2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC)

Quantized BvND: A Better Solution for Optical and Hybrid Switching in Data Center Networks

Liang Liu*, Jun (Jim) Xu† and Lance Fortnow*
School of Computer Science, Georgia Institute of Technology
Email: *{lliu315, fortnow}@gatech.edu, †jx@cc.gatech.edu

***Abstract*—Data center network continues to grow relentlessly in the amount of data traffic it has to "switch" between its server racks. A traditional data center switching architecture, consisting of a network of commodity packet switches (viewed as a giant packet switch), cannot scale with this growing switching demand. Adding an optical switch, which has a much higher bandwidth than the packet switch but incurs a nontrivial reconfiguration delay, to a data center network has been proposed as a cost-effective approach to boosting its switching capacity. However, to effectively do so, we need to meticulously schedule the optical switch. In fact, we are dealing with two very different scheduling problems here, namely hybrid switching and standalone optical switching, depending on whether or not there is effective cooperation between the optical switch and the packet switch during their respective scheduling processes.**

In this work, we propose a solution that performs better than the respective state of art solutions for both scheduling problems. Our solution outperforms by a wide margin all existing optical switching solutions in terms of throughput, yet its computational complexity is comparable to those of others. Our solution also has the best properties of both Eclipse and Solstice, the state of the art hybrid switching solutions. Eclipse and Solstice have different advantages: Eclipse has better throughput performance but incurs a much higher computationally complexity than Solstice. Our solution gets the better of both worlds: it delivers almost the same throughput performance as Eclipse, yet incurs a similar computational complexity as Solstice.

I. Introduction

Fueled by the phenomenal growth of cloud computing services, data center network continues to grow both in size, as measured by the number of server racks, and in link speed. This has led to an explosive growth in the amount of traffic the data center has to switch between its server racks [1]. A traditional data center switching architecture typically consists of a three-level multi-rooted tree of packet switches that starts, at the lowest level, with the Top-of-Rack switches, that each connects a rack of servers to the network [2]. However, such architecture is struggling to scale with this explosively growing switching capacity demand, as we can no longer increase the switching capabilities of the underlying commodity packet switches without increasing their costs significantly.

Adding a circuit switch, typically an optical switch, to the network has been proposed as a possible solution approach to this scalability problem [3], [4], [5], [6], [7], [8]. The optical switch has a much higher bandwidth than the packet switch, but incurs a nontrivial reconfiguration delay δ when the switch *configuration* has to change. Depending on the underlying technology of the optical switch, δ can range from tens of microseconds to tens of milliseconds [4], [9], [3], [10], [11].

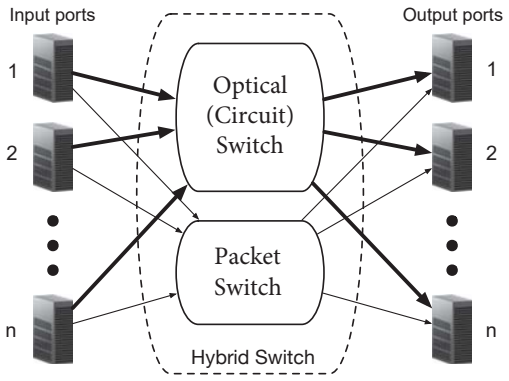

Fig. 1. Hybrid Circuit and Packet Switch

Figure 1 illustrates a dual-switch architecture, in which n racks of computers on the left hand side are connected, by both an optical switch and a packet switch, to n racks on the right hand side. Note that racks on the left hand side is an identical copy of those on the right hand side; however for simplicity of exposition, we assume that the former only transmit data and refer to them as *input ports*, and assume that the latter only receive data and refer to them as the *output ports*. This way, we model each switch as a bipartite graph, and its configuration a bipartite matching over this bipartite graph.

A. Two Different Scheduling Problems

In order to significantly boost the throughput performance of the network, we need to carefully schedule the optical switch to simultaneously fully utilize the bandwidth and minimize the reconfiguration delays. We have significantly different challenges, as outlined below, depending on whether or not we have *effective cooperation* between the optical switch and the packet switch during their respective scheduling processes. Nevertheless in this work we propose a solution that kills both birds (scheduling problems) with one stone.

In the first dual-switch system, the optical switch and the packet switch operate in close coordination with each other to maximize the overall throughput of the dual-switch system. More specifically, these two switches are jointly scheduled, and for every rack-to-rack traffic flow during a scheduling epoch, the joint scheduler decides which portion of it should

978-1-5386-5504-7/18/$31.00 ©2018 IEEE
DOI 10.1109/UCC.2018.00032

be switched by the optical switch and "at what times", while the rest, if any, is handled by the packet switch. This scheduling problem is known as hybrid circuit (optical) and packet switching [12], [13] in the literature, and we simply call it hybrid switching in the sequel. Intuitively, with an ideal hybrid switching schedule, the optical switch needs to remove the bulk of the overall traffic workload, using as few configurations as possible (so that the reconfiguration overhead is kept to a minimum), leaving a residue workload that is barely small enough for the packet switch to handle.

In the second dual-switch system, the optical and the packet switches operate independently of each other with each switch given a separate traffic workload to schedule and transmit. This scenario is equivalent to scheduling a stand-alone optical switch, which is well studied in the optical networking literature [14], [15], [16], [17], [18], [19], [11].

We highlight two major differences between the two systems. First, a stand-alone optical switch has to "sweep clean" the entire workload assigned to it and cannot leave any residue for the packet switch to handle, whereas the optical switch can in hybrid switching. Second, in hybrid switching, how the overall inter-rack traffic demand during an epoch is split into these two workloads is an essential part of the scheduling computation, whereas in optical switching, it is decided "exogenously" (i.e., not a part of the scheduling computation) [4].

B. Our Algorithm

In this work, we propose a solution that performs better than the respective state of art solutions for both scheduling problems. Our solution outperforms by a wide margin all existing optical switching solutions in terms of throughput, yet its computational complexity is comparable to those of others, as we will show in § VI-C. Our solution also has the best properties of both Eclipse [13] and Solstice [12], the state of the art hybrid switching solutions. Eclipse and Solstice have different advantages: Eclipse has better throughput performance but incurs a much higher computationally complexity than Solstice. Our solution, when used for hybrid switching, gets the better of both worlds: it delivers almost the same throughput performance as Eclipse, yet incurs a similar computational complexity as Solstice. We need to keep the complexity low because a scheduling epoch is typically no more than a few milliseconds long to keep file transmission delays low enough for various cloud applications, and ideally the computation of the schedule should take no more than the length of an epoch.

The Birkhoff-von Neumann decomposition (BvND), which expresses a doubly stochastic matrix as the sum of permutation matrices, lies at the heart of many hybrid and optical switching algorithms, including our solution. The standard BvND algorithm [20] (to be presented in Algorithm 1), however, results in a quadratic number of configurations ($O(n^2)$ in our context) and hence cannot be used for optical switching with nontrivial reconfiguration delay. While finding the minimum number of configurations is NP-complete [21], we give an efficient algorithm that results in only a linear number of configurations (i.e., $O(n)$).

Our BvND algorithm, based on the well-known technique of quantization, works as follows. A traffic demand matrix is first quantized into an s-integer (i.e., an integral multiple of s) for a certain $s > 0$, and carefully stuffed to an s-integer matrix that is scaled doubly stochastic. Then, an existing algorithm is applied iteratively to this s-integer matrix to find and remove, in each iteration, a max-min matching [22], [23] that corresponds to a configuration that has the longest duration and contains no *slack* (i.e., an input-output connection running idle) in our context, until the entire matrix is zeroed out. Since the quantized, and stuffed traffic matrix is s-integral, the max-min BvND algorithm guarantees to remove at least s amount of traffic from every row/column of the matrix in each iteration. Hence the number of iterations or configurations is upper-bounded roughly by $O(1/s)$, which is $O(n)$ since $s = \Omega(1/n)$ in our context. We refer to our solution as Quantized BvND (QBvND).

We keep the computational complexity of QBvND low using an algorithmic trick that can reduce the total asymptotical computational complexity of these $O(n)$ max-min matching computations (iterations) by a multiplicative factor of $O(\log n)$ by take advantage of the integral nature of the quantized and stuffed matrix. We will also describe in § IV that a slight alternation gives a bandwidth- and computationally-efficient hybrid switching algorithm as well.

The rest of the paper is organized as follows. In § II, we describe the system model and formulate the problems of both optical and hybrid switching. In § III, we provide the background on BvND. In §IV, we describe our solution QBvND in details. In §V and §VII, we survey related works. In §VI, we evaluate our solution QBvND against other optical and hybrid switching algorithms. Finally, we conclude the paper in §VIII.

II. System Model and Problem Formulation

In this section we describe the formal model for the optical and hybrid switching problems. In both cases we are given an $n \times n$ traffic demand matrix D, and each matrix entry $D(i, j)$ is the amount of traffic that originates at input port (rack) i and is destined for output port (rack) j, within a short (e.g., 3 milliseconds long) scheduling epoch of the recent past (e.g., from 4 milliseconds ago to 1 millisecond ago). Our optical or hybrid switching algorithm needs to meet this demand in the next scheduling epoch. We assume full knowledge of the precise and complete demand matrix D (in this recent past epoch), as do all prior works on hybrid switching save Albedo [24] (to be discussed in §V-C), and on optical switching.

A. The Optical Switching Problem

Given a traffic demand matrix D, we aim to compute a schedule that minimizes the *transmission time*, the amount of time for the optical switch to transmit D. An alternative

formulation, used in Eclipse [13], would maximize the effective throughput, i.e., the amount of traffic that the optical or hybrid switch can transmit within a scheduling epoch of fixed duration. These two formulations are roughly equivalent, as mathematically the latter is roughly the dual of the former. Hence throughout this work we use transmission time as the metric in all performance evaluation plots, but effective throughput as the metric in interpreting these plots.

A schedule of the optical switch consists of a sequence of configurations (matchings) and their durations: (P_1, α_1), (P_2, α_2), $\cdots$, (P_K, α_K). Each configuration P_k is an $n \times n$ permutation (matching) matrix where $P_k(i,j) = 1$ if input i is connected to output j and $P_k(i,j) = 0$ otherwise. The transmission time of the above schedule is $\sum_{k=1}^{K}(\alpha_k + \delta)$, where δ is the reconfiguration delay, and K is the number of configurations in the schedule. The optical switching problem aims to minimize this transmission time, under the constraint that the configurations P_k with their respective durations α_k can "sweep clean" the traffic matrix D, or mathematically

$$\min\{K\delta + \sum_{k=1}^{K} \alpha_k\} \text{ such that } D \leq \sum_{k=1}^{K} \alpha_k P_k \qquad (1)$$

Since this optimization problem is NP-hard [18], almost all optical switching solutions use heuristics.

B. The Hybrid Switching Problem

In a hybrid switching system, the packet switch is typically an electronic switch, which is an order of magnitude or more slower than the optical switch. For example, the optical and packet switches might operate at the respective rates of 100 Gbps and 10 Gbps per port. However, unlike the optical switch, the packet switch does not incur a reconfiguration delay when its configuration (matching) changes from one switching cycle to the next.

In hybrid switching, the optical switch is allowed to leave a small residue matrix $R \triangleq (D - \sum_{k=1}^{K} \alpha_k P_k)^+$ for the packet switch to handle. Suppose the per-port rate of the packet switch is r_p. Then no row sum or column sum of the reside matrix R can exceed r_p, as otherwise the corresponding input or output port would be given a workload larger than its its capacity. Mathematically, this constraint can be written as

$$\mathbb{1}^T \cdot R \leq r_p \cdot \mathbb{1}^T \text{ and } R \cdot \mathbb{1} \leq r_p \cdot \mathbb{1} \qquad (2)$$

where $\mathbb{1}$ is a column vector with n scalars that all have value 1, and T stands for transpose. Hence the mathematical formulation of the hybrid switching problem is the same as that of optical switching, except that the constraint in (1) is changed to (2) above. This optimization problem is also NP-hard, because optical switching is a special case of it (where $r_p = 0$).

III. Background on Birkhoff-von Neumann Decomposition

Since many optical and hybrid switching solutions are based on Birkhoff-von Neumann Decomposition (BvND), we provide a brief introduction of it in this section.

A. Preliminaries

We say that a nonnegative $n \times n$ matrix M is *doubly stochastic* (or doubly sub-stochastic) if every row or column sum of M is equal to 1 (or no larger than 1). The Birkhoff-von Neumann Theorem [20] states that a doubly stochastic matrix M can be expressed as a linear combination of permutation matrices. More precisely, we have

$$M = \sum_{k=1}^{K} \alpha_k P_k \qquad (3)$$

where $\sum_{k=1}^{K} \alpha_k = 1$ and $P_1, P_2, ..., P_K$ are *permutation matrices*, in which each row or column has exactly one non-zero entry with value 1.

Algorithm 1: BvND

Input : Doubly stochastic matrix M;
Output: BvND of M: $M = \sum_{k=1}^{K} \alpha_k P_k$;

```
k ← 1;
while M is not a zero matrix do
    Find a perfect matching P_k in graph M;
    α_k ← min{weights of the edges ∈ P_k};
    M ← M − α_k P_k;
    k ← k + 1;
end
```

The standard BvND algorithm, which is used in the constructive proof of the Birkhoff-von Neumann Theorem, is shown in Algorithm 1. In this algorithm, the matrix M is viewed also as a weighted bipartite graph, with n vertices, denoted as I_1, I_2, ..., I_n, that correspond to the n rows of M in one partite, and another n vertices, denoted as O_1, O_2, ..., O_n that correspond to the n columns of M in the other partite; the bipartite graph contains an edge between I_i and O_j if and only if the (current) value of the matrix element m_{ij} is nonzero, in which case the weight of the edge is m_{ij}. In each (say k^{th}) iteration, Algorithm 1 first finds a perfect matching, which corresponds to a permutation matrix P_k, in the bipartite graph, and then subtracts $\alpha_k P_k$ from M. Here the coefficient α_k is set to the smallest value among the weights of the edges in the perfect matching, so that after the subtraction, at least one previously nonzero matrix element will become zero; once a matrix element becomes zero, the corresponding edge is deleted from the bipartite graph in performing the subsequent iterations. Hence at most n^2 iterations are needed to zero out the matrix M, and Algorithm 1 has a total computational complexity of $O(n^{4.5})$, when the classical $O(n^{2.5})$ maximum cardinality matching algorithm [25] is used to find a perfect matching in each iteration.

Let M be a doubly stochastic matrix. We call any uM, where $u > 0$ is a scaling factor, a scaled doubly stochastic matrix. Clearly, any scaled doubly stochastic matrix can also be expressed as a linear combination of permutation matrices, and

this decomposition can also be computed using Algorithm 1. In this case, the sum of the linear coefficients $\sum_{k=1}^{K} \alpha_k$ is equal to u instead of 1.

B. The Stuffed BvND Algorithm

A BvND-based algorithm was proposed in [26] for packet switching when the traffic arrival rate matrix Λ is constant or very slowly varying. It consists of two steps, namely stuffing and BvND, so we call it stuffed BvND in the sequel. In the stuffing step, the matrix Λ, which is in general not scaled doubly stochastic, is stuffed into a scaled doubly stochastic matrix Λ', using the von Neumann stuffing algorithm [27]. In the BvND step, Algorithm 1, the standard BvND algorithm, is used to decompose the scaled doubly stochastic matrix Λ' into $\sum_{k=1}^{K} \alpha_k P_k$. To service the incoming traffic, the packet switch simply repeats the following schedule forever (or until Λ changes): use permutation matrix P_1 as the switch configuration for a duration of α_1, P_2 for a duration of α_2, ..., and P_K for a duration of α_K. Since this schedule is precomputed for repeated use, we refer to such an operation as precomputed packet switching in the sequel.

The stuffed BvND algorithm in general works for neither optical nor hybrid switching, because the resulting decomposition would in general consist of $K = O(n^2)$ different configurations, where n is the number of racks, and hence incur a prohibitively high reconfiguration delay cost; it works well for packet switching because the reconfiguration delay there is zero. For this reason, although many existing optical and hybrid switching solutions are based on BvND, they all have to somehow reduce this number K.

IV. Our Solution: Quantized BvND (QBvND)

In this section, we first provide an overview of QBvND in §IV-A, then describe the max-min BvND step of QBvND in §IV-B, and finally discuss how to tune a critical parameter of QBvND in §IV-C.

A. Pseudocode of QBvND

In this section, we first provide an overview of QBvND in the context of optical switching, and then describe, toward the end, how to modify it slightly for hybrid switching.

Algorithm 2: QBvND

1 Quantize D to an s-integer matrix $D^{(Q)}$;
2 Stuff $D^{(Q)}$ to a scaled doubly stochastic matrix $D^{(QS)}$;
3 Max-min BvND of $D^{(QS)}$ into $\sum_{k=1}^{K} \alpha_k P_k$;

The pseudocode of our solution, called Quantized BvND or QBvND in short, is shown in Algorithm 2. As its name suggests, QBvND prepends a quantization step before the two other steps described earlier, namely stuffing and BvND. In the quantization step, an appropriate quantization unit $s > 0$ is first chosen and fixed, and then every element D_{ij} in a doubly sub-stochastic traffic demand matrix $D \triangleq (D_{ij})_{n \times n}$ is rounded up to $\lceil \frac{D_{ij}}{s} \rceil \cdot s$, the nearest s-integer, defined as an integral multiple of s. The resulting s-integer matrix is denoted as $D^{(Q)} \triangleq (D_{ij}^{(Q)})_{n \times n}$.

Due to the positive rounding error (no larger than s) added to each matrix element of $D^{(Q)}$ in the rounding-up process, the maximum row or column sum of $D^{(Q)}$ could exceed 1 (i.e., the matrix may not be doubly sub-stochastic any more), but is clearly upper-bounded by $1+ns$. As indirectly explained at the end of § III-A, this maximum row or column sum corresponds to the minimum net (excluding the reconfiguration delays) transmission time $\sum_{k=1}^{K} \alpha_k$ required to serve traffic matrix $D^{(Q)}$. Hence this step is not slack-free in the sense the minimum net transmission time in general increases after the quantization step.

In the second step, the s-integer matrix $D^{(Q)}$ is carefully stuffed into another s-integer matrix $D^{(QS)}$ that is at least as large (i.e, $D^{(QS)} \geq D^{(Q)}$) and is scaled doubly stochastic. This problem, known as matrix stuffing, has been studied in [27], [12], [28]. Among many different such algorithms, we use for our solution QBvND a greedy heuristic algorithm, called QuickStuff, that was used in the Solstice algorithm [12] for hybrid switching, because it possesses two desirable properties that others generally don't. The first property is that, if the input M is an s-integer matrix (hence ϕ is an s-integer), the output computed by QuickStuff is also an s-integer matrix. As explained earlier, preserving this "s-integrality" during the stuffing step is necessary for the next step (BvND) to cap K, the number of configurations (matchings) in the resulting schedule, at $O(n)$. The second property is that, QuickStuff avoids, to the extent possible, converting a zero element to a nonzero element and thereby decreasing the sparsity of the matrix. It is desirable for the traffic demand matrix to be sparse, since the BvND of a sparse matrix generally contains a smaller number of configurations than that of a dense one, resulting in a smaller total reconfiguration delay. Finally, we note that all stuffing algorithms, including QuickStuff, guarantees to be slack-free in the sense the maximum row or column sum of $D^{(QS)}$ is the same as that of $D^{(Q)}$ (so the net transmission time remains unchanged after the stuffing). We refer readers to [12] for a detailed description of QuickStuff.

The last step is the BvND of $D^{(QS)}$. Now since $D^{(QS)}$ is an s-integer scaled doubly stochastic matrix, it can be expressed as a linear combination of at most $O(n)$ (instead of $O(n^2)$) permutation matrices, whose corresponding durations are s-integers, as follows. Suppose for the moment that Algorithm 1 is used for the decomposition. Then all coefficients α_k, $k = 1, 2, \cdots, K$, are positive s-integers for the following reason. In the first iteration of Algorithm 1, the coefficient α_1 computed from Line 4 must be a positive s-integer, as it is the minimum of a set of s-integer edge weights, so $D^{(QS)}$ remains an s-integer matrix after the subtraction of $\alpha_1 P_1$ from it. Hence α_2 is also a positive s-integer, and so on. Since each $\alpha_k P_k$ takes away at least weight s from any row or column in $D^{(QS)}$, whose sum is upper-bounded by $1 + ns$ as explained earlier, there are at most $(ns + 1)/s$ configurations in the BvND of $D^{(QS)}$. Since s is set to $\Omega(1/n)$ or larger, as will be explained in §IV-C, the number of configurations K is at most $O(n)$.

To further reduce the total number of configurations K, we use the max-min BvND algorithm [21], to be described in § IV-B, instead of Algorithm 1, to decompose $D^{(QS)}$. The resulting K is also $O(n)$, but with a much smaller constant factor. We will show in §IV-B that the computational complexity of the last step (max-min BvND) is $O(n^{3.5})$. Since it dominates the complexities of the other two steps, which are both $O(n^2)$, the overall complexity of QBvND is $O(n^{3.5})$.

Finally, to use QBvND for hybrid switching, only the following slight modification needs to be made to its last step (max-min BvND): the iterative process of searching for max-min matchings can stop as soon as what remains of the quantized stuffed matrix $D^{(QS)}$ can be handled by the packet switch (i.e., constraint (2)).

B. A Modified Max-Min BvND Algorithm

In this section, we describe the last step of our QBvND solution: max-min BvND [21]. The max-min BvND algorithm differs from Algorithm 1 only in that, in each iteration (say k^{th}), whereas the latter finds an arbitrary full matching P_k (Line 4), the former finds a P_k that maximizes the value of the corresponding α_k. Such a matching is called max-min matching because α_k is the minimum among the weights of the edges in P_k, and we are trying to maximize this minimum. As a result, max-min BvND can extract most or all of the traffic from the traffic demand matrix using much fewer configurations than almost all other BvND algorithms.

Now we describe the algorithm, proposed in [22], for computing a max-min matching P in the weighted bipartite graph that corresponds to a nonnegative matrix M. With a slight abuse of notation, we denote this bipartite graph also as M. This algorithm can be best explained using the following alternative formulation of this computation problem. Consider the pruning of the graph M according to a threshold $\epsilon > 0$ as follows: an edge is removed from the graph M if and only if its weight is less than ϵ. We denote the resulting pruned graph as M_ϵ. This computing problem can be restated as finding the maximum ϵ for M_ϵ to contain a full matching (which is exactly the max-min matching P we are looking for). This algorithm is simply to binary-search for this ϵ in the interval $[0, \epsilon_{max}]$, where ϵ_{max} is the value of the largest element in the matrix M, as follows: at each binary search point ϵ', the search is considered successful if a full matching can be found using the classical $O(n^{2.5})$ maximum cardinality matching (MCM) algorithm [25]. The computational complexity of finding a single max-min matching is $O(n^{2.5} \log n)$, since $\log(\epsilon_{max})$ binary search steps are needed, each of which involves a MCM computation, and $\log(\epsilon_{max})$ is $O(\log n)$ in our context.

Using this max-min matching algorithm, the total complexity of the max-min BvND step in QBvND would be $O(n^{3.5} \log n)$, since it needs to run this algorithm at most $O(n)$ times thanks to the quantization. In QBvND, by doing away with the binary search, we reduce this complexity further by a factor of $O(\log n)$ as follows. The modified algorithm conducts a linear search, starting from the value of the largest matrix element in $D^{(QS)}$ (also denoted as ϵ_{max}), which is an s-integer no larger than $1+s$ (since the original traffic demand matrix D is sub-stochastic), for all $O(n)$ max-min matchings. In other words, it searches for and extracts full matchings *exhaustively* (i.e., until a full matching can no longer be found) at each of the $O(n)$ arithmetically progressing graph-pruning thresholds ϵ_{max}, $\epsilon_{max} - s$, $\epsilon_{max} - 2s$, $\cdots$, $2s$, and s.

Since the modified algorithm conducts at most $O(n)$ successful searches, each of which results in a max-min matching and has complexity $O(n^{2.5})$, and at most $O(n)$ unsuccessful searches (once at each of the $O(n)$ graph-pruning thresholds), its total complexity is $O(n^{3.5})$. In QBvND, we also significantly reduce the constant factor inside this big-O, by increasing the unit step of the linear search (i.e., the arithmetic progression) from s to $5s$. In doing so, we observe only a negligible degradation in the qualities (i.e., the durations and the number) of the resulting configurations (matchings). Finally, we note this linear search trick can be used here only because the matrix $D^{(QS)}$ is s-integral. Otherwise, the algorithm would have to linearly search through all distinct values, in the decreasing order, among the nonzero matrix elements (in general $O(n^2)$ of them), resulting in a total complexity of $O(n^{4.5})$.

C. Theoretical Analysis and Quantization Unit Selection

In this section, we explain how to approximately maximize the throughput performance (or equivalently minimize the transmission time) of QBvND by tuning the quantization unit s. The (gross) transmission time of an optical switch schedule can be split into two parts: the *net* transmission time $\sum_{k=1}^{K} \alpha_k$ and the reconfiguration cost $K\delta$. As explained in §IV-A, in optical switching, the first part is upper-bounded by $1 + ns$, and K is upper-bounded by $(1 + ns)/s$, so the second part is upper-bounded by $(1+ns)\delta/s$. Hence the (gross) transmission time is upper-bounded by $1+ns+\frac{1+ns}{s}\delta = 1+n\delta+ns+\delta/s$, which reaches the minimum when $s = \sqrt{\delta/n}$. Although this upper bound is not exactly the objective function we would like to minimize, it is reasonable to use this optimal parameter setting $\sqrt{\delta/n}$ "asymptotically", that is, to set s to $\beta\sqrt{\delta/n}$ where $\beta > 0$ is a tunable parameter.

It remains to explain how to set this β. In optical switching, the second part $K\delta$ (upper-bounded by $(1 + ns)\delta/s$) is quite large so we would like to make s larger to reduce it. In this case, we set β to a value larger than 1 (e.g., set to $\sqrt{2}$ in §VI). In hybrid switching, however, the second part becomes smaller since the packet switch can absorb a residue matrix that would otherwise have to be swept clean by a fairly large number of configurations with short durations. Hence, it becomes less important to reduce the second part by increasing s. In this case, we choose $\beta < 1$ (e.g., set to 0.3 in §VI).

V. Closely Related Works

In this section, we describe several (precomputed) packet, optical, and hybrid switching algorithms that are most related to QBvND and we will evaluate QBvND against in §VI. We will also compare their computational complexities against that of QBvND in §V-D.

A. Precomputed Packet Switching Algorithms

We are aware of two other quantized BvND algorithms, namely RQ (Rate Quantization) [29] and FBD (Frame-Based Decomposition) [30]. Both were proposed, more than a decade ago, for precomputed packet switching (PPS). They both address the following problem with Chang et. al's original solution (i.e., the stuffed BvND algorithm described in §III-B) [26] for PPS: among the numerous (more specifically $O(n^2)$) configurations $\{(P_k, \alpha_k)\}_{k=1}^{K}$ contained in the resulting decomposition, most of them are very short in duration (i.e., with a tiny α_k), and moreover much shorter than a switching cycle of the packet switch (called a frame in FBD). However, each of these short configurations still has to be covered by a full switching cycle, resulting in a poor utilization of the packet switch bandwidth.

In both RQ and FBD, the sole purpose of quantization is to reduce the number of such severely underutilized switching cycles. Although as an unintended side benefit of the quantization, both algorithms also reduce the number of configurations in the resulting decomposition, this number is still too large for bandwidth-efficient optical switching, as we will show in §VI-C3. Note this is not a shortcoming of either algorithm: there is no incentive to reduce the number of configurations in packet switching, where the reconfiguration delay is zero.

B. Optical Switching Algorithms

Scheduling of optical switch alone has been studied for decades. In early works the reconfiguration delay is often assumed to be either zero [14], [15], [11] or infinity (a very large value) [15], [16], [17]. In later works, such as DOUBLE [15], ADJUST [18] and [16], [19], the reconfiguration delay is usually assumed to be finite and nonzero. Towles et al. [15] proposed three different scheduling algorithms, EXACT, MIN, and DOUBLE, for the optical switches with zero, infinite, and nonzero finite reconfiguration delays respectively. EXACT is identical to the stuffed BvND algorithm described in §III-B, which is very bandwidth-inefficient for optical switching when the reconfiguration delay is high, due to the very large number ($O(n^2)$ in general) of configurations it has to use.

MIN is proposed to significantly reduce the number of configurations to at most n. Its algorithm is identical to Algorithm 1 except that in Line 4, α_k is set to the maximum (instead of the minimum in Algorithm 1) among the weights of edges in P_k. This way, after the subtraction of $\alpha_k P_k$, n matrix elements become zero. Hence only n configurations are needed to "cover" the matrix. The flip side of the coin however is that most of the n input-output connections in such a configuration can be severely underutilized (i.e., contain considerable slack), as their corresponding edge weights can be much smaller than this maximum.

DOUBLE is a compromise between the two extremes EXACT and MIN. Each schedule computed by DOUBLE uses at most $2n$ configurations, each of which has the same duration $1/n$, to "cover" a doubly sub-stochastic traffic demand matrix. The algorithm is named DOUBLE because the total duration of these configurations is bounded at 2 ($= 2n * 1/n$), and any set of configurations that "cover" a doubly sub-stochastic matrix has a total duration of at least 1. In DOUBLE, each matrix element D_{ij} is rounded down to the closest $1/n$-integer $\lfloor nD_{ij} \rfloor / n$ called the quotient, and their difference is called the residue. This way, the matrix D is split into a quotient matrix and a residue matrix. Each matrix can be covered by at most n configurations, each of which has duration $1/n$.

ADJUST [18] is different than DOUBLE only in that ADJUST sets the value of quantization unit s to $\sqrt{\delta/n}$ (like we did in §IV-C) to strike the optimal compromise between EXACT and MIN. ADJUST is the same as DOUBLE when the configuration delay δ is equal to $1/n$, since both are using the same quantization unit $\sqrt{\delta/n} = 1/n$ in this case.

C. Hybrid Switching Algorithms

Liu et al. [12] first formulated the mathematical problem of the hybrid switch scheduling and proposed a greedy heuristic solution, called Solstice. Like QBvND, Solstice is also a BvND-based algorithm, and it also attempts to find and extract a max-min matching in each iteration. However, different than QBvND, Solstice does not quantize the demand matrix D, and to reduce its computational complexity also by a factor of $\log(n)$, Solstice settles with an approximate max-min matching (in each iteration) whose weight is at least half of that of an actual one.

Different than Solstice and QBvND, Eclipse [13] is not a BvND-based algorithm. In each iteration, Eclipse attempts to extract and subtract a configuration (P, α) from the demand matrix D that maximizes a cost-adjusted utility function. However, this optimization problem is very computationally expensive: Even a much more efficient heuristic used instead in [13] has to perform $4 \log n$ computationally expensive *maximum weighted matching* (MWM) computations [31] to arrive at such a (approximate) utility-maximizing matching. This high computational cost will be highlighted in Table I.

Finally, we note neither Solstice nor Eclipse is suitable for optical switching for the following reason. Both are designed and optimized only for identifying a relatively small number of configurations with relatively long durations that the optical switch can use to remove the bulk of D. If we were to run both to the "bitter end" (i.e., until D is a zero matrix), as we would have to in optical switching, both would produce a large number of configurations with tiny durations, and hence incur a very high reconfiguration cost.

D. Computational Complexity Comparisons

Table I summarizes the computational complexities of the optical and the hybrid switching algorithms that QBvND will be compared against in §VI. In the complexities of Solstice and Eclipse, K denotes the total number of configurations in the schedule. Although K is not too large (roughly $O(n)$) in our simulation scenarios, it could be as large as $O(n^2)$ in the worst case. Hence the worst case complexities of Solstice and Eclipse are both higher than that of QBvND, which is $O(n^{3.5})$. Compared to optical switching algorithms, although QBvND has a higher computational complexity than

TABLE I
COMPLEXITIES OF VARIOUS ALGORITHMS

	Algorithms	Complexities
Optical Switching	DOUBLE [15]	$O(n^2 \log n)$
	ADJUST [18]	$O(n^2 \log n)$
	MIN [15]	$O(n^{3.5})$
	EXACT [15]	$O(n^{4.5})$
Hybrid Switching	Solstice [12]	$O(Kn^{2.5})$
	Eclipse [13]	$O(Kn^{2.5} \log^2 n)$
Precomputed Packet Switching	BvND [26]	$O(n^{4.5})$
	FBD [30]	$O(n^2 \log n)$
	RQ [29]	$O(n^{4.5})$

DOUBLE and ADJUST, it outperforms all four of them, in terms of throughput performance, by a wide margin, as will be shown in §VI.

VI. EVALUATION

In this section, we evaluate the performance of QBvND both as an optical switching solution (§ VI-C) and as a hybrid switching solution (§VI-D). We compare QBvND with other optical and hybrid switching solutions, under various system parameter settings and traffic demands. For all these comparisons, we use the transmission time $K\delta + \sum_{k=1}^{K} \alpha_k$ (needed to transmit D) as the performance metric.

A. Traffic Demand Matrix D

As shown in [12], [13], the typical workloads we see in data centers exhibit two characteristics: sparsity (the vast majority of the demand matrix elements have value 0 or close to 0) and skewness (few large elements in a row or column account for the majority of the row or column sum). Hence, for our simulations, we use the same set of sparse and skewed demand matrices as used in [12], [13]. In each such matrix D, each row (or column) contains n_L large equal-valued elements (large input-output flows) that as a whole account for c_L (percentage) of the total workload to the row (or column), n_S medium equal-valued elements (medium input-output flows) that as a whole account for the rest $c_S = 1 - c_L$ (percentage), and noises. Hence n_L and n_S control the sparsity, and c_L and c_S control the skewness, of the traffic demand, respectively. Roughly speaking, we have

$$D = \sum_{i=1}^{n_L} \frac{c_L}{n_L} P_i + \sum_{i=1}^{n_S} \frac{c_S}{n_S} P_i' + \mathcal{N} \quad (4)$$

where each P_i and each P_i' is an $n \times n$ random permutation matrix.

Same as in [12], [13], in our simulation studies, the default values of the sparsity parameters n_L and n_S are set to 4 and 12 respectively and the default values of c_L and c_S are set to 0.7 (*i.e.*, 70%) and 0.3 (*i.e.*, 30%) respectively. In other words, in each row (or column) of the demand matrix, by default the 4 large flows account for 70% of its total traffic demand, and the 12 medium flows account for the rest 30%. We will also vary the sparsity parameters n_L and n_S and skewness parameters c_L and c_S in our evaluations. In Equation (4), before a noise matrix $\mathcal{N}$ (described next) is added to it, each such D is doubly stochastic, that is, the sum of each row or column of it is 1. This normalized workload 1 is defined as the amount of traffic that an *established* (after paying for the reconfiguration cost) optical switch connection/link, which we assume to have a normalized rate also of 1, can transmit in 1 unit of time, defined as the length of a scheduling epoch (e.g., 3 milliseconds).

As shown in Equation (4), we also add a noise matrix term $\mathcal{N}$ to D, like in [12], [13]. Each nonzero element in $\mathcal{N}$ is a Gaussian random variable that is added to a traffic demand matrix element that was nonzero before the noise added. Each nonzero (noise) element here in $\mathcal{N}$ has a standard deviation, which is equal to 0.3% of the normalized workload 1.

B. System Parameters

In this section, we introduce the system parameters, for both optical and hybrid switching, used in our simulations.

Network size: By default, both the optical switch and the packet switch (if applicable) has $n = 100$ input/output ports (i.e., 100 racks of servers in the data center) although we will vary n in §VI-D3. Other reasonably large (say ≥ 32) switch sizes produce similar results.

Reconfiguration delay of the optical switch δ: In both optical and hybrid switching, the larger this reconfiguration delay is, the more time the optical switch has to spend on reconfigurations, and hence the higher the transmission time is. By default, $\delta = 0.01$ (i.e., $1/100$ of the scheduling epoch), although we will vary δ in our simulation studies.

Optical switch per-port rate $r_c = 1$ and packet switch per-port rate r_p: As far as designing hybrid switching algorithms is concerned, only their ratio r_c/r_p matters. The higher this ratio is, the higher percentage of traffic should be transmitted by the optical switch. This ratio varies from 8 to 40 in our simulations. As explained earlier, we normalize r_c to 1 throughout this paper.

The largest row or column sum in a (random) demand matrix D generated using (4), also a random variable, has an expectation that is roughly equal to 1.0325, where the fractional part 0.0325 comes from the noise matrix $\mathcal{N}$. In optical switching, since $r_c = 1$, even with perfect scheduling and zero reconfiguration delay, the total transmission time is at least 1.0325. In hybrid switching, the transmission time could be smaller than 1, thanks to the switching and transmission capacity of the packet switch, although that never materialized in our simulation studies, likely due to the nontrivial reconfiguration delays.

In the rest of §VI, every point in every plot in every figure is the sample mean averaged from 100 simulation runs, so is every number in Table II and Table III.

C. QBvND vs. Others for Optical Switching

In this section, we compare QBvND, as an optical switching solution, with three state of the art optical switching algorithms: ADJUST [18], DOUBLE [15], and MIN [15]. Note

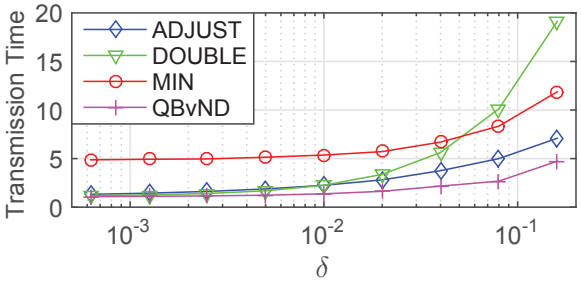

Fig. 2. Comparison while varying the reconfiguration delay (optical)

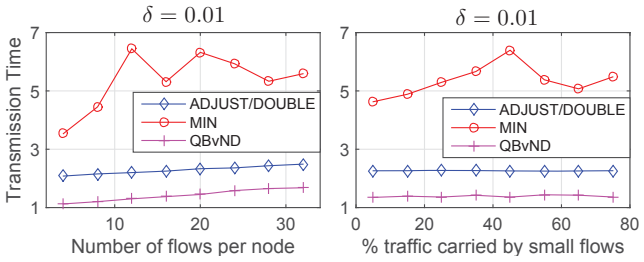

Fig. 3. Comparison under various demand matrices (optical)

that when $\delta = 0.01$ (i.e., $\delta = 1/n$ since $n = 100$), ADJUST is the same as DOUBLE, as we explained in § V-B. Hence there is a combined plot for ADJUST/DOUBLE in Figure 3.

1) Performances under Different System Parameters: In this section, we evaluate the performances of ADJUST, DOUBLE, MIN, and QBvND for different values of δ under the traffic demand matrix with the default parameter settings (4 large flows and 12 small flows accounting for roughly 70% and 30% of the total traffic demand into each input port). The simulation results, shown in Figure 2, demonstrate that the schedules generated by QBvND are consistently better (*i.e.*, shorter transmission times) than those generated by ADJUST, DOUBLE, and MIN. More specifically, when $\delta = 0.01$ (default setting) and $\delta = 0.04$, the average transmission times of the schedules generated by QBvND are roughly 40% shorter than those of schedules generated by ADJUST, the best among others.

2) Performances under Different Traffic Demands: We also evaluate the performance of QBvND under a large set of traffic demand matrices that vary by sparsity and skewness. Figure 3 (left) compares the transmission times under ADJUST, DOUBLE, MIN, and QBvND when the sparsity parameter $n_L + n_S$ varies from 4 to 32 (while n_L/n_S is fixed at 1/3) while the value of the skewness parameter c_S is fixed at the default value 0.3. Figure 3 (right) compares the transmission times under these four algorithms when the the skewness parameter c_S varies from 5% to 75% while the sparsity parameter $n_L + n_S$ is fixed at the default value 16 (= 4+12). In all these evaluations, the reconfiguration delay δ is set to the default value of 0.01. Figure 3 shows that schedules computed by QBvND have an average transmission time that is roughly 40% shorter than that computed by ADJUST, the best scheduler among others.

3) An "Anatomic" Comparison of Transmission Time: To better understand the reason why the QBvND outperforms all other optical switching algorithms by a wide margin, we split the (gross) transmission time into the aforementioned two components: the net transmission time $\sum_{k=1}^{K} \alpha_k$ and the reconfiguration cost $K\delta$. Table II separately compares these two components in schedules computed by different algorithms under the default setting (4 large flows and 12 small flows accounting for roughly 70% and 30% of the total traffic demand into each input port, $\delta = 0.01$). Besides ADJUST, DOUBLE and MIN, we compare QBvND also with three other algorithms: FBD [30], RQ [29], and EXACT [15]. Note that FBD and RQ are proposed originally for precomputed packet switching, but here we view them as optical switching algorithms and impose the reconfiguration delay on them. ADJUST is not in the table since it is equivalent to DOUBLE under this parameter setting. The simulation results are summarized in Table II.

TABLE II
TRANSMISSION TIME COMPARISON OF OPTICAL SWITCHING ALGORITHMS

Algorithm	$K\delta$	$\sum_{k=1}^{K} \alpha_k$	$K\delta + \sum_{k=1}^{K} \alpha_k$
DOUBLE	1.1245	1.1245	2.2490
MIN	0.4403	4.8337	5.2740
EXACT	12.98	1.0325	14.0125
FBD	0.8329	1.1697	2.0026
RQ	1.7252	2.0185	3.7437
QBvND	0.2294	1.1457	1.3751

As expected, the average net transmission time $\sum_{k=1}^{K} \alpha_k$ under EXACT is equal to 1.0325, the aforementioned theoretical minimum, since it is precisely the stuffed BvND algorithm (described in § III-B). However, its reconfiguration cost is humongous (12.98), due to the large number of configurations in the resulting BvND. The net transmission time under QBvND (1.1457) is only slightly larger than the theoretical minimum 1.0325 and smaller than that of all other algorithms. Its reconfiguration cost (0.2294) is smaller, by at least 70%, than that of all other algorithms except MIN, whose net transmission time (4.8337) is extremely high.

D. QBvND vs. Solstice and Eclipse for Hybrid Switching

In this section, we compare QBvND, as a hybrid switching solution, with the two state of the art algorithms, Eclipse [13] and Solstice [12].

1) Performances under Different System Parameters: In this section, we evaluate the performances of Eclipse, Solstice, and QBvND for different value combinations of δ and r_c/r_p under the traffic demand matrix with the default parameter settings (4 large flows and 12 small flows accounting for roughly 70% and 30% of the total traffic demand into each input port). We set r_c/r_p and δ to their respective default values 10 and 0.01 in Figure 4(left) and Figure 4(right) respectively. The simulation results demonstrate that the schedules generated by QBvND are better than those generated by Solstice, especially

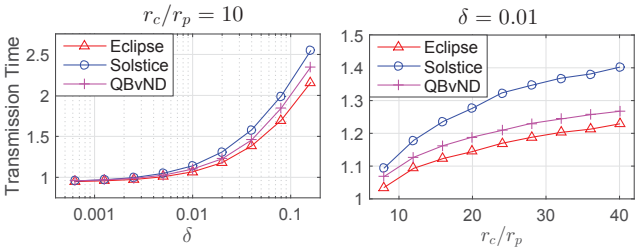

Fig. 4. Comparison under different system settings (hybrid)

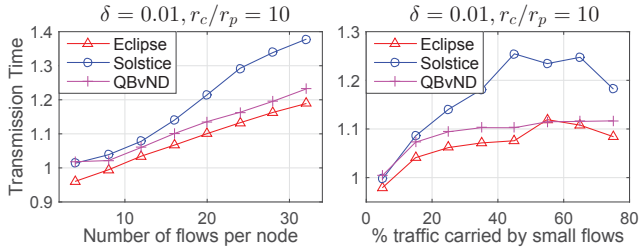

Fig. 5. Comparison under various demand matrices (hybrid)

when the reconfiguration delay δ or the rate ratio r_c/r_p is large. More specifically, as shown in Figure 4(left), when $\delta = 0.04, r_c/r_p = 10$, QBvND results in 8% shorter average transmission time than Solstice; as shown in Figure 4(right), when $\delta = 0.01, r_c/r_p = 32$, QBvND results in 11% shorter average transmission time than Solstice. On the other hand, QBvND results in slightly longer (approximately 5% longer) average transmission times in both cases than Eclipse.

2) Performances under Different Traffic Demands: In this section, we evaluate the performance of QBvND under the same set of traffic demand matrices used in §VI-D2 that vary by sparsity and skewness. The simulation results are shown in Figure 5. In both subfigures, we set δ and r_c/r_p to their respective default values 0.01 and 10. Like Figure 4, Figure 5 shows that QBvND results in shorter average transmission times than Solstice, and slightly longer average transmission times than Eclipse, under various traffic demand matrices.

3) Execution Time Comparison: In this section, we present the execution times of Eclipse, Solstice, and QBvND (all implemented in C++) for three different δ values (0.0025, 0.01, and 0.04), under the traffic demand matrix with the default parameter settings ($n_L = 4$, $n_S = 12$, $c_L = 0.7$, $c_S = 0.3$). We set r_c/r_p to 10 in each scenario. These execution time measurements, shown in Table III, are performed on a Dell Precision Tower 3620 workstation equipped with an Intel Core i7-6700K CPU @4.00GHz processor with 16GB RAM, and running Windows 10 Professional.

As shown in Table III, the average execution time of QBvND is roughly 20 times smaller than that of Eclipse under the default setting ($n = 100$). As n increases to 200, QBvND outperforms Eclipse even more (e.g., 100 times when $\delta = 0.01$) in terms of execution time. Meanwhile, QBvND's average execution time is only slightly larger than that of

TABLE III
COMPARISON OF AVERAGE EXECUTION TIME

	δ	Eclipse	QBvND	Solstice
$n = 50$	0.0025	$152.3ms$	$52.75ms$	$33.23ms$
	0.01	$141.5ms$	$24.90ms$	$33.83ms$
	0.04	$128.8ms$	$19.45ms$	$25.95ms$
$n = 100$	0.0025	$1.430s$	$111.67ms$	$76.07ms$
	0.01	$1.282s$	$55.65ms$	$72.67ms$
	0.04	$0.943s$	$48.15ms$	$59.56ms$
$n = 200$	0.0025	$17.57s$	$263.6ms$	$182.8ms$
	0.01	$12.37s$	$139.1ms$	$165.8ms$
	0.04	$7.384s$	$111.2ms$	$134.0ms$

Solstice when $\delta = 0.0025$. When $\delta = 0.01, 0.04$, QBvND is even faster than Solstice.

VII. Other Related Works

A. Optical Switching Algorithms

Recently, a solution called Adaptive MaxWeight (AMW) [32], [33] was proposed for optical switches (with nonzero reconfiguration delays). The basic idea of AMW is that when the maximum weighted configuration (matching) has a much higher weight than the current configuration, the optical switch is reconfigured to the maximum weighted configuration; otherwise, the configuration of the optimal switch stays the same. However, this algorithm may lead to long queueing delays (for packets) since it usually reconfigures infrequently.

B. Hybrid Switching Algorithms

The hybrid switching problem has also been considered in two other works, namely ProjecToR [34] and FireFly [35]. Their problem formulations are a bit different than that in Solstice [12] and Eclipse [13], and so are their solution approaches. In ProjecToR [34], the problem of matching senders with receivers is modeled as a (distributed) stable marriage problem, in which a sender's preference score for a receiver is equal to the age of the data the former has to transmit to the latter in a scheduling epoch, and is solved using a variant of the Gale-Shapely algorithm [36]. This solution is aimed at minimizing transmission latencies while avoiding starvation, and not at maximizing network throughput, or equivalently minimizing transmission time. The innovations of FireFly [35] are mostly in the aspect of systems building and not on matching algorithm designs.

All hybrid switching algorithms we have described so far consider and allow only direct routing in the sense all optical-switched data packets reach their respective final destinations in one-hop (*i.e.*, enters and exits the optical switch only once). Only two hybrid switching algorithms, namely Eclipse++ [13] and Albedo [24], have explored indirect routing. To do so, however, both algorithms have to perform a large number of single-source shortest-path computations, and hence appear to be extremely computationally expensive.

VIII. CONCLUSION

While researchers have proposed multiple algorithms for either optical switching or hybrid switching, none of these solutions works particularly well, especially if we have a nontrivial reconfiguration delay. In this work, we propose QBvND, a scheduling algorithm that combines matrix quantization and the max-min Birkhoff-von Neumann decomposition techniques. Comparing to the state of the art solutions, QBvND achieves a much better performance for the optical switching problem and strikes a much better tradeoff between the resulting performance of the hybrid switch and the computational complexity of the algorithm for the hybrid switching problem.

ACKNOWLEDGE

This work is supported in part by US NSF through award CNS 1423182.

REFERENCES

[1] C. DeCusatis, "Optical interconnect networks for data communications," *J. Lightw. Technol.*, vol. 32, no. 4, pp. 544–552, 2014.

[2] P. Patel, D. Bansal, L. Yuan, A. Murthy, A. Greenberg, D. A. Maltz, R. Kern, H. Kumar, M. Zikos, H. Wu *et al.*, "Ananta: cloud scale load balancing," in *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4. ACM, 2013, pp. 207–218.

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

[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," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 339–350, 2010.

[5] X. Ye, Y. Yin, S. B. Yoo, P. Mejia, R. Proietti, and V. Akella, "Dos: A scalable optical switch for datacenters," in *Proceedings of the 6th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*. ACM, 2010, p. 24.

[6] A. Singla, A. Singh, K. Ramachandran, L. Xu, and Y. Zhang, "Proteus: a topology malleable data center network," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010, p. 8.

[7] K. Xi, Y.-H. Kao, and H. J. Chao, "A petabit bufferless optical switch for data center networks," in *Optical interconnects for future data center networks*. Springer, 2013, pp. 135–154.

[8] O. Liboiron-Ladouceur, I. Cerutti, P. G. Raponi, N. Andriolli, and P. Castoldi, "Energy-efficient design of a scalable optical multiplane interconnection architecture," *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 17, no. 2, pp. 377–383, 2011.

[9] K. Chen, A. Singla, A. Singh, K. Ramachandran, L. Xu, Y. Zhang, X. Wen, and Y. Chen, "OSA: An optical switching architecture for data center networks with unprecedented flexibility," *IEEE/ACM Trans. Netw.*, vol. 22, no. 2, pp. 498–511, 2014.

[10] H. Liu, F. Lu, A. Forencich, R. Kapoor, M. Tewari, G. M. Voelker, G. Papen, A. C. Snoeren, and G. Porter, "Circuit switching under the radar with reactor." in *NSDI*, vol. 14, 2014, pp. 1–15.

[11] 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," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 447–458, Aug. 2013.

[12] H. Liu, M. K. Mukerjee, C. Li, N. Feltman, G. Papen, S. Savage, S. Seshan, G. M. Voelker, D. G. Andersen, M. Kaminsky, G. Porter, and A. C. Snoeren, "Scheduling techniques for hybrid circuit/packet networks," in *ACM CoNEXT*, ser. CoNEXT '15. New York, NY, USA: ACM, 2015, pp. 41:1–41:13.

[13] S. Bojja Venkatakrishnan, M. Alizadeh, and P. Viswanath, "Costly circuits, submodular schedules and approximate carathéodory theorems," in *SIGMETRICS*. ACM, 2016, pp. 75–88.

[14] T. Inukai, "An efficient SS/TDMA time slot assignment algorithm," *IEEE Trans. Commun.*, vol. 27, no. 10, pp. 1449–1455, 1979.

[15] B. Towles and W. J. Dally, "Guaranteed scheduling for switches with configuration overhead," *IEEE/ACM Transactions on Networking*, vol. 11, no. 5, pp. 835–847, 2003.

[16] B. Wu and K. L. Yeung, "Nxg05-6: Minimum delay scheduling in scalable hybrid electronic/optical packet switches," in *GLOBECOM*. IEEE, 2006, pp. 1–5.

[17] I. Gopal and C. Wong, "Minimizing the number of switchings in an ss/tdma system," *IEEE Trans. Commun.*, vol. 33, no. 6, pp. 497–501, 1985.

[18] 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.

[19] S. Fu, B. Wu, X. Jiang, A. Pattavina, L. Zhang, and S. Xu, "Cost and delay tradeoff in three-stage switch architecture for data center networks," in *HPSR*. IEEE, 2013, pp. 56–61.

[20] D. Birkhoff, "Tres observaciones sobre el algebra lineal," *Universidad Nacional de Tucuman Revista , Serie A*, vol. 5, pp. 147–151, 1946.

[21] F. Dufossé and B. Uçar, "Notes on birkhoff–von neumann decomposition of doubly stochastic matrices," *Linear Algebra and its Applications*, vol. 497, pp. 108–115, 2016.

[22] I. S. Duff and J. Koster, "The design and use of algorithms for permuting large entries to the diagonal of sparse matrices," *SIAM Journal on Matrix Analysis and Applications*, vol. 20, no. 4, pp. 889–901, 1999.

[23] ——, "On algorithms for permuting large entries to the diagonal of a sparse matrix," *SIAM Journal on Matrix Analysis and Applications*, vol. 22, no. 4, pp. 973–996, 2001.

[24] C. Li, M. K. Mukerjee, D. G. Andersen, S. Seshan, M. Kaminsky, G. Porter, and A. C. Snoeren, "Using indirect routing to recover from network traffic scheduling estimation error," in *ANCS*. IEEE Press, 2017, pp. 13–24.

[25] J. E. Hopcroft and R. M. Karp, "An n^5/2 algorithm for maximum matchings in bipartite graphs," *SIAM Journal on computing*, vol. 2, no. 4, pp. 225–231, 1973.

[26] C.-S. Chang, W.-J. Chen, and H.-Y. Huang, "On service guarantees for input-buffered crossbar switches: a capacity decomposition approach by birkhoff and von neumann," in *Quality of Service, 1999. IWQoS'99. 1999 Seventh International Workshop on*. IEEE, 1999, pp. 79–86.

[27] J. Von Neumann, "A certain zero-sum two-person game equivalent to the optimal assignment problem," *Contributions to the Theory of Games*, vol. 2, pp. 5–12, 1953.

[28] J. Chou and B. Lin, "Birkhoff-von neumann switching with statistical traffic profiles," *Computer Communications*, vol. 33, no. 7, pp. 848–851, 2010.

[29] C. E. Koksal, R. G. Gallager, and C. E. Rohrs, "Rate quantization and service quality over single crossbar switches," in *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 3. IEEE, 2004, pp. 1962–1973.

[30] B. Lin and I. Keslassy, "A scalable switch for service guarantees," in *High Performance Interconnects, 2005. Proceedings. 13th Symposium on*. IEEE, 2005, pp. 93–99.

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

[32] C.-H. Wang, T. Javidi, and G. Porter, "End-to-end scheduling for all-optical data centers," in *INFOCOM*. IEEE, 2015, pp. 406–414.

[33] C.-H. Wang, S. T. Maguluri, and T. Javidi, "Heavy traffic queue length behavior in switches with reconfiguration delay," *arXiv preprint arXiv:1701.05598*, 2017.

[34] M. Ghobadi, R. Mahajan, A. Phanishayee, N. Devanur, J. Kulkarni, G. Ranade, P.-A. Blanche, H. Rastegarfar, M. Glick, and D. Kilper, "Projector: Agile reconfigurable data center interconnect," in *SIGCOMM*, 2016, pp. 216–229.

[35] N. Hamedazimi, Z. Qazi, H. Gupta, V. Sekar, S. R. Das, J. P. Longtin, H. Shah, and A. Tanwer, "Firefly: A reconfigurable wireless data center fabric using free-space optics," in *Proceedings of the ACM SIGCOMM*, 2014, pp. 319–330.

[36] D. Gale and L. S. Shapley, "College admissions and the stability of marriage," *The American Mathematical Monthly*, vol. 69, no. 1, pp. 9–15, 1962.