# Composite-Path Switching

Shay Vargaftik[1,2], Katherine Barabash[1], Yaniv Ben-Itzhak[1], Ofer Biran[1],
Isaac Keslassy[2,3], Dean Lorenz[1], Ariel Orda[2]
[1] IBM Research [2] Technion [3] VMware Research

## ABSTRACT

Hybrid switching combines a high-bandwidth optical circuit switch in parallel with a low-bandwidth electronic packet switch. It presents an appealing solution for scaling datacenter architectures. Unfortunately, it does not fit many traffic patterns produced by typical datacenter applications, and in particular the skewed traffic patterns that involve highly intensive one-to-many and many-to-one communications.

In this paper, we introduce *composite-path switching* by allowing for composite circuit/packet paths between the two switches. We show how this enables the datacenter network to deal with skewed traffic patterns, and offer a practical scheduling algorithm that can directly extend any hybrid-switching scheduling algorithm. Through extensive evaluations using modern datacenter workloads, we show how our solution outperforms two recently proposed state-of-the-art scheduling techniques, both in completion time and in circuit utilization.

## CCS Concepts

**•Networks → Bridges and switches; Data center networks; Packet scheduling; Hybrid networks;**

## Keywords

hybrid networks; composite-path switching; OCS; EPS

## 1. INTRODUCTION

*Hybrid switching* has emerged in recent work as an appealing solution for scaling datacenter architectures [1–11]. As illustrated in Figure 1(a), a hybrid switch typically combines a low-bandwidth electronic packet switch (EPS) and a high-bandwidth optical circuit switch (OCS). The EPS system can send traffic from any port to any port simultaneously at a relatively low speed, and can therefore handle low-bandwidth densely-connected traffic. The OCS system can create arbitrary low-latency high-capacity circuits, using a relatively slow-to-reconfigure cross-board. It handles high-bandwidth flows that require sparse connections.

The hybrid switch has $n$ senders and $n$ receivers, corresponding to either ToR (Top of Rack) switches or individual servers. While the EPS and the OCS systems are modeled as single switches, they could be generalized to multi-layer networks of switches.

Unfortunately, these hybrid switches are not suited to many modern data-parallel datacenter applications, because they cannot efficiently switch their communication patterns. To simplify, these applications can be modeled using the *coflow* abstraction, as a collection of flows with a shared completion time [12–15]. The coflows consist of four main types: (a) *Many-to-many*, including a large number of low-bandwidth point-to-point communications, *e.g.,* data-parallel applications and dataflow pipelines [16, 17]; (b) *One-to-one*, involving a few high-bandwidth point-to-point flows, *e.g.,* distributed file systems [18, 19]; (c) One-to-many, *e.g.,* distributed data storage and backup, query traffic, etc. [16, 20, 21]; and (d) Many-to-one, *e.g.,* aggregation of data (*i.e.,* MapReduce, Partition-Aggregate, etc.) [16, 22–24]. The last two kinds (*i.e.,* (c) and (d)) are often more delay-sensitive [21].

The hybrid switch can deal well with the first two types of coflows: the EPS system with (a) low-bandwidth many-to-many, and the OCS system with (b) high-bandwidth one-to-one. However, (c) one-to-many and (d) many-to-one coflows achieve poor performance in hybrid switches. They are constrained by the low bandwidth of EPS switches and the high reconfiguration times of OCS switches. For instance, the black line in Figure 1(a) shows how a one-to-many coflow that exceeds the EPS capacity would need to use time-division multiplexing over the OCS system, and therefore would achieve a poor performance.

In practice, this poor performance with skewed one-to-many and many-to-one traffic is further compounded

*CoNEXT '16, December 12-15, 2016, Irvine, CA, USA*
© 2016 ACM. ISBN 978-1-4503-4292-6/16/12...$15.00
DOI: http://dx.doi.org/10.1145/2999572.2999610

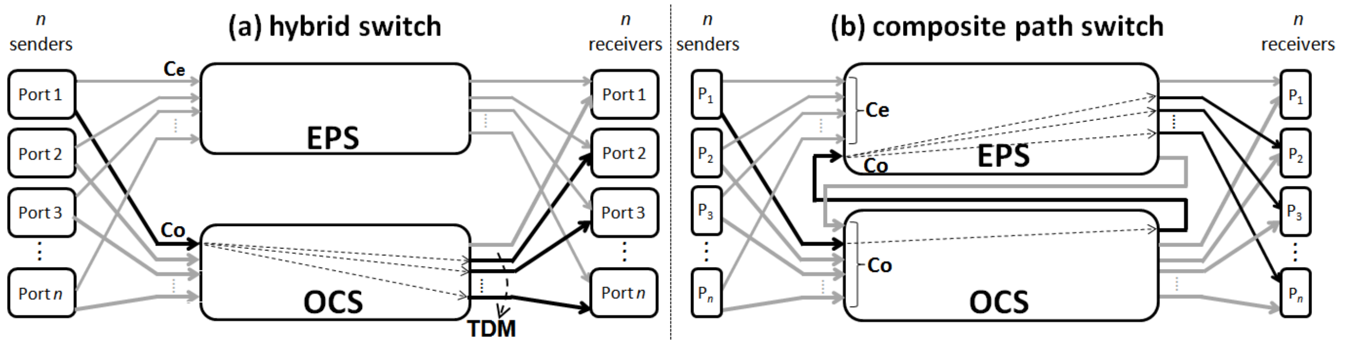


**Figure 1: (a) The hybrid switch model vs. (b) the composite-path switch model. The black line shows one-to-many communication from the first sender to all the other receivers. In (a) the flows are serialized with Time Division Multiplexing (TDM), while in (b) the flows are sent through the OCS over the OCS/EPS composite-path, and then are efficiently demultiplexed by the Packet Switch.**

by the fact that modern workloads are increasingly decomposed into micro-services, often implemented using a large number of identical containers, and interconnected using message brokers as barriers that receive messages from many service endpoints and deliver messages to many other service endpoints. As a result, this multi-component datacenter traffic is even more skewed than each of its components, and its performance acutely depends on the last flow to complete in each coflow [25, 26].

In this paper, we introduce the *composite-path switch* (*cp-Switch*) to deal with skewed one-to-many and many-to-one coflows. As Figure 1(b) illustrates, we propose a small extension to the hybrid switch by interconnecting the EPS and OCS systems using *composite paths*. For instance, to schedule the same black one-to-many coflow, the scheduler can first forward the aggregate traffic though the OCS system, then through the one-to-many composite path to the EPS system. Finally, at the EPS system, the different flows of this coflow are switched to their respective destinations. Thus, this one-to-many coflow can be switched without a need for many costly OCS reconfigurations.

The goal of this paper is to extend hybrid Circuit/Packet switches and allow them to efficiently support modern datacenter workloads that include a large variety of traffic patterns and applications.

First, we introduce the composite-path switch architecture and explain how it can better serve skewed one-to-many and many-to-one datacenter traffic.

Next, we tackle the challenge of simplifying cp-Switch scheduling so as to leverage state-of-the-art hybrid switch scheduling techniques. We show how it is possible to build a reduced cp-Switch demand matrix using an additional column and row that would represent the composite paths. We also introduce intuition about traffic that would fit such composite paths, and deduce constraints on how to filter the demand matrix.

We are then able to apply hybrid-switch scheduling to our problem. However, we still need to process results in order to be able to schedule our cp-Switch. In particular, we explain how to schedule flows within a composite path and why we want to reserve EPS slots.

Finally, using extensive evaluation, we demonstrate that the composite-path switch is capable of accommodating more traffic patterns while significantly improving both the flow completion time and the circuit utilization metrics.

The rest of this paper is structured as follows: In Section 2 we present our composite path Circuit/Packet switching model, and formalize the translation of the known scheduling techniques to our new Circuit/Packet switching model. We follow up by describing our evaluation methodology and presenting the evaluation results in Section 3. In Section 4 we discuss possible extensions and present future research directions for the cp-Switch. Section 5 describes related work, and Section 6 concludes the paper.

## 2. COMPOSITE-PATH SWITCH

Our goal in this section is to provide a scheduling algorithm for our proposed composite-path switch.

### 2.1 Composite-Path Switch Model

We start by defining and modeling our composite-path switch, denoted *cp-Switch*. We largely follow the model for the hybrid switch, denoted *h-Switch* [1, 2].

We already saw in Figure 1(b) how cp-Switch combines an EPS system and an OCS system to connect $n$ input ports to $n$ output ports. Both the EPS and OCS systems implement a matching between their respective input and output ports. While EPS can be reconfigured instantaneously, the circuits of the OCS can be reconfigured with a reconfiguration time penalty $\delta$. Following [1, 2], we assume that during the reconfiguration time no data can pass through the circuit switch.

In addition, EPS has a much lower capacity than OCS (*e.g.,* if the EPS link rate is $C_e$ = 10 Gbps and the OCS link rate is $C_o$ = 100 Gbps, there is a 1-to-10 ratio). Each sender implements per-receiver *Virtual Output Queues* (VOQs) [27, 28]. The occupancy of these VOQs can be used to build the demand matrix. EPS also implements input and output queues. We further classically assume that time is slotted and propagation times are negligible.

**Composite OCS/EPS paths:** The main addition in cp-Switch compared to h-Switch are the composite paths, *i.e.,* two high-bandwidth links connecting the output of each fabric to the input of the other fabric. As mentioned, these composite paths help us deal with skewed datacenter traffic, and especially with one-to-many and many-to-one traffic. These new composite paths require two main architectural abilities.

First, the dedicated use of an additional input/output port pair at each switch to inter-connect the EPS and the OCS systems. In particular, the cp-Switch needs the ability for the EPS fabric to deal with a higher-bandwidth input/output port of capacity $C_o$. Current switches are already offering heterogeneous port bandwidth, with many low-bandwidth and a few high-bandwidth ports. For instance, the Mellanox SN2410 switch [29] offers 48x25GbE and 8x100GbE ports; Arista 7280TR-48C6 [30] offers 48x10GbE and 6x100GbE ports; and Cisco 93180YC-EX [31] offers flexible 48x10/25-Gbps ports and 6x40/100-Gbps ports.

This is also generally related to the well-known issue of what to do with switches when upgrading the link speed.

Second, the ability to synchronize the transmission of subsets of hosts through the EPS upon the creation of a many-to-one composite path. To that end, Precision Time Protocol (PTP) [32] is commonly used in order to achieve sub-microsecond network-wide time synchronization. For instance, Fastpass [33] presents an implementation for fine-grain timing between endpoints and switches by using commodity NICs, which achieves a time accuracy of a few hundred nanoseconds across a multi-hop network. White Rabbit [34], which is used by CERN, provides a sub-nanosecond accuracy, by using a state-of-the-art FPGA-based implementation.

## 2.2 cp-Switch Demand Reduction

We would like to introduce a simple scheduling algorithm for cp-Switch that essentially follows the h-Switch scheduling algorithm approach. Unfortunately, cp-Switch scheduling is not so simple. In the h-Switch, the OCS and EPS configurations can be scheduled in a simple, pipeline-like manner. First, given an $n \times n$ demand matrix $D$, the scheduler decides on the OCS configurations and their respective duration. Next, it selects all unscheduled packets and schedules them through the EPS.

On the other hand, since cp-Switch has composite cross-platform paths, they induce a *strong dependency* between the two fabrics, such that the above pipelined scheme with separate decisions for OCS and EPS cannot be directly applied. As a result, a joint schedule for the EPS, the OCS and the composite paths seems significantly more complex to design.

To deal with this challenge, we pursue the ability to *translate* h-Switch scheduling to cp-Switch scheduling. In addition to making the approach more tractable, this would also enable us to leverage the existing body of research on h-Switch scheduling.

**Demand Reduction:** Our approach is to reduce the $n \times n$ input demand matrix $D$ into a new $(n+1) \times (n+1)$ demand matrix $D_I$, such that $D_I$ could be directly fed to a h-Switch scheduling algorithm, and the resulting schedule could be modified into a viable schedule for the cp-Switch.

To construct the new $(n+1) \times (n+1)$ matrix $D_I$, we augment the demand matrix $D$ by adding a column that represents the one-to-many composite path and an additional row over $D$ that represents the many-to-one composite path. Our scheduler then selects entries in $D$ that it aggregates and puts in the new composite entries, while removing them from the non-composite entries. The new non-zero entries will be served by the composite paths, whereas the rest of the demand matrix is served by the regular EPS-EPS and OCS-OCS paths. Now, when $D_I$ is fed to an h-Switch scheduling algorithm, the resulting permutation matrices can indicate both the creation of regular OCS-OCS circuits and the creation of composite paths (*i.e.,* when the last row or column has an entry of 1, it means that the corresponding sender or receiver has a composite path within the duration of this permutation matrix).

The above reduction helps us use h-Switch scheduling as a sub-routine in cp-Switch scheduling. However, it also introduces a new challenge: the scheduler now needs to decide which demand-matrix entries in $D$ should be aggregated into composite-path entries in $D_I$. Of course, exhaustively considering all possibilities leads to an exponential complexity.

**Demand Filtering:** We adopt a heuristic approach motivated by the following intuition:

**(a)** The traffic along the composite paths should provide high utilization for the OCS while not overwhelming the EPS—*i.e.,* it should serve one-to-many or many-to-one patterns.

**(b)** The time to serve each individual entry by the OCS should not exceed too much the reconfiguration time penalty of the OCS.

The intuition for (a) is relatively straightforward. A demand-matrix row with 1-2 entries would not gain in being aggregated into a composite entry. It would be more efficient to simply service these entries one by one using either OCS or EPS. (b) was more surprising at first. It would seem that a row with several large entries would be a perfect candidate for composite aggregation. However, consider a demand-matrix row with 5 entries of 100 packets each. Scheduling it using a one-to-many composite path would take 100 slots, where at each slot 5 packets transfer first through the OCS, then through the composite path, and finally through the EPS (as in

**Algorithm 1:** cp-SwitchDemandReduction

```
Data: $D$, $R_t$, $B_t$.
1  $D_I = 0$ ;
2  $D_f = 0$ ;
3  $D_{low} = ZerosAboveB_t(D)$ ;
4  $rows = RowSums(Sign(D_{low})) \geq R_t$ ;
5  $cols = ColSums(Sign(D_{low})) \geq R_t$ ;
6  for each $(i,j) \in rows, (i,j) \notin cols$ do
7      $D_f[i,j] = D[i,j]$ ;
8      $D_I[i,n+1] += D[i,j]$ ;
9  for each $(i,j) \notin rows, (i,j) \in cols$ do
10     $D_f[i,j] = D[i,j]$ ;
11     $D_I[n+1,j] += D[i,j]$ ;
12 for each $(i,j) \in rows, (i,j) \in cols$ do
13     $(x,y) = argmin(D_I[n+1,j], D_I[i,n+1])$ ;
14     $D_f[i,j] = D[i,j]$ ;
15     $D_I[x,y] += D[i,j]$ ;
16 $D_I[1:n, 1:n] = D - D_f$ ;
17 return $(D_I, D_f)$
```

Figure 1(b)). On the other hand, assuming a 1-to-10 link rate ratio, it would take $100/10 = 10$ slots for each of the 5 entries to go through the OCS, together with a reconfiguration time of $5\delta$, *i.e.,* $50 + 5\delta$. As long as $\delta \leq 10$, there is no point using the composite path.

Following (a) and (b), to find these specific patterns, we filter the input demand matrix $D$ and obtain the filtered demand matrix $D_f$.

Specifically, we define two threshold parameters, $B_t$ and $R_t$. First, following intuition (b), we filter the demand $D$ by removing large entries, and assign zeros to each entry that is bigger than threshold $B_t$. Then, following intuition (a), we look for rows and columns that contain at least $R_t$ non-zero entries. All entries that do not belong to such row or column are also set to zero.

We are left with a last challenge. Some entries could be assigned to either their row composite path or their column composite path. Intuitively, we try to load-balance such entries so that the sums of the composite paths will tend to equalize, and one composite path sum will not grow too large compared to the other.

Specifically, we greedily assign such entries to the less loaded composite path, and consider such entries in an arbitrary order.

Finally, Algorithm 1 (cp-SwitchDemandReduction) combines the demand reduction and filtering processes.

**Demand Reduction and Filtering Example:** Figure 2 illustrates a demand reduction and filtering example with $n = 6$, $B_t = 10$ and $R_t = 4$. The input matrix is $D$. First, we set to zero each entry in $D$ that is greater than $B_t$=10. Then, we count the number of non-zero entries in each row and column. Each entry that does not belong to a row or column with a total of at least $R_t$=4 non-zero entries is set to zero as well. Thus, we receive the filtered matrix $D_f$, containing the entries we want to schedule using composite paths. Finally, the reduced demand $D_I$ is equal to $D - D_f$ in the first $n \times n$ entries, representing traffic that is destined to be served by regular EPS-EPS and OCS-OCS paths. The last row and column of $D_I$ represent the composite-path traffic.

Note that the orange entry $D_f[5,2] = 3$ belongs to both a row and a column with at least 4 non-zero entries. Therefore, $D_f[5,2]$ can be assigned to a composite path in either direction (*i.e.,* $D_f[5,2]$ can be added to $D_I[5,n+1]$ and served by a one-to-many composite path given to input port 4. Alternatively, $D_I[5,2]$ can be added to $D_I[n+1,2]$ and served by the many-to-one composite path given to output port 1). After handling all other entries of $D_f$, $D_I[5,n+1] = 15$ and $D_I[n+1,2] = 14$. Thus, we greedily add $D_f[5,2] = 3$ to the smallest sum, *i.e.,* $D_I[n+1,2]$.

## 2.3 cp-Switch Scheduling

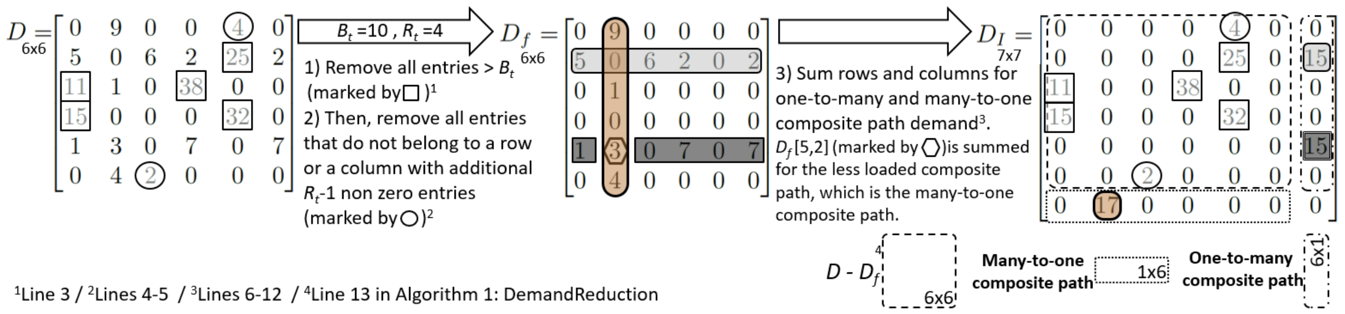
**Scheduling Interpretation:** The scheduler can finally use the h-Switch scheduling algorithm as a subroutine. It feeds it with the new $(n+1) \times (n+1)$ demand matrix $D_I$, and receives a scheduling output, *i.e.,* a set of OCS permutation matrices with their respective duration.

Permutation matrices are treated as usual for entries that are not at the last row or column. When a permutation matrix contains an entry at the last row or column, it represents a port that receives a composite one-to-many or many-to-one path.

**Scheduling Within a Composite Path:** Upon the creation of a composite path with its respective duration, we still need to decide on the scheduling *within* this path. Specifically, we need to choose among the different sources or destinations to serve, and to decide on the duration of this service. In addition, the rate at which data can be transferred along such a path is restricted *asymmetrically.* On one hand, each particular source or destination is connected to the EPS and thus restricted by $C_e$. On the other hand, the path is connected to the OCS, and therefore the *total* rate is restricted by $C_o$. This reflects an inherent tradeoff in the cp-Switch between the benefits of data parallelism and of optical speedup. We choose to send data to all available sources or destinations simultaneously with a rate that respects both constraints.

The scheduler relies on Algorithm 2, denoted *CPSched*, to schedule within a composite path. CPSched receives an input or an output port $p$ with a set of demands $S$ and the path time duration $t$. The output is the remaining set $R$ of demands after the duration of this path. Namely, examining $S - R$ reveals how much traffic is sent to or from each specific port along this path.

**EPS Reservation:** Since composite paths are asymmetric, their scheduling indirectly assumes that the EPS links involved in every particular one-to-many or many-to-one composite-path are reserved for its use when



$B_t$ =10 , $R_t$ =4
1) Remove all entries > $B_t$ (marked by □ )[1]
2) Then, remove all entries that do not belong to a row or a column with additional $R_t$-1 non zero entries (marked by ○ )[2]
3) Sum rows and columns for one-to-many and many-to-one composite path demand[3].
$D_f$[5,2] (marked by ⬡ )is summed for the less loaded composite path, which is the many-to-one composite path.
D - $D_f$[4]
6x6
Many-to-one composite path
1x6
One-to-many composite path
6x1
[1]Line 3 / [2]Lines 4-5 / [3]Lines 6-12 / [4]Line 13 in Algorithm 1: DemandReduction

**Figure 2: Demand reduction and filtering example with $B_t = 10$ and $R_t = 4$. In steps 1) and 2), the scheduler first filters demand matrix $D$ to obtain the filtered demand matrix $D_f$, representing the entries that should be scheduled using composite paths. Then, in step 3), it assigns these entries to specific composite paths (*i.e.*, in the last row or column of $D_I$). The remaining entries of $D_I$ in the other rows and columns will simply be scheduled in the OCS and EPS systems without using composite paths. Next, Figure 3 illustrates the final schedule computation using $D_f$ and $D_I$.**

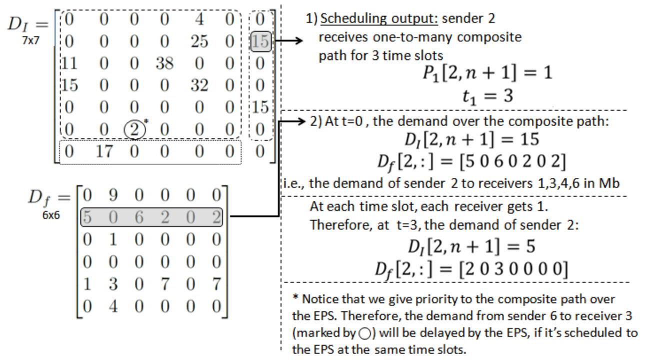
```
Algorithm 2: CPSched
  Data: $S$, $t$, $C_o$, $C_e$
1 $\tau = t$ ;
2 $R = S$ ;
3 while $\tau > 0$ do
4     $R_m = \min\{\text{non-zero}\{R\}\}$ ;
5     $R_c = |\text{non-zero}\{R\}|$ ;
6     $t_{max} = \max\{\frac{R_m}{C_e}, \frac{R_m \cdot R_c}{C_o}\}$ ;
7     $t_{curr} = \min\{t_{max}, \tau\}$ ;
8     for each $d \in R$ do
9         $d = max\{d - t_{curr} \cdot \min\{C_e, \frac{C_o}{R_c}\}, 0\}$ ;
10    $\tau = \tau - t_{curr}$ ;
11 return $R$
```

needed. This assumption may adversely impact short and delay-sensitive flows that want to concurrently use these EPS links.

To resolve this issue, we adopt a simple approach. The scheduler assigns a bandwidth budget $C_e^* \leq C_e$ to the composite paths on each *individual* EPS link when they are used. Then, the scheduler feeds Algorithm 2 with $C_e^*$ instead of $C_e$ to respect this budget. Such bandwidth management on the EPS links can be enforced by well-established traffic-shaping techniques, *e.g.,* [35–38], and bandwidth and resource-reservation techniques, *e.g.,* [39, 40].

*DivideByType* is the last sub-function in our scheduler, as presented in Algorithm 3. It receives an outputted permutation matrix for cp-Switch, and returns its decomposition into regular paths and composite paths with their respective port numbers.

Finally, as illustrated in Figure 4 and formally defined in Algorithm 4, *CPSwitchSched* forms the entire scheduling algorithm for cp-Switch. CPSwitchSched receives as an input the demand and the switch parameters. Its output is a full schedule for cp-Switch.



1) Scheduling output: sender 2 receives one-to-many composite path for 3 time slots
$P_1[2, n+1] = 1$
$t_1 = 3$
2) At t=0 , the demand over the composite path:
$D_I[2, n+1] = 15$
$D_f[2, :] = [5\ 0\ 6\ 0\ 2\ 0\ 2]$
i.e., the demand of sender 2 to receivers 1,3,4,6 in Mb
At each time slot, each receiver gets 1.
Therefore, at t=3, the demand of sender 2:
$D_I[2, n+1] = 5$
$D_f[2, :] = [2\ 0\ 3\ 0\ 0\ 0\ 0]$
* Notice that we give priority to the composite path over the EPS. Therefore, the demand from sender 6 to receiver 3 (marked by ○) will be delayed by the EPS, if it's scheduled to the EPS at the same time slots.

**Figure 3: Example for CPSched. Assume that the scheduler needs to schedule the second one-to-many entry, illustrated in gray, for 3 time-slots. It can service up to 3 packets from each of the non-zero entries in the gray row. Only the first and third entries are left with packets at the end, *i.e.*, $5 - 3 = 2$ and $6 - 3 = 3$ packets respectively.**

```
Algorithm 3: DivideByType
  Data: $P$
1 $P_h = P[1:n, 1:n]$ ;
2 $row = argmax(P[:, n+1] > 0)$ ;
3 if $row \neq NULL$ then
4     $S_{row} = P[row, :]$ ;
5 $col = argmax(P[n+1, :] > 0)$ ;
6 if $col \neq NULL$ then
7     $S_{col} = P[col, :]$ ;
8 return $P_h, S_{row}, row, S_{col}, col$
```

**Complexity:** We want to evaluate the complexity of the entire scheduling procedure. We begin by analyzing Algorithm 1 (cp-SwitchDemandReduction). Since this algorithm requires to go over the demand matrix

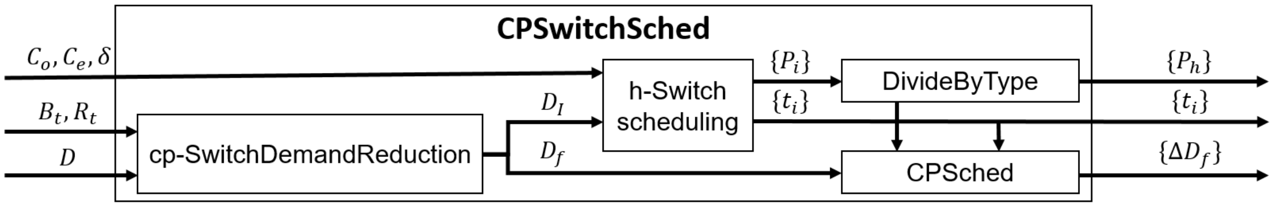


**Figure 4: Diagram block for the cp-Switch scheduler CPSwitchSched. The initial demand matrix $D$ is first reduced and filtered to obtain demand matrix $D_I$ of size $(n+1)\times(n+1)$. $D_I$ is then sent to an h-Switch scheduler, which outputs permutation matrices and their respective duration. But these permutations fit an OCS in the h-Switch, not in the cp-Switch. They need to be sent to DivideByType in order to obtain a decomposition into regular paths and composite paths with their respective port numbers. Finally, CPSched provides the specific scheduling time of each sender or receiver using a composite path.**

**Algorithm 4:** CPSwitchSched

```
Data: $D$, $C_o$, $C_e$, $C_e^*$, $\delta$, $R_t$, $B_t$.
1  $(D_I, D_f)$=cp-SwitchDemandReduction($D$,$R_t$,$B_t$) ;
2  $sched$=h-Switch-Scheduling($D_I$, $C_o$, $C_e$, $\delta$) ;
3  $cpssched = \{\}$ ;
4  for $each\ (P_i, t_i) \in sched$ do
5      $D_{f,prev} = D_f$ ;
6      $(P_h, S_r, r, S_c, c)$ = DivideByType($P_i$) ;
7      if $r \neq NULL$ then
8          $D_f[r,:]$ = CPSched($S_r, t_i, C_o, C_e^*$) ;
9      if $c \neq NULL$ then
10         $D_f[:,c]$ = CPSched($S_c, t_i, C_o, C_e^*$) ;
11     $cpssched = cpssched \bigcup \{P_h, t_i, D_{f,prev} - D_f\}$ ;
12 return $cpssched$
```

a constant number of times (*i.e.,* 3 at most), we obtain a complexity of $O(n^2)$. Algorithm 2 (CPSched) can do no more than $n-1$ iterations (*i.e.,* the *while* loop) where in each such iteration it needs to update the remaining demand. Each such update is done efficiently in $O(\log n)$ by using a priority queue and two counters. Thus, the complexity of Algorithm 2 is $O(n \log n)$. Clearly, since Algorithm 3 requires examining the permutation matrix only once, its complexity is $O(n^2)$. Finally, putting it all together, the complexity of the *interpretation process* part of Algorithm 4 (CPSwitchSched) is $O(n^2)$ for each permutation matrix in the schedule. Thus the complexity of the entire algorithm is $O(\max\{m \cdot n^2, \text{h-Switch-Scheduling}\})$ where $m$ is the number of OCS reconfigurations in the schedule (*i.e.,* the number of permutation matrices) and $O$(h-Switch-Scheduling) is the complexity of the h-Switch scheduler.

## 3. EVALUATION

In this section, we evaluate and compare h-Switch and cp-Switch with $C_e = 10$ Gbps and $C_o = 100$ Gbps (*i.e.,* 1/10 ratio), for radix (*i.e.,* number of ports) 32, 64 and 128. Two OCS types are considered throughout our evaluation: *Fast OCS* with $\delta = 20\mu s$ (2D MEMS wavelength selective switches, such as [4, 6, 41]), and *Slow OCS* with $\delta = 20ms$ (3D MEMS optical circuit switches, such as [3, 5, 42, 43]).

We evaluate different demands, which include a DCN traffic workload [44], one-to-many, and many-to-one traffic demands. For each evaluation, we define the demand model and produce 100 random demand matrices accordingly. Then, each such demand is evaluated for both h-Switch and cp-Switch, by the same order of the permutation matrices produced by the scheduling algorithms (termed *online execution*). Finally, the results of all the 100 random demands are averaged and compared between h-Switch and cp-Switch.

The demand reduction process for cp-Switch is configured with $R_t = 0.7 \cdot n$, and $B_t = 2Mb$ ($B_t = 200Mb$) for the Fast OCS (Slow OCS). Further discussion about $R_t$ and $B_t$ settings can be found in Section 4.

In the following we present the evaluation criteria in Section 3.1. Then, in Section 3.2 we evaluate a given one-to-many/many-to-one DCN demand example. We extend the demand model by adding typical background DCN demand in Section 3.3. In Section 3.4 we stress the schedulers with intensive background demand along with one-to-many and many-to-one demand. Finally, in Section 3.5 we stress the composite paths by increasing the one-to-many/many-to-one demands along with typical DCN traffic.

### 3.1 Evaluation Criteria

For our evaluation, we adopt two recent state-of-the-art scheduling algorithms optimized for h-Switch, Eclipse [1] and Solstice [2], which attempt to optimize two different metrics: fraction of demand over the OCS, and demand completion time, respectively.

**Eclipse (Fraction of Demand over the OCS[1]):** Eclipse identifies a submodularity structure [45] and leverages it to *guarantee* an efficiency of at least half of the optimal solution according to the associated op-

[1] Notice that for a similar demand and switch parameters (*i.e.,* $n, C_e, C_o$), increase of this metric corresponds to increase of the OCS utilization.

timization criterion, which is to maximize the fraction of demand over the OCS in a given time window. In our evaluation, Eclipse is used with a window period input of $1ms$ for the Fast OCS, and $100ms$ for the Slow OCS.

**Solstice (Completion Time):** Solstice takes advantage of the sparsity and skewness observed in real DCN traffic. The basic idea of Solstice is to *stuff* the demand into a bistochastic matrix and then efficiently decompose the demand by *slicing* it into permutation matrices with long duration. The guiding optimization criterion of Solstice is the completion run-time.

Each given demand matrix $D$ is scheduled by Eclipse and Solstice, for both h-Switch and cp-Switch[2]. Then, we measure the corresponding metrics for the total demand. Notice that for cp-Switch the metrics include the demand that is assigned to the composite paths, which we also measure separately. Furthermore, for comparison purpose, we measure the metrics of the same demand (*i.e.,* assigned to the composite paths in the cp-Switch) for the h-Switch.

**OCS Reconfigurations:** We count the number of OCS configurations required by Solstice and Eclipse. This measurement has a strong correlation with the results. Namely, a large number of reconfigurations leads to both longer completion times and diminished OCS utilization, because of the idle time periods during the OCS reconfigurations.

## 3.2 One-to-Many/Many-to-One Demand Example

Our one-to-many and many-to-one demand model is based on previous work that presents DCN measurements, *e.g.,* DCTCP [23] and TCP Outcast [24].

**One-to-Many/Many-to-One Demand Model:** In this section, we start with a simple example in order to provide insight about the benefit of cp-Switch composite paths. We randomly choose a single sender for which we create one-to-many traffic and a single receiver for which we create many-to-one traffic. All other demands are zero. The number of destinations for the sender and the number of sources for the receiver are chosen randomly and uniformly in the range of $[0.7{\cdot}n, n]$. The demand towards each destination of the sender and each source of the receiver is chosen randomly and uniformly in the range of $[1, 1.3]$ Mb for Fast OCS and $[100, 130]$ Mb for Slow OCS.

**Evaluation Results:** Figure 5(a) and Figure 5(b) present the completion time of the total demand, and the demand that is assigned to the one-to-many and many-to-one composite paths (noted by *o2m* and *m2o*, respectively). Compared to h-Switch, cp-Switch results in a better completion time for the total, one-to-many, and many-to-one demands, for both the Fast and Slow OCSs. Furthermore, the relative completion time improvement of cp-Switch increases as the switch radix increases.

There are several reasons for the h-Switch completion time degradation: First, the total demand of the sender and the receiver exceeds the EPS port bandwidth due to the high number of destinations and sources, respectively. Second, Figure 5(c) presents the required number of OCS configurations. h-Switch serves the demand by many costly OCS reconfigurations, which increase with the switch radix. On the other hand, cp-Switch employs its one-to-many and many-to-one composite paths to serve the demand, without any OCS reconfigurations. Moreover, the completion time improvement of cp-Switch as compared to h-Switch is more significant for the Slow OCS, due to its higher reconfiguration penalty as compared to the time it takes to serve the demand.

Figure 6(a) presents the fraction of the demand that is served by the OCS, which corresponds to the OCS utilization. The objective of Eclipse is to maximize the OCS utilization, and therefore it prefers to schedule large entries over the OCS. Therefore, one-to-many and many-to-one demands are scheduled over the OCS with lower priority.

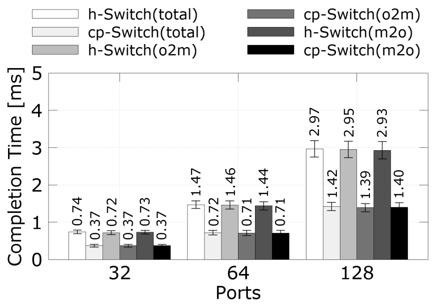
For switch radix 32, the one-to-many and many-to-one demands do not incur a severe bottleneck over the sender and the receiver, respectively; hence, Eclipse achieves the demand fraction over the OCS for both h-Switch and cp-Switch. However, as the switch radix increases, the sender and the receiver are overloaded by the one-to-many and many-to-one demand as it increases as well. Therefore, the cp-Switch results improve with the switch radix as compared to h-Switch.

Eclipse achieves a lower OCS utilization as the OCS reconfiguration penalty increases. Therefore, for the Slow OCS case, cp-Switch achieves much higher results as compared to h-Switch (Figure 6(b)). Figure 6(c) presents the required OCS configurations for h-Switch and cp-Switch with both Fast and Slow OCSs. Eclipse produces approximately the same number of OCS configurations independently of the switch radix. h-Switch with Fast OCS requires approximately 31-35 OCS configurations by Eclipse, where each re-configuration takes $20\mu s$. Therefore, the total reconfiguration penalty of the fast OCS equals $620\mu s$-$700\mu s$, out of its $1ms$ period window.

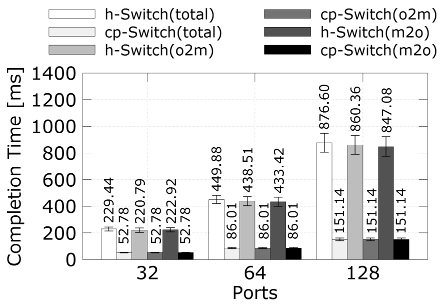
## 3.3 Typical DCN with One-to-Many/Many-to-One Demand Example

Our demand model consists of both typical data center demand as background traffic, and one-to-many and many-to-one traffic patterns. The typical DCN demand is based on the aforementioned works on data center traffic; and the one-to-many and many-to-one demand is based on our demand model described in Section 3.2.

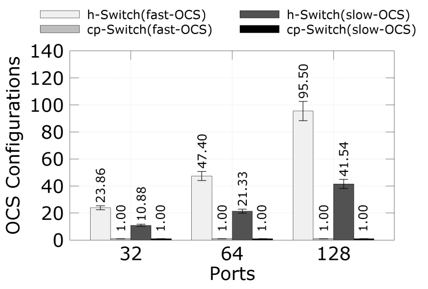
[2]For cp-Switch, we employ the demand reduction and filtering, then we employ Eclipse or Solstice, respectively, for the new demand $D_I$, and finally produce cp-Switch scheduling (*i.e.,* we employ Algorithm 4).



(a) Completion time with Fast OCS.

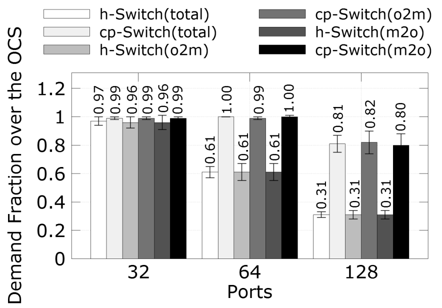


(b) Completion time with Slow OCS.

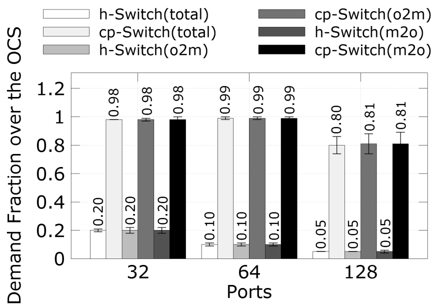


(c) OCS configurations.

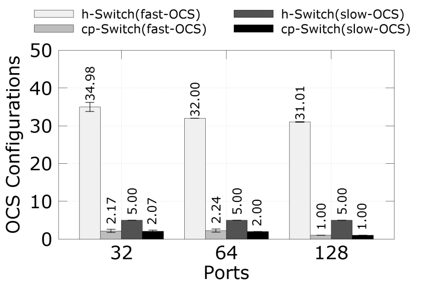
**Figure 5: One-to-Many/Many-to-One Demand Example - Completion Time (Based on Solstice). The cp-Switch advantage over h-Switch increases with the switch radix. This is due to the increased number of reconfigurations required by h-Switch as the switch radix increases; while cp-Switch does not require any reconfiguration. The number of reconfigurations for h-Switch is proportional to the increased number of sources and destinations in the many-to-one and one-to-many traffic, respectively.**



(a) Fast OCS. Window of 1 *ms*.



(b) Slow OCS. Window of 100 *ms*.



(c) OCS configurations.

**Figure 6: One-to-Many/Many-to-One Demand Example - Fraction of Demand Served by the OCS (Based on Eclipse). There is a significant degradation of the OCS utilization for h-Switch that is proportional to the switch radix. On the contrary, cp-Switch demonstrates high utilization for 32 and 64 ports, and only a slight degradation for 128 ports. This is because h-Switch with fast OCS spends more than half of the time window on reconfigurations, and with slow OCS more than 80%. On the other hand, cp-Switch requires at most 1-2 reconfigurations on average.**

**Our Typical Background Demand Modeling:** Our typical background demand modeling is based on the DCN measurements presented in [44], and is constructed similarly to the demand used in Eclipse [1] and Solstice [2]. Some of the input ports have four big flows (a.k.a. elephant flows, 30Mb and 3Gb for Fast OCS and Slow OCS, respectively) and 12 small flows (a.k.a. mice flows, 3Mb and 300Mb for Fast OCS and Slow OCS, respectively), where the big flows carry 70% of the demand. The destination of the flows is chosen randomly and uniformly. We refer to this demand as *typical background traffic*.

**Evaluation Results:** Figures 7(a) and 7(b) present the completion time of the total demand, and the demand that is assigned to the one-to-many and many-to-one composite paths (noted by *o2m* and *m2o*, respectively). The cp-Switch with Fast OCS results in completion time reduction of 15%-70% for the one-to-many and many-to-one demands, and of 9%-37% for the total demand; and the cp-Switch with Slow OCS results in completion time reduction of 11%-75%, and 4%-49%, respectively. The results fit with the number of OCS configurations presented in Figure 7(c); *i.e.*, more OCS configurations result in higher completion time.

Figure 8(a) indicates that the fraction of the demand that is served by the Fast OCS within the 1*ms* period window is 2-3 times higher for cp-Switch, as compared to h-Switch. Figure 8(b) shows that cp-Switch with Slow OCS results in improvements that are 5.4-10 times higher, within the 100*ms* period window. Similarly to the results in Section 3.2, Figure 8(c) shows that Ecplise results in approximately the same number of OCS configurations, independently of the switch radix. Again, the h-Switch with the Fast OCS incurs significant penalty by the OCS reconfigurations, which consumes 62%-72% of its 1*ms* period.

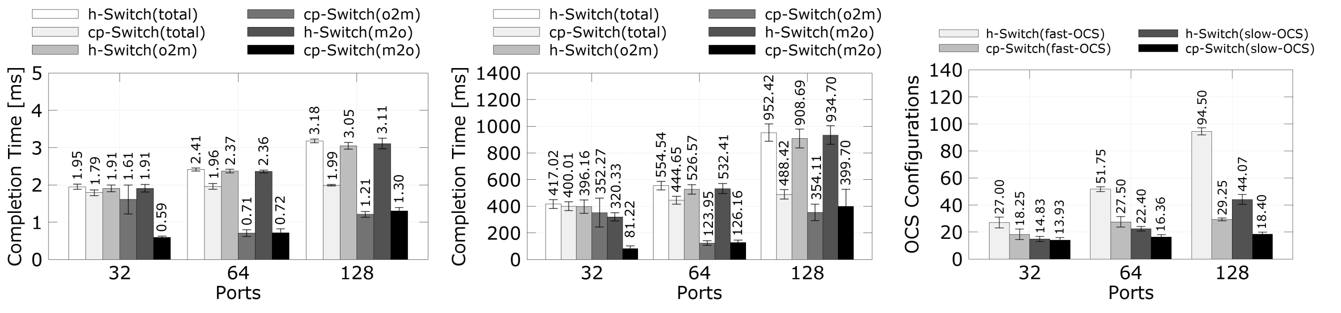

(a) Completion time with Fast OCS. (b) Completion time with Slow OCS. (c) OCS configurations.

**Figure 7: Typical DCN with One-to-Many/Many-to-One Demand Example - Completion Time (Solstice Based). In addition to the significantly faster completion time for one-to-many and many-to-one traffic, the completion time of the entire demand decreases by up to 37% and 49% for fast and slow OCS, respectively.**

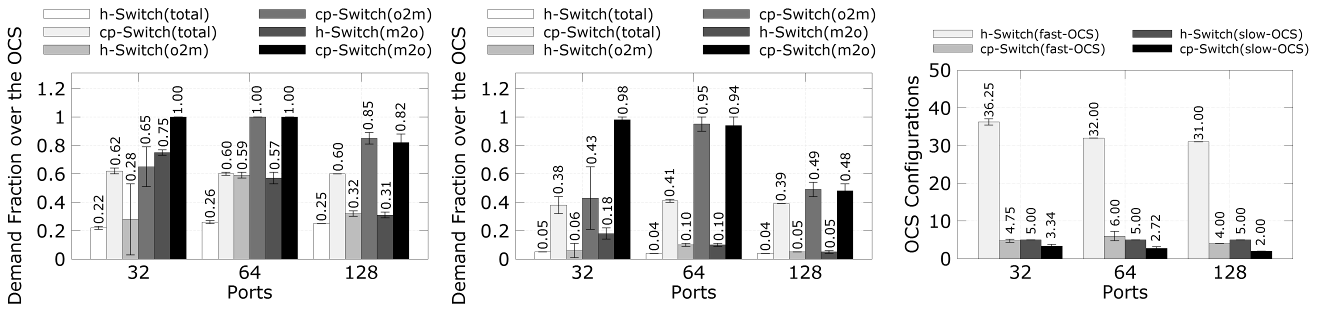

(a) Fast OCS. Window of 1 *ms*. (b) Slow OCS. Window of 100 *ms*. (c) OCS configurations.

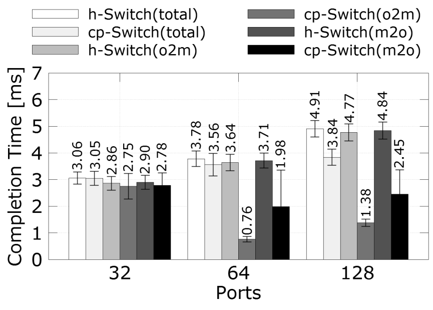
**Figure 8: Typical DCN with One-to-Many/Many-to-One Demand Example - OCS Utilization (Eclipse Based). The OCS utilization of both h-Switch and cp-Switch for the total demand is similar for each switch radix. The cp-Switch improves the OCS utilization by up to 800%, as compared to h-Switch.**

The completion time (by Solstice) and fraction of demand over the OCS (by Eclipse) are improved by cp-Switch over h-Switch for both the total and one-to-many and many-to-one demands. The improvements of the one-to-many and many-to-one demands by cp-Switch clearly result from the use of the composite paths. On the other hand, the improvements of the total demand stem from the following reasons: First, the lower number of OCS reconfigurations as indicated by Figures 7(c) and 8(c). Second, both Solstice and Eclipse perform better when the demand matrix is more sparse and contains similar traffic patterns (*i.e.,* our empirical results that correspond to [1] and [2] show that such matrices are more efficiently decomposed into a smaller set of permutation matrices with a longer duration). It allows both algorithms to use less configurations with a larger duration; hence providing better utilization of the OCS and reducing the completion time of the total demand. Specifically, in these experiments the mean number of non-zero entries in the reduced demand matrix for cp-Switch is lower by $1.63 \cdot n$. Hence, it results in a better scheduling.

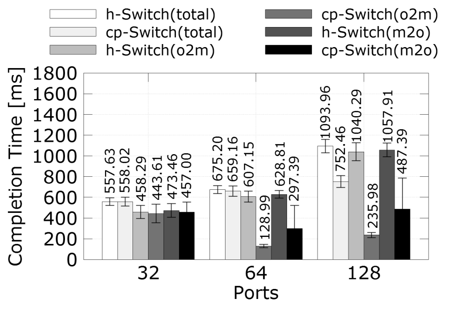
## 3.4 Intensive Typical DCN and One-to-Many/Many-to-One Demand

In this section, in order to stress the cp-Switch scheduler, we create a one-to-many/many-to-one demand (as defined in Section 3.2) together with an intensive typical background demand. To that end, similarly to Solstice scheduler stressing tests, we increase the density of the demand matrix (*i.e.,* non-zero entries) by a factor of four.

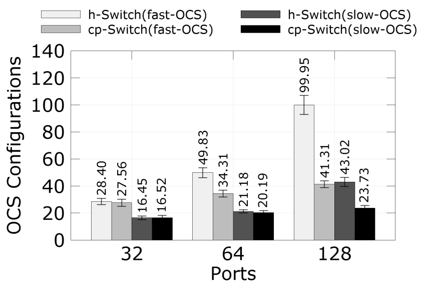
Figures 9(a) and 9(b) present the completion time of the total demand, and the demand that is assigned to the one-to-many and many-to-one composite paths (noted by *o2m* and *m2o*, respectively). It can be seen that for switch radix 32, the differences of the completion times between h-Switch and cp-Switch are within 5%. This result stems from the heavy background demand, which dictates the completion time for both switches. In addition, the number of OCS reconfigurations is almost identical, as indicated by Figure 9(c)—causing the completion time of the one-to-many and many-to-one demand to increase as well due to the expensive OCS idle periods. As the switch radix increases



(a) Completion time with Fast OCS.

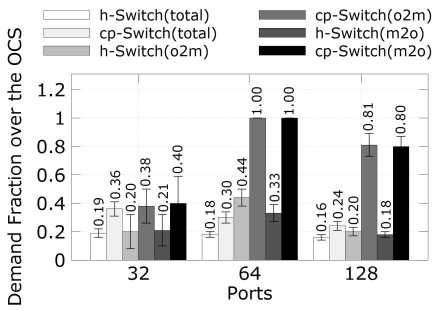


(b) Completion time with Slow OCS.

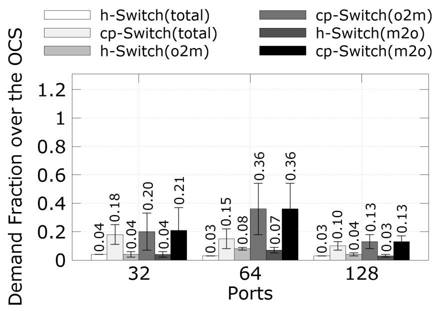


(c) OCS configurations.

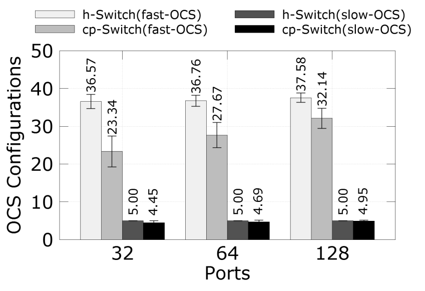
**Figure 9: Intensive Typical DCN and One-to-Many/Many-to-One Demand - Completion Time (Solstice Based). For 32 and 64 ports, the completion time is similar since it is dominated by the intensive background traffic. However, for 128 ports the completion time of the total demand for cp-Switch decreases by up to 7% and 27% for the fast and slow OCS, respectively.**



(a) Fast OCS. Window of 1 $ms$.



(b) Slow OCS. Window of 100 $ms$.



(c) OCS configurations.

**Figure 10: Intensive Typical DCN and One-to-Many/Many-to-One Demand - OCS Utilization (Eclipse Based). cp-Switch improves the OCS utilization by up to 400%, as compared to h-Switch.**

to 64 and 128, cp-Switch improves the completion time of the total demand, as compared to h-Switch, by 6% and 27%, respectively. The improvement stems from the increased number of OCS reconfigurations of h-Switch, as shown in Figure 9(c). On the other hand, cp-Switch significantly improves the completion time of the one-to-many and many-to-one demand by 46%-80%.

Figure 10 demonstrates the same OCS utilization improvement trend as shown in Figure 8. Therefore, we demonstrate the stability of our scheduler, which results in the same improvements also when it is stressed by a higher density demand matrix.

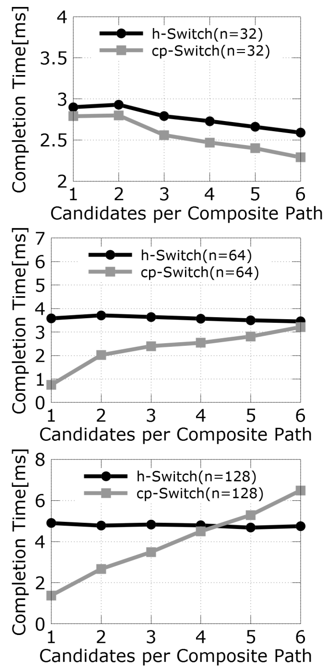
## 3.5 Typical DCN Traffic and Intensive One-to-Many/Many-to-One Demand

In this section, we create a typical background demand (as defined in Section 3.2) together with a varying one-to-many/many-to-one demand. Specifically, we increase the number of senders and receivers with one-to-many and many-to-one demand from one to six, respectively. These demands are generated such that they are chosen to be served by the composite paths, according to the filtering parameters employed by algorithm 1 for the demand reduction process. We seek the point at which the composite paths are overloaded; *i.e.,* how many one-to-many/many-to-one demand ports can cp-Switch bear?

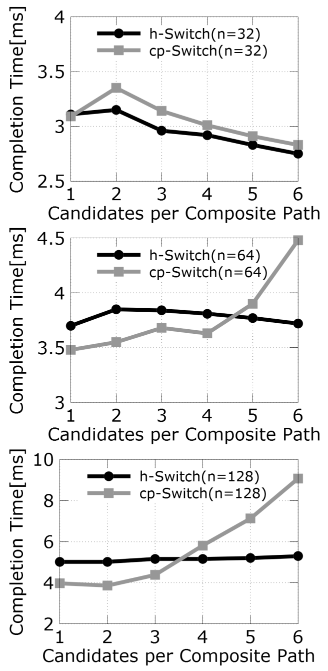
Figure 11 presents the completion time of the total demand, and the demands that are assigned to the one-to-many and many-to-one composite paths, for switch radix 32, 64, and 128.

For switch radix 128 and more than four ports with one-to-many/many-to-one demand, cp-Switch results in a higher completion time as compared to h-Switch. In such scenarios, enforcing too many one-to-many/many-to-one demand ports over the composite paths results in performance degradation for cp-Switch.

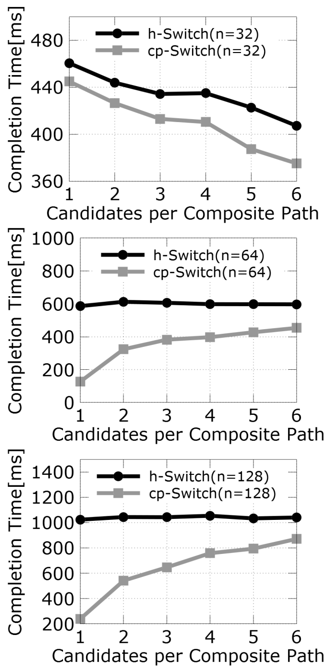
There are several solutions to mitigate the following performance degradation of cp-Switch: First, tuning the filter parameters to select a lower number of ports with one-to-many/many-to-one demand (*i.e.,* increasing $R_t$ and/or decreasing $B_t$). Second, we can revise the cp-Switch scheduler to limit the number of served one-to-many/many-to-one demand ports over the composite paths. Third, we can increase the number of composite paths in cp-Switch (Further discussed in Section 4).



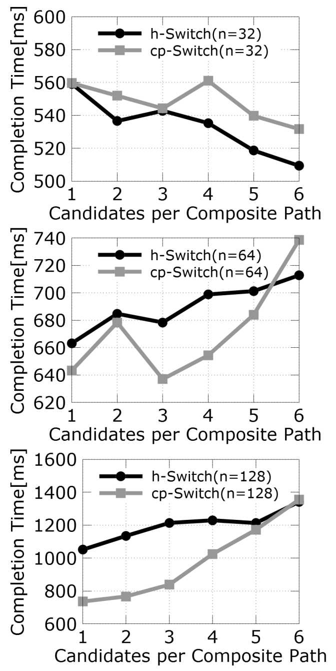
(a) Fast OCS. One-to-many and many-to-one ports.



(b) Fast OCS. Total demand completion time.



(c) Slow OCS. One-to-many and many-to-one ports.



(d) Slow OCS. Total demand completion time.

**Figure 11: Typical DCN Traffic and Increasing One-to-Many/Many-to-One Demand. The advantage in favor of cp-Switch decreases as the composite one-to-many and many-to-one paths are overloaded.**

## 4. DISCUSSION AND FUTURE WORK

In this section we discuss possible extensions and future directions for composite-path switching.

**Binding the cp-Switch and h-Switch Scheduling Run Times:** We want to provide an efficient way to extend a given h-Switch algorithm to cp-Switch scheduling, and thus to *bind* the scheduling feasibility of both systems. Accordingly, in our evaluation, we found that the complexity of CPSwitchSched is dominated by the used h-Switch scheduling algorithm, i.e., Eclipse or Solstice[3]. Specifically, for all switch sizes we test (*i.e.,* 32, 64, 128), the h-Switch scheduling sub-routine takes at least 88% of the cp-Switch algorithm run time. Moreover, as the switch radix grows, this percentage increases and reaches more than 99% for 128 ports under moderate load.

We summarize the comparison between h-Switch and cp-Switch scheduling algorithms run-time in Tables 1 and 2. Since our absolute running times are not optimized (we use a high-level Python implementation for our controller), we emphasize the ratio between h-Switch and cp-Switch algorithms run-time which also corresponds to the theoretical analysis.

[3] In our implementation, the complexity of Eclipse is $(O(n^3 log(n) \cdot \frac{W}{\delta}))$ where $W$ is the time window for the schedule and there are at least $n$ non-zero entries in $D$. The complexity of Solstice is $O(n^3 log^2(n))$.

| n/Ev. | Typical (Figure 7) | | | Intensive (Figure 9) | | |
|---|---|---|---|---|---|---|
| | h-Switch | cp-Switch | **Ratio** | h-Switch | cp-Switch | **Ratio** |
| 32 | 7.1, 16.5 | 8.2, 11.1 | **0.86, 1.49** | 8.3, 14.6 | 9.2, 14.2 | **0.9, 1.03** |
| 64 | 35.7, 97 | 31.1, 41.3 | **1.48, 2.35** | 35.4, 75.5 | 32.2, 56 | **1.1, 1.35** |
| 128 | 222, 453 | 111, 154 | **2, 2.94** | 244, 456 | 132, 243 | **1.85, 1.87** |

**Table 1: Comparing h-Switch and cp-Switch scheduling run-times using Solstice. Each entry presents the run-time in milliseconds for slow and fast OCS pair, *i.e.,* (slow, fast).**

| n/Ev. | Typical (Figure 8) | | | Intensive (Figure 10) | | |
|---|---|---|---|---|---|---|
| | h-Switch | cp-Switch | **Ratio** | h-Switch | cp-Switch | **Ratio** |
| 32 | 15.1, 97.2 | 9.4, 102 | **1.6, 0.95** | 146, 1070 | 86, 480 | **1.7, 2.38** |
| 64 | 55.3, 330 | 25, 40.2 | **2.22, 8.2** | 310, 3310 | 341, 2643 | **0.91, 1.25** |
| 128 | 210, 1350 | 70, 133 | **3, 10.15** | 1370, 9340 | 1020, 6782 | **1.35, 1.38** |

**Table 2: Comparing h-Switch and cp-Switch scheduling run-times using Eclipse. Each entry presents the run-time in milliseconds for slow and fast OCS pair, *i.e.,* (slow, fast).**

As can be seen in Tables 1 and 2, in all our experiments, the run time of the scheduling algorithm for cp-Switch was at most 14% slower than for h-Switch when both scheduling algorithms produce similar number of permutation matrices, and mostly much faster, up to an order of magnitude, when the produced number of permutation matrices for cp-Switch is significantly lower. The reason for this is the strong dependence of the scheduling run time in the number of produced permutation matrices, which is consistently lower for cp-Switch, especially as the switch radix grows.

**Tuning Heuristic:** In our evaluation we set the filtering parameters $B_t$ and $R_t$ according to the intuition provided in Section 2.2. Specifically, $B_t$ and $R_t$ are set to: (1.) capture traffic that utilizes the OCS links of composite paths without overwhelming the corresponding EPS links of each particular source or destination port within these paths; and (2.) make sure that the time to serve each individual entry does not exceed to much the reconfiguration penalty of the OCS—namely, by aggregating such entries to the composite entries, we want to avoid multiple OCS reconfigurations with potentially low utilization.

**(1.)** $R_t$ is set to filter one-to-many and many-to-one demands with large fan-out relative to the switch radix (*i.e.,* large number of destination/sources, respectively). Hence, we set $R_t$ to be $\beta \cdot n$, where $n$ is the switch radix and $\beta$ is a factor variable ($0 < \beta \leq 1$). Specifically, in our evaluations we use $\beta = 0.7$, such that the fan-out is high enough to guarantee that the OCS is fully utilized when serving the demand over the composite paths.

**(2.)** $B_t$ is set to filter demand entries which are suitable for the composite paths, which aggregation avoids the OCS configurations penalty by utilizing the EPS through the composite paths. To that end, such demand entries should be below certain value, in order to sustain the EPS ports bandwidth, and to avoid frequent OCS reconfigurations. Since $R_t$ is already set to utilize the OCS, $B_t$ is set to be proportional to the OCS configuration penalty. Specifically, any demand below a certain value over the OCS results in inefficient utilization and expensive OCS reconfiguration penalty. Therefore, we set $B_t$ to $\alpha \cdot (\delta \cdot C_o)$, where $\alpha$ is a proportion factor. In our evaluations we use $\alpha = 1$ ($B_t = 2Mb$) and $\alpha = 0.1$ ($B_t = 200Mb$) for the Fast OCS and Slow OCS, respectively.

**Optimal Tuning:** In this work, we did not exhaustively search for the best $B_t$ and $R_t$. We chose the specific $(\alpha, \beta)$ pair heuristically according to the provided intuition and by examining the workload. Generally, we have found that providing the best $(\alpha, \beta)$ pairs to an arbitrary demand is challenging since there is a strong coupling between the algebraic structure of the demand matrix, the switch parameters (*i.e.,* $n, C_e, C_o$) and the performance of the scheduling algorithms (*i.e.,* some matrices are more easily decomposed to a smaller set of permutation matrices as previously indicated by [1, 2]). Thus, we leave a further study of $(\alpha, \beta)$ tuning to future work.

**Offline Execution:** During our work, we have also examined possible reorderings of the permutation matrices execution. Clearly, nor the total completion time neither the average OCS utilization in a given time window are affected by such reordering. In addition, we have found that such a reordering also has a negligible impact on the average completion time of one-to-many and many-to-one traffic in h-Switch (such traffic patterns require multiple OCS reconfigurations to be served via the OCS. Thus, even such a change in the order of execution is limited by the total high penalty of constant reconfigurations and lack of bandwidth of the EPS links to serve such demand). On the contrary, such reordering can reduce the completion time of one-to-many and many-to-one demand in cp-Switch. Specifically, consider a single input port with a demand of a single big flow and many small flows, each sent to a different destination. Then, the scheduler can schedule the big flow over the OCS repeatedly, before the one-to-many traffic has been satisfied. Therefore, in such a case of mixed *big* single flow demand and one-to-many/many-to-one demand, reordering of the permutation matrices can reduce the completion time (termed *offline execution*).

**Additional Composite Paths:** The tradeoff of having more or less composite paths in cp-Switch is between having the ability to handle more one-to-many and many-to-one traffic on one hand, and under-utilizing high bandwidth EPS ports in the absence of such traffic on the other. Thus the choice of how many composite paths lead to the best performance/price tradeoff is dependent both on the switch radix and on the workload. We examined the benefit of having a single composite path in each direction. However, for high-radix switches it may be beneficial to have more than a single composite path in each direction. In Section 3.5 we demonstrate that for intensive one-to-many/many-to-one demand, cp-Switch might underperform, due to lack of composite paths. To that end, the demand reduction (Algorithm 1) and cp-Switch scheduler (Algorithm 4) should be extended.

Specifically, the desired extension to handle $k$ composite paths in each direction should preserve the ability of using h-Switch scheduling technique as a subroutine in order to make cp-Switch scheduling more tractable and leverage state-of-the-art evolving h-Switch scheduling techniques to cp-Switch scheduling.

To that end, the reduced demand matrix $D_I$ has additional $k$ rows and $k$ columns to represent the composite paths (*i.e.,* a row for each many-to-one composite-path and a column for each many-to-one composite-path). Then, we need to decide during the demand filtering process (Algorithm 1) how to balance the traffic among the $k$ composite paths. This is achieved by extending lines 7, 10 and 13 in Algorithm 1 to consider the minimal composite entry so far. This can be done efficiently by using priority queues to hold the minimal composite entries among the different paths. Thus, the resulting complexity of Algorithm 1 is $O(log(k) \cdot (n+k)^2)$. The extensions to Algorithms 2, 3 and 4 are straightforward resulting in a total complexity of $O(\max\{m \cdot (n+k)^2, \text{h-Switch-Scheduling}\})$ for Algorithm 4.

**Scaling:** As depicted in Figure 1, cp-Switch is composed of both EPS and OCS planes; therefore, one should scale both of the planes in order to scale the cp-Switch. First, scaling the EPS plane can be done using different multi-layer topologies (*e.g.,* , Folded-Clos or Fat-Tree). Furthermore, current electronic packet switches already offer for example 128 ports of 25 Gbps [46], and their radix is regularly increased. Second, OCS systems also offer a higher radix either using switches with a high radix (*e.g.,* of about 1,000 ports [47]), or using Folded-Clos optical cross-connect fabrics that can scale incrementally to tens of thousands of ports [48].

**Augmenting Hybrid Architectures:** Current hybrid architectures present topologies with a hard separation between the OCS and the EPS switching planes. As we showed in this work, integrating between the two fabrics using composite paths has the potential to better support modern data center traffic and provide better flow completion times and better link utilization. For example: (a) a leaf-spine hybrid solution such as [3] can be extended by connecting among the OCS and the EPS spines. Clearly, the exact number of the suggested connections is highly dependent on the number of OCS and EPS spines as well as on the typical workload. (b) Three tier hybrid solutions such as [5] can be extended by connecting the OCS to the aggregation layer switches on top of the ToR switches, thus allowing composite-paths among them.

**Additional Use Cases:** As mentioned, cp-Switches can be used to connect among servers and/or among ToR switches. As such, they can handle more traffic patterns. In addition, cp-switches can be used for special uses such as ToRs in specific racks that require one-to-many and many-to-one traffic support that nowadays cannot be served efficiently by a hybrid switch, *e.g.,* storage racks and especially object storage [49].

## 5. RELATED WORK

**Hybrid Switching:** One of the very first comprehensive works that advocated OCS for the DCN [50] has considered HPC workloads with static or semi-static traffic patterns. Such traffic patterns naturally benefit from offloading the long lived one-to-one data transfers to high capacity optical circuits, while sending the rest of the traffic over traditional EPS. Follow up works, *e.g.,* Helios [3] and c-Through [5], have refined the approach and presented different methods for identifying big one-to-one flows, heuristics for circuits scheduling, and control planes for sending the traffic over EPS and OCS paths. REACToR [6] leveraged a microsecond scale Mordia [51] optical circuit switch and proposed a control plane which synchronizes end host transmissions with end-to-end circuit assignments over a hybrid network. Their control plane is designed to react to rapid, bursty changes in the traffic from end hosts on a time scale of 100s of microseconds. XFabric [41] suggested a *rack-scale* network that reconfigures the topology and uplink placement using a circuit-switched physical layer over which System-on-Chips perform packet switching.

**Free Space Optics:** FireFly [52] suggested free-space optics (FSO) as a potential technology to provide flexible wireless interconnect throughout the data center network. An additional advancement was made by ProjecTor [53] that suggested to use digital micromirror devices (DMDs), instead of the Galvo or switchable mirrors used by Firefly, which enable a single transmitter to reach high fan-out and a microseconds scale reconfiguration time periods.

In a sense, an FSO system can be logically viewed as a high radix distributed OCS, thus shares its scheduling disadvantages. Accordingly, we believe that FSO can also benefit from composite-paths. Consider a rack producing one-to-many coflows. Then in a pure FSO system, such traffic, in order to be served quickly can repeatedly occupy multiple DMDs for short time periods causing poor circuit utilization due to the reconfiguration penalties, or alternatively suffer from large completion times waiting for multiple reconfigurations of fewer DMDs. Such an FSO system can reach higher utilization when augmented by low-bandwidth EPS system with fewer high-bandwidth EPS ports reachable by DMDs.

## 6. CONCLUSIONS

In this work, we introduced the *composite path switch* to deal with skewed one-to-many and many-to-one coflows. We explained how it efficiently extends hybrid circuit/packet switches and allows them to efficiently support modern datacenter workloads.

We also tackled the challenge of simplifying cp-Switch scheduling so as to leverage state-of-the-art hybrid-switch scheduling techniques, by building a reduced demand matrix, filtering irrelevant traffic, scheduling flows within a composite path and reserving specific packet-switching slots. Finally, using extensive evaluation, we demonstrated that the composite-path switch is capable of accommodating more traffic patterns while improving significantly both the flow completion time and the circuit utilization metrics.

Looking forward, our results open new opportunities for leveraging the reconfiguration capabilities of contemporary optical circuit switches for scaling modern data center networks and modern application workloads.

## 7. ACKNOWLEDGMENTS

The authors would like to thank Eitan Zahavi, Tal Mizrahi, our shepherd Paolo Costa and our anonymous reviewers for their helpful comments. The research leading to results published in this paper was partially supported by the European Community's Seventh Framework Programme (FP7/2001-2013) under grant agreement number 619572, in the context of the COSIGN Project, as well as by the Hasso Plattner Institute Research School, the Technion Fund for Security Research, the Gordon Fund for Systems Engineering, and the Israel Ministry of Science and Technology.

## 8. REFERENCES

[1] Shaileshh Bojja, Mohammad Alizadeh, and Pramod Viswanath. Costly circuits, submodular schedules and approximate Carathéodory theorems. *ACM Sigmetrics*, 2015.

[2] He Liu, Matthew K Mukerjee, Conglong Li, et al. Scheduling techniques for hybrid circuit/packet networks. *ACM CoNEXT*, 2015.

[3] Nathan Farrington et al. Helios: a hybrid electrical/optical switch architecture for modular data centers. *ACM SIGCOMM Computer Communication Review*, 2011.

[4] George Porter, Richard Strong, Nathan Farrington, et al. *Integrating microsecond circuit switching into the data center.* ACM SIGCOMM, 2013.

[5] Guohui Wang, David G Andersen, Michael Kaminsky, et al. c-Through: Part-time optics in data centers. *ACM SIGCOMM Computer Communication Review*, 2011.

[6] He Liu, Feng Lu, Alex Forencich, Rishi Kapoor, et al. Circuit switching under the radar with reactor. In *ACM/USENIX NSDI*, 2014.

[7] Christoforos Kachris, Konstantinos Kanonakis, and Ioannis Tomkos. Optical interconnection networks in data centers: recent trends and future challenges. *Communications Magazine, IEEE*, 2013.

[8] Nathan Farrington, George Porter, Yeshaiahu Fainman, George Papen, and Amin Vahdat. Hunting mice with microsecond circuit switches. In *Proc. ACM Workshop on Hot Topics in Networks*, 2012.

[9] Bin Wu and Kwan L Yeung. Nxg05-6: Minimum delay scheduling in scalable hybrid electronic/optical packet switches. In *Global Telecommunications Conference, 2006. GLOBECOM'06. IEEE*, pages 1–5. IEEE, 2006.

[10] Shoaib Kamil, Ali Pinar, Daniel Gunter, et al. Reconfigurable hybrid interconnection for static and dynamic scientific applications. In *Proceedings of the 4th international conference on Computing frontiers*, 2007.

[11] Howard Wang, Yiting Xia, Keren Bergman, TS Ng, Sambit Sahu, and Kunwadee Sripanidkulchai. Rethinking the physical layer of data center networks of the next decade: Using optics to enable efficient*-cast connectivity. *ACM SIGCOMM Computer Communication Review*, 2013.

[12] Mosharaf Chowdhury, Yuan Zhong, and Ion Stoica. Efficient coflow scheduling with varys. In *ACM SIGCOMM Computer Communication Review*, 2014.

[13] Yangming Zhao, Kai Chen, Wei Bai, et al. Rapier: Integrating routing and scheduling for coflow-aware data center networks. In *Computer Communications (INFOCOM), 2015 IEEE Conference on*, 2015.

[14] Zhen Qiu, Cliff Stein, and Yuan Zhong. Minimizing the total weighted completion time of coflows in datacenter networks. In *Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures*, 2015.

[15] Mosharaf Chowdhury and Ion Stoica. Coflow: A networking abstraction for cluster applications. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, 2012.

[16] Michael Isard et al. Dryad: distributed data-parallel programs from sequential building blocks. In *ACM SIGOPS Operating Systems Review*, 2007.

[17] Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, et al. Pregel: a system for large-scale graph processing. In *ACM SIGMOD*, 2010.

[18] Dhruba Borthakur. The hadoop distributed file system: Architecture and design. *Hadoop Project Website*, 2007.

[19] Mosharaf Chowdhury, Srikanth Kandula, and Ion Stoica. Leveraging endpoint flexibility in data-intensive clusters. In *ACM SIGCOMM Computer Communication Review*, 2013.

[20] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, 2012.

[21] Christo Wilson, Hitesh Ballani, Thomas Karagiannis, and Ant Rowtron. Better never than late: Meeting deadlines in datacenter networks. In *ACM SIGCOMM Computer Communication Review*, 2011.

[22] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 2008.

[23] Mohammad Alizadeh, Albert Greenberg, David A Maltz, et al. Data center tcp (dctcp). In *ACM SIGCOMM computer communication review*, 2010.

[24] Pawan Prakash, Advait Dixit, Y Charlie Hu, and Ramana Kompella. The tcp outcast problem: exposing unfairness in data center networks. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, 2012.

[25] Mosharaf Chowdhury, Matei Zaharia, Justin Ma, Michael I Jordan, and Ion Stoica. Managing data transfers in computer clusters with orchestra. *ACM SIGCOMM Computer Communication Review*, 2011.

[26] Fahad R Dogar, Thomas Karagiannis, Hitesh Ballani, and Antony Rowstron. Decentralized

task-aware scheduling for data center networks. In *ACM SIGCOMM Computer Communication Review*, 2014.

[27] Nick McKeown, Martin Izzard, Adisak Mekkittikul, William Ellersick, and Mark Horowitz. Tiny tera: a packet switch core. *Micro, IEEE*, 1997.

[28] Balaji Prabhakar and Nick McKeown. On the speedup required for combined input-and output-queued switching. *Automatica*, 1999.

[29] Mellanox SN2410 Switch System. http://www.mellanox.com/related-docs/prod_eth_switches/PB_SN2410.pdf.

[30] Arista 7280R Series Data Center Switch Router. https://www.arista.com/assets/data/pdf/Datasheets/7280R-DataSheet.pdf.

[31] Cisco Nexus 93180YC-EX Switch. http://www.cisco.com/c/en/us/products/collateral/switches/nexus-93108tc-ex-switch/datasheet-c78-736651.html.

[32] IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)*, pages 1–269, July 2008.

[33] Jonathan Perry, Amy Ousterhout, Hari Balakrishnan, Devavrat Shah, and Hans Fugal. Fastpass: A centralized zero-queue datacenter network. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 307–318. ACM, 2014.

[34] Pedro Moreira et al. White rabbit: Sub-nanosecond timing distribution over ethernet. In *2009 International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, pages 1–5. IEEE, 2009.

[35] Steven Blake, David Black, Mark Carlson, Elwyn Davies, Zheng Wang, and Walter Weiss. An architecture for differentiated services. 1998.

[36] CCITT Recommendation. I. 371: Traffic control and congestion control in b-isdn. *Geneva, Study Group*, 13, 1992.

[37] Cisco Tech Notes. Comparing traffic policing and traffic shaping for bandwidth limiting. *Document ID*, 19645:22–42.

[38] ITUT SGI. Traffic control and congestion control in b-isdn. *ITU-T Rec*, 1.

[39] John William Evans and Clarence Filsfils. *Deploying IP and MPLS QoS for Multiservice Networks: Theory & Practice*. Morgan Kaufmann, 2010.

[40] Cisco DocWiki. Resource Reservation Protocol. http://docwiki.cisco.com/wiki/Resource_Reservation_Protocol.

[41] Sergey Legtchenko, Nicholas Chen, Daniel Cletheroe, et al. Xfabric: a reconfigurable in-rack network for rack-scale computers. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, 2016.

[42] Kai Chen et al. OSA: An Optical Switching Architecture for Data Center Networks With Unprecedented Flexibility. *Networking, IEEE/ACM Transactions on*, 2014.

[43] Polatis 6000n Protection Services Switch Data Sheet. http://www.polatis.com/datasheets/products/Polatis_6000n_Protection_Services_Switch_Data_Sheet.pdf.

[44] Theophilus Benson, Aditya Akella, and David A Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, 2010.

[45] Yossi Azar and Iftah Gamzu. Efficient submodular function maximization under linear packing constraints. In *Automata, Languages, and Programming*. 2012.

[46] High-Density 25/100 Gigabit Ethernet StrataXGS Tomahawk Ethernet Switch Series. https://www.broadcom.com/products/ethernet-communication-and-switching/switching/bcm56960-series.

[47] Dan Alistarh, Hitesh Ballani, Paolo Costa, Adam Funnell, Joshua Benjamin, Philip Watts, and Benn Thomsen. A high-radix, low-latency optical switch for data centers. In *ACM SIGCOMM Computer Communication Review*, 2015.

[48] SDN-Enabled All-Optical Circuit Switching: An Answer to Data Center Bandwidth Challenges. https://www.sdxcentral.com/wp-content/uploads/2015/02/Polatis-SDN-Enabled-All-Optical-Circuit-Switching.pdf.

[49] Mike Mesnier, Gregory R Ganger, and Erik Riedel. Object-based storage. *IEEE Communications Magazine*, 2003.

[50] Kevin J Barker, Alan Benner, Ray Hoare, et al. On the feasibility of optical circuit switching for high performance computing systems. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*. IEEE Computer Society, 2005.

[51] George Porter, Richard Strong, Nathan Farrington, Alex Forencich, et al. Integrating microsecond circuit switching into the data center. In *Proc. ACM SIGCOMM.*, 2013.

[52] Navid Hamedazimi, Zafar Qazi, Himanshu Gupta, et al. Firefly: a reconfigurable wireless data center fabric using free-space optics. *ACM SIGCOMM Computer Communication Review*, 2015.

[53] Monia Ghobadi, Ratul Mahajan, Amar Phanishayee, et al. Projector: Agile reconfigurable data center interconnect. In *ACM SIGCOMM*, 2016.