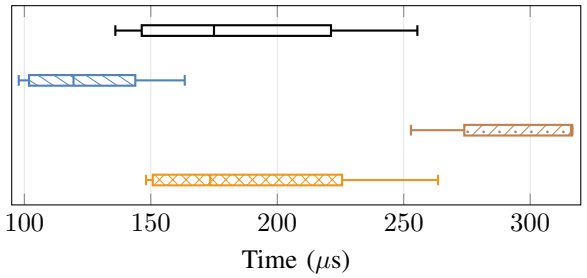
Another issue we are interested in is how well the algorithms perform under skewed multicast loading. Since Algorithm 2 and CPSwitchSched are both designed for one-to-many and many-to-one traffic, we expect the algorithms would perform better under the skewed setting. The expectation is confirmed in Fig. 5, which summarizes the results from 100 random demand. Under meshed multicast, the 50^{th} percentile of Algorithm 2 schedule is 75.5% and 53.1% faster than CPSwitchSched with scheduler Solstice and Eclipse, respectively. Under skewed multicast, it is 31.2% and 62.2% faster.

Under skewed multicast, Algorithm 2 is relatively closer to the upper bound $L(0)$. That is because the skewed multicast demand is sparser, which leads to shorter schedule. When the schedule is shorter, using OCS would be less efficient than using EPS which does not suffer the reconfiguration delay.

An interesting result from Fig. 5(b) is that CPSwitchSched with scheduler Solstice outperforms the one with Eclipse. It results from the facts that the performance of CPSwitchSched



(a) Meshed multicast with $\delta = 20$ (μs).



(b) Skewed multicast with $\delta = 20$ (μs).

Fig. 5: The $1^{\text{st}} - 5^{\text{th}} - 50^{\text{th}} - 95^{\text{th}} - 99^{\text{th}}$ percentiles of the schedule length given by each algorithms. Under both loading conditions, Algorithm 2 outperforms CPSwitchSched with h-switch scheduler Solstice or Eclipse.

greatly depends on its h-switch scheduler [15], and that Solstice is developed under the sparse demand assumption [3]. Therefore, the performance of Solstice improves under skewed multicast.

D. Effects of OCS Reconfiguration Overhead

The third issue we investigate is how OCS reconfiguration overhead δ affects the length of the resulted schedule. 100 random demand matrices are generated under lighter loading setting. The generated demand is multiplied by 10 to create a heavier load. We vary δ under different loading conditions and Fig. 6 shows the percentiles of the resulted schedule length. $L(0)$ is also plotted, which is the shortest time schedule using only EPS.

The results suggest that Algorithm 2 outperforms CPSwitchSched under all circumstances. In Table I, we compute the performance improvement of Algorithm 2 on the 50^{th} percentile over CPSwitchSched with different schedulers. The improvement is indicated by the ratio of the length difference between the Algorithm 2 schedule and the CPSwitchSched schedule to the CPSwitchSched schedule length. CPSwitchSched using Eclipse as its h-switch scheduler performs better than using Solstice, which results from the fact that Eclipse finds a matching with maximum weight instead of just finding a perfect matching as in Solstice.

CPSwitchSched can perform relatively better (closer to $L(0)$) when δ is shorter. It results from the fact that the two h-switch schedulers Solstice and Eclipse tend to greedily put more demand on OCS. As the reconfiguration overhead δ decreases, more data can be sent through OCS within the same period, and hence the performance is improved.

TABLE I: Performance improvement of Algorithm 2 on the 50^{th} percentile over CPSwitchSched with different schedulers.

Loading, δ	Solstice	Eclipse
Lighter, 20 (μs)	70.2%	27.2%
Lighter, 200 (μs)	75.0%	54.4%
Heavier, 2 (ms)	71.4%	27.8%
Heavier, 20 (ms)	75.8%	54.4%

However, there is a major design drawback of CPSwitchSched demand reduction, which prevents it to fully enjoy the advantage of introducing a composite path. A composite path has asymmetric port capacity: On one side it respects to EPS port capacity c_E, and on the other side it is constrained by OCS port capacity c_O. During the CPSwitchSched demand reduction, CPSwitchSched greedily aggregates the demand that might be sent through the composite path as OCS demand. The h-scheduler schedules the converted OCS demand with respect to the only the OCS port capacity c_O. As such, the demand that is allocated to composite paths may exceed how much the composite paths can support due to EPS constraints. CPSwitchSched then allocates the excess demand to EPS, which normally takes much longer time to deliver the surplus than OCS.

Algorithm 2 takes a conservative approach. Every maximum weight matching in Algorithm 1 is a feasible solution to both the OCS and the composite paths. As such, it avoid overloading the composite paths and losing throughput by involving EPS.

VI. Conclusion

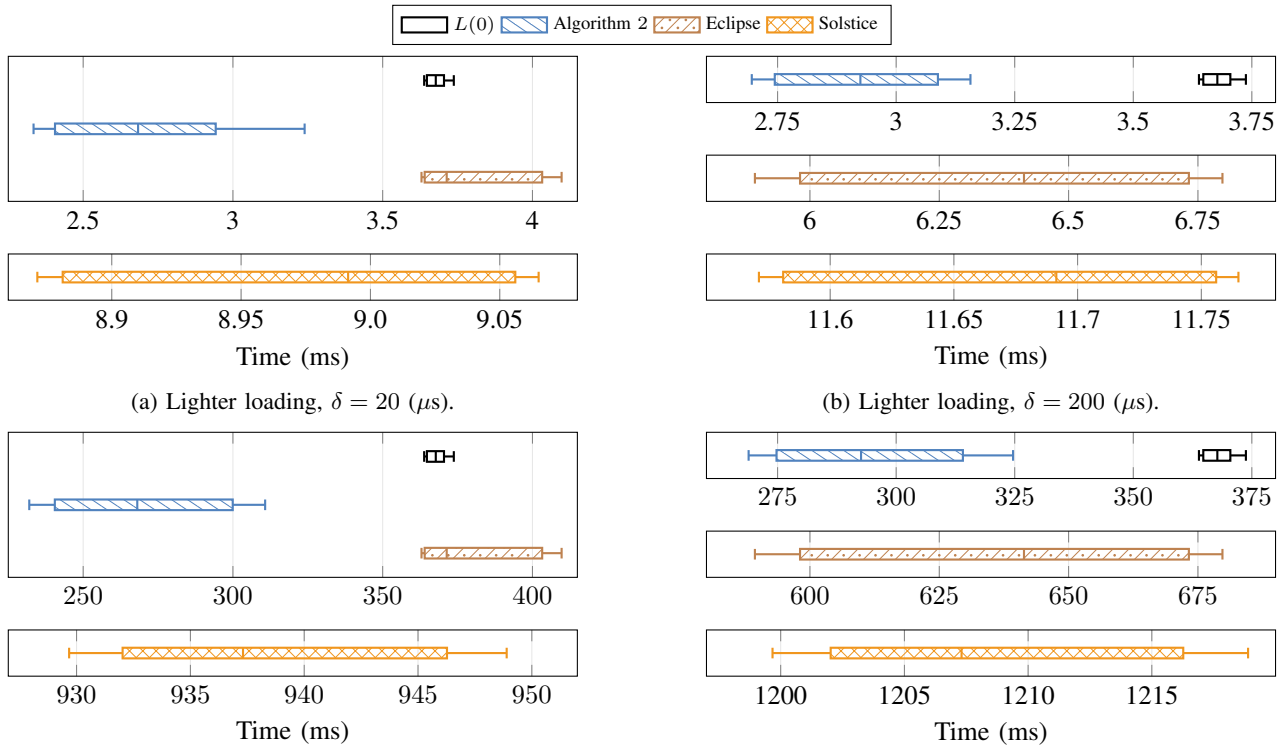
We study the shortest time cp-switch scheduling problem with multiple composite paths, which can be applied to hybrid networks consisting of EPS and OCS. The problem is formulated and shown to have an MILP form. Due to NP-hardness, the shortest schedule cannot be obtained by solving the subproblems for each step. However, we can still establish upper and lower bounds on the shortest schedule, which leads to an LP-relaxation based approximation algorithm with fixed approximation ratio. The proposed algorithm can work online, and simulations demonstrate that it outperforms the existing algorithm, which leverages only one composite path.

References

[1] K. J. Barker, A. Benner, R. Hoare, A. Hoisie, A. K. Jones, D. K. Kerbyson, D. Li, R. Melhem, R. Rajamony, E. Schenfeld *et al.*, "On the feasibility of optical circuit switching for high performance computing systems," in *Proc. ACM/IEEE Supercomputing*. IEEE Computer Society, 2005, p. 16.

[2] C. M. Gauger, P. J. Kuhn, E. V. Breusegem, M. Pickavet, and P. Demeester, "Hybrid optical network architectures: Bringing packets and circuits together," *IEEE Commun. Mag.*, vol. 44, no. 8, pp. 36–42, 2006.

[3] H. Liu, M. K. Mukerjee, C. Li, N. Feltman, G. Papen, S. Savage, S. Seshan, G. M. Voelker, D. G. Anderson, M. Kaminsky, G. Porter, and A. C. Snoeren, "Scheduling techniques for hybrid circuit/packet networks," in *Proc. CoNEXT*. ACM, 2015, pp. 41:1–41:13.



(a) Lighter loading, $\delta = 20$ (μs).

(b) Lighter loading, $\delta = 200$ (μs).

(c) Heavier Loading, $\delta = 2$ (ms).

(d) Heavier Loading, $\delta = 20$ (ms).

Fig. 6: The $1^{\text{st}} - 5^{\text{th}} - 50^{\text{th}} - 95^{\text{th}} - 99^{\text{th}}$ percentiles of the resulted schedule length under different loading conditions. Algorithm 2 is always bounded by the upper bound $L(0)$ and it outperforms CPSwitchSched with either Solstice or Eclipse as its h-switch scheduler.

[4] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat, "Helios: A hybrid electrical/optical switch architecture for modular data centers," *ACM SIGCOMM CCR*, vol. 40, no. 4, pp. 339–350, 2010.

[5] G. Wang, D. G. Andersen, M. Kaminsky, K. Papagiannaki, T. Ng, M. Kozuch, and M. Ryan, "c-Through: Part-time optics in data centers," *ACM SIGCOMM CCR*, vol. 40, no. 4, pp. 327–338, 2010.

[6] G. Porter, R. Strong, N. Farrington, A. Forencich, P. Chen-Sun, T. Rosing, Y. Fainman, G. Papen, and A. Vahdat, "Integrating microsecond circuit switching into the data center," *ACM SIGCOMM CCR*, vol. 43, no. 4, pp. 447–458, Aug. 2013. [Online]. Available: http://doi.acm.org/10.1145/2534169.2486007

[7] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[8] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," in *ACM SIGOPS operating systems review*, vol. 41, no. 3. ACM, 2007, pp. 59–72.

[9] M. Chowdhury and I. Stoica, "Coflow: A networking abstraction for cluster applications," in *HotNets*. ACM, 2012, pp. 31–36.

[10] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proc. USENIX NSDI*. USENIX Association, 2012, pp. 2–2.

[11] Apache hadoop project. [Online]. Available: http://hadoop.apache.org/

[12] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: A system for large-scale graph processing," in *Proc. ACM SIGMOD*. ACM, 2010, pp. 135–146.

[13] H. Wang, Y. Xia, K. Bergman, T. Ng, S. Sahu, and K. Sripanidkulchai, "Rethinking the physical layer of data center networks of the next decade: Using optics to enable efficient$*$-cast connectivity," *ACM SIGCOMM CCR*, vol. 43, no. 3, pp. 52–58, 2013.

[14] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," *ACM SIGCOMM CCR*, vol. 41, no. 4, pp. 50–61, 2011.

[15] S. Vargaftik, K. Barabash, Y. Ben-Itzhak, O. Biran, I. Keslassy, D. Lorenz, and A. Orda, "Composite-path switching," in *Proc. CoNEXT*. ACM, 2016, pp. 329–343.

[16] S. Bojja Venkatakrishnan, M. Alizadeh, and P. Viswanath, "Costly circuits, submodular schedules and approximate carathéodory theorems," in *ACM SIGMETRICS*, vol. 44, no. 1. ACM, 2016, pp. 75–88.

[17] M. Shirazipour, W. John, J. Kempf, H. Green, and M. Tatipamula, "Realizing packet-optical integration with SDN and openflow 1.1 extensions," in *Proc. IEEE ICC*. IEEE, 2012, pp. 6633–6637.

[18] A. Gushchin, S.-H. Tseng, and A. Tang, "Optimization-based network flow deadline scheduling," in *Proc. IEEE ICNP*, nov 2016.

[19] X. Li and M. Hamdi, "On scheduling optical packet switches with reconfiguration delay," *IEEE J. Sel. Areas Commun.*, vol. 21, no. 7, pp. 1156–1164, 2003.

[20] J. Munkres, "Algorithms for the assignment and transportation problems," *SIAM J. Appl. Math.*, vol. 5, no. 1, pp. 32–38, 1957.

[21] R. Duan and H.-H. Su, "A scaling algorithm for maximmum weight matching in bipartite graphs," in *Proc. SODA*. SIAM, 2012, pp. 1413–1424.

[22] J. E. Hopcroft and R. M. Karp, "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs," *SIAM J. Comput.*, vol. 2, no. 4, pp. 225–231, 1973.