# *Costly circuits, submodular schedules and approximate Carathéodory Theorems*



**Citation:** Bojja Venkatakrishnan, Shaileshh, et al. “Costly Circuits, Submodular Schedules and Approximate Carathéodory Theorems.” Queueing Systems, vol. 88, no. 3–4, Apr. 2018, pp. 311–47.

**Published Version:** http://dx.doi.org/10.1007/s11134-017-9546-x

**Publisher:** Springer US

**Permanent Link:** http://hdl.handle.net/1721.1/116926

**Version:** Author's final manuscript: final author’s manuscript post peer review, without publisher’s formatting or copy editing

**Noname manuscript No.**
(will be inserted by the editor)

# Costly Circuits, Submodular Schedules and Approximate Carathéodory Theorems

**Shaileshh Bojja Venkatakrishnan · Mohammad Alizadeh · Pramod Viswanath**

Received: date / Accepted: date

**Abstract** Hybrid switching – in which a high bandwidth circuit switch (optical or wireless) is used in conjunction with a low bandwidth packet switch – is a promising alternative to interconnect servers in today's large scale data centers. Circuit switches offer a very high link rate, but incur a non-trivial reconfiguration delay which makes their scheduling challenging. In this paper, we demonstrate a lightweight, simple and nearly-optimal scheduling algorithm that trades-off reconfiguration costs with the benefits of reconfiguration that match the traffic demands. Seen alternatively, the algorithm provides a fast and approximate solution towards a constructive version of Carathéodory's Theorem for the Birkhoff polytope. The algorithm also has strong connections to submodular optimization, achieves a performance at least half that of the optimal schedule and strictly outperforms state of the art in a variety of traffic demand settings. These ideas naturally generalize: we see that indirect routing leads to exponential connectivity; this is another phenomenon of the power of multi-hop routing, distinct from the well-known load balancing effects.

**Keywords** Data center networks · Bridges and switches · Circuit networks · Network flows · Submodular optimization · Approximation algorithms

**Mathematics Subject Classification (2000)** 68W25 · 68M12 · 68M20

Shaileshh Bojja Venkatakrishnan
University of Illinois Urbana-Champaign
E-mail: bjjvnkt2@illinois.edu

Mohammad Alizadeh
Massachusetts Institute of Technology
E-mail: alizadeh@csail.mit.edu

Pramod Viswanath
University of Illinois Urbana-Champaign
E-mail: pramodv@illinois.edu

## 1 Introduction

Modern data centers are massively scaling up to support demanding applications such as large-scale web services, big data analytics, and cloud computing. The computation in these applications is distributed across tens of thousands of interconnected servers. As the number and speed of servers increases,[1] providing a fast, dynamic, and economic switching interconnect in data centers constitutes a topical networking challenge. Typically, data center networks use multi-rooted tree designs: the servers are arranged in racks and an Ethernet switch at the top of the rack (ToR) connects the rack of servers to one or more aggregation (or spine) layers. These designs use multiple paths between the ToRs to deliver uniform high bisection bandwidth, and consist of a large number of high speed electronic packet switches that provide fine-grained switching capabilities but at poor speed/cost ratios.

Recent work has proposed the use of high speed circuit switches based on optical [54,17,60] or wireless [32,63,29] links to interconnect the ToRs. These architectures enable a dynamic topology tuned to actual traffic patterns, and can provide a much higher aggregate capacity than a network of electronic switches at the same price point, consume significantly less power, and reduce cabling complexity. For instance, Farrington et al [16] report 2.8×, 6× and 4.7× lower cost, power, and cabling complexity respectively using optical circuit switching relative to a baseline network of electronic switches.

The drawback of circuit switches, however, is that their switching configuration time is much slower than electronic switches. Depending on the specific technology, reconfiguring the circuit switch can take a few milliseconds (e.g., for 3D MEMS optical circuit switches [54,17,60]) to tens of microseconds (e.g., for 2D MEMS wavelength-selective switches [48]). During this reconfiguration period, the circuit switch cannot carry any traffic. By contrast, electronic switches can make per-packet switching decisions at sub-microsecond timescales. This makes the circuit switch suitable for routing stable traffic or bursts of packets (e.g., hundreds to thousands of packets at a time), but not for sporadic traffic or latency sensitive packets. A natural approach is then to have a *hybrid* circuit/packet switch architecture: the circuit switch can handle traffic flows that have heavy intensity but also require sparse connections, while a lower capacity packet switch handles the complementary (low intensity, but densely connected) traffic flows [17].

With this hybrid architecture, the relatively low intensity traffic is taken care of by the packet switch — switch scheduling here can be done dynamically based on the traffic arrival and is a well studied topic [43,33,42]. On the other hand, scheduling the circuit switch, based on the heavy traffic demand matrix, is still a fundamental unresolved question. Consider an architecture where a centralized scheduler samples the traffic requirements at each of the ToR ports at regular intervals ($W$, of the order of 100$\mu$s–1ms), and looks

[1] Servers with 10Gbps network interfaces are common today and 40/100Gbps servers are being deployed.

to find the schedule of circuit switch configurations over the interval of $W$ that is "matched" to the traffic requirements. The challenge is to balance the overhead of reconfiguring the circuits with the capability to be flexible and meet the traffic demand requirements.

The centralized scheduler must essentially decide a sequence of *matchings* between sending and receiving ToRs which the circuit switch then implements. For an optical circuit switch, for instance, the switch realizes the schedule by appropriately configuring its MEMS mirrors. As another example, in a broadcast-select optical ring architecture [10], the ToRs implement the controller's schedule by tuning in to the appropriate wavelength to receive traffic from their matching sender as dictated by the schedule.

Hence, we need a scheduling algorithm that decides the state (i.e., matching) of the circuit switch at each time and also a routing protocol to decide on an appropriate (direct or indirect) route packets can take to reach their destination ToR port. This is a challenging problem and entails making several choices on: (a) number of matchings, (b) choice of matchings (switch configuration), (c) durations of the matchings and (d) the routing protocol, in each interval $W$. Mathematically, this leads to a well defined optimization problem, albeit involving both combinatorial and real-valued variables. Even special cases of this problem [37] are NP hard to solve exactly.

Central to understanding this scheduling problem is finding a good sparse representation of the traffic matrix – a fundamental algorithmic question in Carathéodory's Theorem that has remained largely unanswered so far [45]. Recent papers have proposed heuristic algorithms to address this scheduling problem. In Solstice [40], the authors present a greedy perfect-matching based heuristic for a hybrid electrical-optical switch. Experimental evaluations show Solstice performing well over a simple baseline (where the schedules are provided by a truncated Birkhoff-von Neumann decomposition of the traffic matrix), although no theoretical guarantees are presented. Indirect routing in a distributed setting, but without considerations of configuration switching costs, is studied in another recent work [10].

### 1.1 Our Contributions

We first focus on routing policies where packets are sent from the source port to the destination port only via a direct link connecting the two ports, leading to *direct* or *single-hop* routing.

**Approximate Carathéodory's Theorem:** Our main result here is an approximately optimal, very simple and fast algorithm for computing the switch schedule in each interval. In turn this corresponds to a fast algorithm for computing a sparse approximate representation of a point on the Birkhoff polytope [11]. While Carathéodory's Theorem guarantees the *existence* of such a representation, an efficient *algorithm* to compute it has remained elusive so far. Our algorithm, which we christen Eclipse, has a performance that is at least half that of optimal for every instance of the traffic demands, and

experimentally shows a strict and consistent improvement over the state-of-the-art [40]. A key technical contribution here is the identification of a submodularity structure [5] in the problem, which allows us to make connections between submodular function maximization and the circuit switch scheduling problem with reconfiguration delay.

**Indirect Routing:** Next, we consider routing polices where packets are allowed to reach their destination after (potentially) transiting through many intermediate ports, leading to *indirect* or *multi-hop* routing. This class of routing policies is motivated by our observation that if the number of matchings is limited, multi-hop routing can *exponentially* improve the *reachability* of nodes; a novel benefit of multi-hop routing distinct from the classical and well known load balancing effects [50, 57, 26]. We again identify submodularity in the problem, but the constraints for this submodular maximization problem are no longer linear and efficient solutions are challenging to find. However, for the important special case where the sequence of switch configurations have already been calculated (and the indirect routing policy has to be decided) we propose a simple and fast greedy algorithm that is near-optimal universally for all traffic requirements. Detailed simulation results demonstrate strong improvements over direct routing, which are especially pronounced when the switch reconfiguration delays are relatively large. We also propose fast heuristics for the case where the switch configurations are not pre-calculated and provide key insights towards solving the general indirect routing problem.

The paper is organized as follows. In Section 2, the model, framework and the problem objective are formally stated along with a succinct summary of the state of the art. Section 3 focuses on direct routing and Section 4 on indirect routing. In Section 5, we present a detailed evaluation of the proposed algorithms on a variety of traffic inputs. Section 6 closes with a brief discussion. Technical aspects of the algorithm and its evaluation, including connections to submodularity and combinatorial optimization problems are deferred to the Appendix.

## 2 System Model

In this section, we present our model for a hybrid circuit-packet switched network fabric, and formally define our scheduling problem. Our model closely follows [40].

### 2.1 Hybrid Switch Model

We consider an $n$-port network where each port is simultaneously connected to a circuit switch and a packet switch as shown in Figure 1. A set of nodes are attached to the ports and communicate over the network. The nodes could either be individual servers or top-of-rack switches.

We model the circuit switch as an $n \times n$ crossbar comprising of $n$ input ports and $n$ output ports. At any point in time, each input port can send

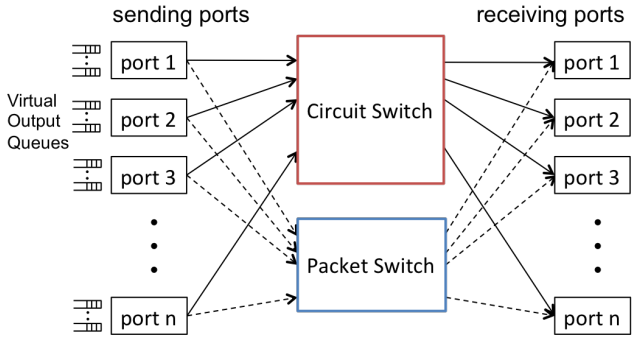



**Fig. 1** An illustration of our hybrid switch architechture.

packets to at most one output port and each output port can receive packets from at most one input port over the circuit switch. The circuit switch can be reconfigured to change the input-output connections. We assume that the packets at the input ports are organized in virtual-output-queues [49] (VOQ) which hold packets destined to different output ports.

In practice, the circuit switch is typically an optical switch [54,17,60].[2] These switches have a key limitation: changing the circuit configuration imposes a *reconfiguration delay* during which the switch cannot carry any traffic. The reconfiguration delay can range from few milliseconds to tens of microseconds depending on the technology [48,39]. This makes the circuit switch suitable for routing stable traffic or bursts of packets (e.g., hundreds to thousands of packets at a time), but not for sporadic traffic or latency sensitive packets. Therefore, hybrid networks also use a (electrical) packet switch to carry traffic that cannot be handled by the circuit switch. The packet switch operates on a packet-by-packet basis, but has a much lower capacity than the circuit switch. For example, the circuit and packet switches might respectively run at 100Gbps and 10Gbps per port.

We divide time into slots, with each slot corresponding to a (full-sized) packet transmission time on the circuit switch. We consider a *scheduling window* of $W \in \mathbb{Z}$ time units. A central controller uses measurements of the aggregated traffic demand between different ports to determine a *schedule* for the circuit switch at the start of each scheduling window. The schedule comprises of a sequence of configurations and how long to use each configuration (Section 2.3). The controller communicates the schedule to the circuit switch, which then follows the schedule for the next scheduling window (W) without involving the controller. We assume that the delay for each reconfiguration is $\delta \in \mathbb{Z}$ time units.

[2] Designs based on point-to-point wireless links have also been proposed [32,63]. Our abstract model is general.

### 2.2 Traffic Demand

Let $T \in \mathbb{Z}^{n \times n}$ denote the accumulated traffic at the start of a scheduling window. We assume $T$ is a feasible traffic demand, i.e., $T$ is such that $\sum_{j=1}^{n} T(i,j) \leq W$ and $\sum_{i=1}^{n} T(i,j) \leq W$ for all $i, j \in \{1, 2, \ldots, n\}$. The $(i,j)$th entry of $T$ denotes the amount of traffic that is in the VOQ at node $i$ destined for node $j$.

We assume that the controller knows $T$.[3] We also assume that non-zero entries in the traffic matrix $T$ are bounded as $2\delta \leq T(i,j) \leq \epsilon W$ for all $i, j \in [n] : T(i,j) > 0$ and some parameter $0 < \epsilon < 1$. This is a mild condition because traffic between pairs of ports that is small relative to $\delta$ is better served by the packet switch anyway.

Previous measurement studies have shown that the inter-rack traffic in production data centers is sparse [40, 8, 3, 51]. Over short periods of time (e.g., 10s of milliseconds), most nodes communicate with only a small number of other nodes (e.g., few to low tens). Further, in many cases, a large fraction of the traffic is sent by a small fraction of "elephant" flows [3]. While our algorithms and analysis are general, it is important to note that such sparse traffic patterns are necessary for hybrid networks to perform well (especially with larger reconfiguration delay).

### 2.3 The Scheduling Problem

Given the traffic demand, $T$, our goal is to compute a schedule that maximizes the total amount of traffic sent over the circuit switch during the scheduling window $W$. This is desirable to minimize the load on the slower packet switch. In general, the scheduling problem involves two aspects:

**1. Determining a schedule of circuit switch configurations:** The algorithm must determine a sequence of circuit switch configurations: $(\alpha_1, P_1)$, $(\alpha_2, P_2), \ldots, (\alpha_k, P_k)$. Here, $\alpha_i \in \mathbb{Z}$ denotes the duration of the $i^{th}$ switch configuration, and $P_i$ is an $n \times n$ permutation matrix, where $P_i(s,t) = 1$ if input port $s$ is connected to output port $t$ in the $i^{th}$ configuration. For a valid schedule, we must have $\alpha_1 + \alpha_2 + \ldots + \alpha_k + k\delta \leq W$ since the total duration of the configurations cannot exceed the scheduling window $W$.

**2. Deciding how to route traffic:** The simplest approach is to use only direct routes over the circuit switch. In other words, each node only sends traffic to destinations to which it has a direct circuit during the scheduling window. Alternatively, we can allow nodes to use indirect routes, where some traffic is forwarded via (potentially multiple) intermediate nodes before being delivered to the destination. Here, the intermediate nodes buffer traffic in their VOQs for transmission over a circuit in a subsequent configuration.

[3] Our work is orthogonal to how the controller obtains the traffic demand estimate. For example, the nodes could simply report their backlogs before each scheduling window, or a more sophisticated prediction algorithm could be used.

In the next section, we begin by formally defining the problem in the simpler setting with direct routing and developing an algorithm for this case. Then, in Section 4, we consider the more general setting with indirect routing.

**Remark 1.** Prior work [40,37] has considered the objective of *covering* the entire traffic demand in the least amount of time. For example, the ADJUST algorithm in [37] takes the traffic demand $T$ as input and computes a schedule $(\alpha_1, P_1), \ldots, (\alpha_k, P_k)$ such that $\sum_{i=1}^{k} \alpha_i + k\delta$ is minimized while $\sum_{i=1}^{k} \alpha_i P_i \geq T$. Our formulation (and solution) is more general, since an algorithm which maximizes throughput over a given time period can also be used to find the shortest duration to cover the traffic demand (e.g., via binary search).

**Remark 2.** From a systems viewpoint, the traffic demand estimation can be done either by directly polling ToR switches or end-host NICs [39,60], or indirectly through applications and flow information [17,2]. For systems at scale, this represents a non-trivial task (and often taking 100s of $\mu$s [39]) necessitating the need for a window $W$ to compute the schedule (vis-à-vis dynamic policies; see following Section 2.4).

### 2.4 Related Work

Before presenting the work in this paper, we briefly summarize related work on this topic. Scheduling in crossbar switches is a classical and well studied topic in queuing theory. Traditionally the crossbar has been used to model packet switches where the reconfiguration delay is very small. Hence the scheduling solutions proposed – ranging from centralized Birkhoff-von-Neumann decomposition scheduler [43] on one end to the decentralized load-balanced scheduler [12] on the other – did not account for reconfiguration delay. With the proposals on hybrid circuit/packet switching systems [17,60], simplified models that factor for the reconfiguration delay were considered. A variant of the well known MaxWeight algorithm is presented in [59] and is shown to be throughput optimal. Fixed-Frame MaxWeight (FFMW) is a frame based policy proposed in [38] and has good delay performance. However it requires the arrival statistics to be known in advance. A hysteresis based algorithm that adapts many previously proposed algorithms for crossbar switch scheduling to the case with reconfiguration delay is presented in [58]. All of these works are "dynamic" policies where scheduling decisions are made time-slot by time-slot and the analyses are probabilistic. They also require perfect queue state information at every instant.

Another research direction is to consider "batch" policies [58] in which each computational call returns a schedule for an entire window of time. Research works in this category are often analyzed combinatorially or via real-world system evaluations. Early works often assumed the delay to be either zero [31] or infinity [56,61]. The infinite delay setting corresponds to a problem where the number of matchings is minimized. However they still require $O(n)$ matchings. In a different context (satellite-switched time-division multiple access), works such as [25] also computed schedules that minimized the number of

matchings. Moderate reconfiguration delays are considered in DOUBLE [56] and other algorithms such as [22,37,62] that explicitly take reconfiguration delay into account. The algorithm ADJUST [37] minimizes the covering time but still requires around $n$ configurations. All of these algorithms do not benefit from sparse demands and continue to require $O(n)$ configurations [40]. In a complementary approach, [13] considers conditions on the input traffic matrix under which efficient polynomial time algorithms to compute the optimal schedule exists. Yet other approaches have been to introduce speedup [42], or randomization in the algorithms [24], however they do not address the basic optimization problem underlying this scenario head-on. Such is the goal of this paper.

## 3 Direct Routing

The centralized scheduler samples the ToR ports and arrives at the traffic demand (matrix) $T$ to be met in the upcoming slot. In this section, we develop an algorithm, named Eclipse, that takes the traffic demand, $T$, as input and computes a schedule of matchings (circuit configurations) and their durations to maximize throughput over the circuit switch; only direct routing of packets from source to destination ports are allowed here. Eclipse is fast, simple and nearly-optimal in every instance of the traffic matrix $T$. Towards a formal understanding of the notion of optimality, consider the following optimization problem:

$$\boxed{\begin{aligned} \text{maximize} \quad & \left\| \min\left( \sum_{i=1}^{k} \alpha_i P_i, T \right) \right\|_1 \\ \text{s.t.} \quad & \alpha_1 + \alpha_2 + \ldots + \alpha_k + k\delta \leq W \\ k \in \mathbb{N}, P_i \in \mathcal{P}, & \alpha_i \geq 0 \ \forall i \in \{1, 2, \ldots, k\}, \end{aligned}} \tag{1}$$

where $\mathbb{N} = \{1, 2, \ldots\}$ and $\mathcal{P}$ is the set of permutation matrices.

This optimization problem is NP-hard [37], and a recent work [40] in the literature has focused on heuristic solutions. Our proposed algorithm has some similarities to the prior work in [40] in that the matchings and their durations are computed successively in a *greedy* fashion. However, the algorithm is overall quite different in terms of both ideas and details; we uncover and exploit the underlying *submodularity* [52] structure inherent in the problem to design and analyze the algorithm in a principled way.

We also note that this problem can be viewed as finding permutation matrices $P_1, \ldots, P_k$ and weightings $\alpha_1, \ldots, \alpha_k$ such that their weighted sum is a good *approximation* of the traffic matrix $T$. Carathéodory's Theorem applied to the Birkhoff polytope guarantees the *existence* of such a dual representation; however till date we do not know of an efficient *algorithm* to compute this representation. A recent work [6] proposes an approximation algorithm, but it relies on an exhaustive search over the vertices of the polytope which

**Algorithm 1:** A general greedy algorithm template

**Input** : Traffic demand $T$, reconfiguration delay $\delta$ and scheduling window size $W$
**Output**: Sequence of matchings $P_1, \ldots, P_k$ and their corresponding durations $\alpha_1, \ldots, \alpha_k$:

```
sch ← {} ;                                              // schedule
k ← 0 ;
T_rem ← T ;                                             // traffic remaining
while Σ_{i=1}^{k} (α_i + δ) ≤ W do
    k ← k + 1;
    Decide on a duration α for the matching;
    M ← argmax_{M∈M} ‖min(αM, T_rem)‖_1 ;
    sch ← sch ∪ {(α, M)} ;
    T_rem ← T_rem − min(αM, T_rem) ;
end
if Σ_{i=1}^{k} (α_i + δ) > W then
    sch ← sch\{(α, M)};
end
k ← k − 1;
```

can be very slow (moreover in the case of the Birkhoff polytope, the number of vertices is also exponential in the dimension). Our approach does not involve such a search, and is also very efficient in obtaining an approximate Carathéodory expansion.

### 3.1 Intuition

Before a formal presentation and analysis of the algorithm, we begin with an intuitive and less-formal approach to how one might solve this optimization problem. Consider greedy algorithms with the template shown in Algorithm 1. The template starts with an empty schedule, and proceeds to add a new matching to the schedule in each iteration. This process continues until the total duration of the matchings exceeds the allotted time budget of $W$, at which point the algorithm terminates and outputs the schedule computed so far. In each iteration, the algorithm first picks the duration of the matching, $\alpha$. It then selects the *maximum weight matching* in the traffic graph whose edge weights are thresholded by $\alpha$ (i.e., edge weights $> \alpha$ are clipped to $\alpha$). The traffic graph is a bipartite graph between $n$ input and $n$ output vertices, with an edge of weight $T(i, j)$ between input node $i$ and output node $j$. It remains to specify how to choose $\alpha$ in each iteration.

Consider an exercise where we vary the matching duration $\alpha$ from 0 to $W$ and compute the maximum weight matching in the thresholded traffic graph for each $\alpha$. For a typical traffic matrix, this results in a curve similar to the solid-blue line in Fig. 2. Notice that the value of the maximum weight matching is precisely equal to the sum-throughput that can be achieved in that round of the switch schedule. It is straightforward to see that the maximum weight matching curve has the following properties: (a) it is non-decreasing and (b) piecewise linear. These are explained as follows: when $\alpha$ is very small

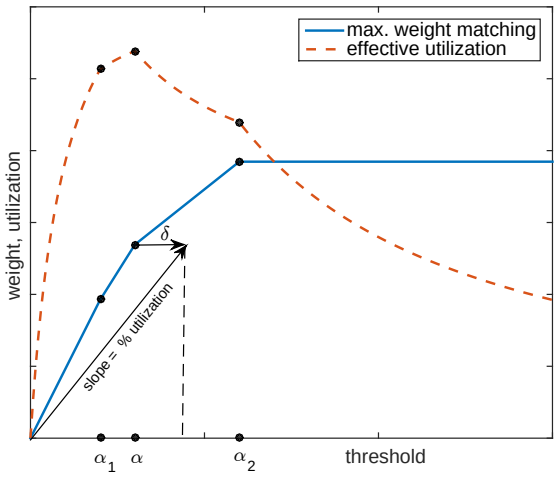



**Fig. 2** Throughput of max. weight matching as a function of threshold duration. The effective utilization curve of the matchings is also shown.

a lot of the edges in the traffic graph have a weight that is saturated at $\alpha$. Hence it is likely to find a perfect matching with total weight of $n\alpha$. As such the slope of the curve when $\alpha$ is small is $n$. However, as $\alpha$ becomes large there are increasingly fewer edges whose weights are saturated at $\alpha$ and, correspondingly, the slope reduces. When $\alpha$ is so large that all of the edge weights are strictly smaller than $\alpha$, then the value of the maximum weight matching does not change even with any further increase in $\alpha$ and the curve ultimately flattens out.

Two operating points of interest, considering Fig. 2, are (a) the largest $\alpha$ where the slope of the curve is maximum ($= n$ in the typical case where every ingress/egress port has traffic) and (b) the smallest $\alpha$ where the value of the maximum weight matching is the largest. These points have been denoted by $\alpha_1$ and $\alpha_2$ in Fig. 2 respectively. Setting $\alpha = \alpha_1$ is interesting because it results in a matching where the links are all fully utilized. For example, the Solstice algorithm presented in [40] implicitly adopts this operating point. On the other hand, $\alpha = \alpha_2$ gives a matching that achieves the largest possible sum-throughput in that round.

However we note that both choices of $\alpha$ are less than ideal for the following reasons. Recall that after every round of switching we incur a delay of $\delta$ time units. As such if the value of $\alpha_1$ is small (say comparable to $\delta$) in each round, then the number of matchings, and hence the time wasted due to the reconfiguration delay, becomes large. As a concrete example, consider the transpose of the traffic matrix $T_1 = [A_1^t \mathbf{b}_1^t]$ where $A_1$ is a sparse $(n-1) \times n$ matrix and $\mathbf{b} = [2\delta, 2\delta, \ldots, 2\delta, 0, 0, \ldots, 0]$ comprises of some $k$ entries of value $2\delta$ and $n-k$ entries of value 0. In other words, we are considering an input where a

node or a collection of nodes have a large number of small flows to a particular node or vice-versa. For such an instance it is clear that if we insist on matchings with 100% utilized links, then the maximum duration of the matching is $2\delta$ (i.e., $\alpha_1 = 2\delta$). Thus, continuing the process described in Algorithm 1 results in a sequence of $k$ matchings each of which is only $2\delta$ time units long. Hence in the worst case (if $k > 1/(3\delta)$) about 1/3rd of the entire scheduling window is wasted just due to reconfiguration delay limiting the maximum possible throughput to $2n/3$. On the other hand, if we had ignored the entries in $\mathbf{b}$, then we could have scheduled just $A_1$ achieving a total throughput of $n - 2k\delta \approx n$ for large $n$. We point out that the phenomenon described above happens in a large family of instances, of which $T_1$ is a specific example. We also emphasize that such instances are pretty likely to occur in practice; for example, [51, Fig.5-b] shows traffic measurements in a Facebook data center where the interactions between Cache and Web servers lead to traffic matrices having this property.

Similarly for the operating point with $\alpha = \alpha_2$, consider the traffic matrix $T_2 = \begin{bmatrix} A_2 & \mathbf{0} \\ \mathbf{0} & B_2 \end{bmatrix}$ where $A_2$ is a sparse $(n-2) \times (n-2)$ matrix and $B_2 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$. This is a diametrically opposite situation from $T_1$ where a small collection of nodes interact only amongst themselves with no interaction outside. Such a situation occurs, for example, in multi-tenant cloud-computing data centers [53] where individual tenants run their jobs on small clusters of servers. In such a case, the value of the maximum weight matching can be maximum for a large $\alpha$. For $T_2$ the maximum value occurs at $\alpha = 1 - \delta$ (i.e., $\alpha_2 = 1 - \delta$), resulting in a schedule with just one matching of duration $1 - \delta$ and potentially missing a lot of traffic for $A_2$. For example, if $A_2$ is uniformly $k$-sparse, we miss out roughly $(k-1)n/k$ units of traffic. On the other hand, by choosing the duration of the matching to be $1/k - \delta$ in each step we can achieve a sum throughput of $n - O(\delta) \approx n$.

In scenarios exemplified by $T_2$, setting $\alpha = 1$ is bad because the utilization of the resulting matching is poor, i.e., a vast majority of the matching links carry only a fraction of their capacity. This can be overcome by insisting that we choose only those matchings with utilization of at least 75% (say). However, in the case of $T_1$ we observe a poor performance in spite of all matchings having a utilization of 100%. The issue in this case is that the duration of the matchings are small compared to the reconfiguration delay. Hence to avoid this scenario we can insist on $\alpha \geq 20\delta$ (say) in Algorithm 1.

Our first main observation is that both of the above heuristics are captured if we consider the *effective utilization* of the matchings. We define effective utilization as the ratio $\text{mwm}(\alpha)/(\alpha + \delta)$ where $\text{mwm}(\alpha)$ denotes the value of the maximum weight matching at $\alpha$. This ratio indicates the overall efficiency of a matching by including the reconfiguration delay into the duration. In Fig. 2 we plot the effective utilization of the matchings as the red-dotted curve. As can be seen there, the effective utilization at both $\alpha_1$ and $\alpha_2$ is suboptimal. We propose an algorithm that selects $\alpha$ to maximize effective utilization; a detailed description is deferred to Section 3.3.

The justification for selecting matchings according to the above is further reinforced by the *submodularity* structure of the problem (we discuss submodularity in Section 3.2). It turns out that for a certain class of submodular maximization problems with linear packing constraints, greedy algorithms take a form that precisely matches the intuitive thought process above [5]: the proposed intuitively correct algorithm is borne out naturally from submodular combinatorial optimization theory. We briefly recall relevant aspects of submodularity and associated optimization algorithms next.

### 3.2 Submodularity

A set function $f : 2^{[n]} \rightarrow \mathbb{R}$ is said to be *submodular* if it has the following property: for every $A, B \subseteq [n]$ we have $f(A \cup B) + f(A \cap B) \leq f(A) + f(B)$. Alternatively, submodular functions are also defined through the property of *decreasing marginal values*: for any $S, T$ such that $T \subseteq S \subseteq [n]$ and $j \notin S$, we have

$$f(S \cup \{j\}) - f(S) \leq f(T \cup \{j\}) - f(T).$$

The difference $f(S \cup \{j\}) - f(S)$ is called the incremental marginal value of element $j$ to set $S$ and is denoted by $f_S(j)$. For our purpose we will only focus on submodular functions that are *monotone* and *normalized*, i.e., for any $S \subseteq T \subseteq [n]$ we have $f(S) \leq f(T)$ and further $f(\{\}) = 0$.

Many applications in computer science involve maximizing submodular functions with *linear packing constraints*. This refers to problems of the form:

$$\max f(S) \quad \text{s.t. } A\mathbf{x}_S \leq b \text{ and } S \subseteq [n],$$

where $A \in [0,1]^{m \times n}, b \in [1, \infty)^m$ and $\mathbf{x}_S$ denotes the characteristic vector of the set $S$. Each of the $A_{ij}$'s is a cost incurred for including element $j$ in the solution. The $b_i$'s represent a total budget constraint. A well-known example of a problem in the above form is the Knapsack problem (the objective function in this case is in fact modular).

With the above background, we formulate the optimization problem under direct routing as one of submodular function maximization. Recall that for any given input traffic matrix $T$, the schedule that is computed is described by a sequence of matchings and corresponding durations. Consider the set $\mathcal{M}$ of all perfect matchings in the complete bipartite graph $K_{n \times n}$ with $n$ nodes in each partite. Then any round in the schedule is simply $(\alpha, P) \in \mathbb{Z} \times \mathcal{M}$. The key observation we make now is to view the schedules as a subset of $\mathbb{Z} \times \mathcal{M}$. Formally, define a *switch schedule* as any subset $\{(\alpha_1, M_1), \ldots, (\alpha_k, M_k)\}$ of $\mathbb{Z} \times \mathcal{M}$. The objective function in our case is the sum-throughput defined as

$$f(\{(\alpha_1, M_1), \ldots, (\alpha_k, M_k)\}) = \left\| \min\Big(\sum_{i=1}^{k} \alpha_i M_i, T\Big) \right\|_1, \tag{2}$$

**Algorithm 2:** Eclipse: greedy direct routing algorithm

**Input** : Traffic demand $T$, reconfiguration delay $\delta$ and scheduling window size $W$
**Output**: Sequence of matchings $P_1, \ldots, P_k$ and their corresponding durations $\alpha_1, \ldots, \alpha_k$:

```
sch ← {} ;                                              // schedule
k ← 0 ;
T_rem ← T ;                                             // traffic remaining
while Σ_{i=1}^{k} (α_i + δ) ≤ W do
    k ← k + 1;
    (α, M) ← argmax_{M∈M, α∈Z} ||min(αM, T_rem)||_1 / (α+δ) ;
    sch ← sch ∪ {(α, M)} ;
    T_rem ← T_rem − min(αM, T_rem) ;
end
if Σ_{i=1}^{k} (α_i + δ) > W then
    sch ← sch\{(α, M)};
end
k ← k − 1 ;
```

where the minimum is taken entrywise and $\|\cdot\|_1$ refers to the entrywise $L_1$-norm of the matrix. We observe that the function $f$ is submodular, deferring the proof to the Appendix.

**Theorem 1** *The function $f : 2^{\mathbb{Z}\times\mathcal{M}} \to \mathbb{R}$ defined by Equation (2) is a monotone, normalized submodular function.*

We have established that optical switch scheduling under the sum-throughput metric is a submodular maximization problem. With this, we are ready to present a greedy algorithm that achieves a sum-throughput of at least a constant factor of the optimal algorithm for *every* instance of the traffic matrix.

### 3.3 Algorithm

Algorithm 2 – Eclipse – captures our proposed solution under direct routing. Eclipse takes the traffic matrix $T$, the time window $W$ and reconfiguration delay $\delta$ as inputs, and computes a sequence of matchings and durations as the output. The algorithm proceeds in rounds (the "while loop"), where in each round a new matching is added to the existing sequence of matchings. The sequence terminates whenever the sum of the matching durations exceeds the allocated time window $W$ or whenever the traffic matrix $T$ is fully covered.

Consider any round $t$ in the algorithm; let $(\alpha_1, M_1), \ldots, (\alpha_{t-1}, M_{t-1})$ denote the schedule computed so far in $t-1$ rounds (stored in variable `sch`) and let $T_{\text{rem}}(t)$ denote the amount of traffic yet to be routed. The matching that is selected in the $t$-th round is the one for which *utilization* – the percentage of the total matching capacity that is actually used – is maximum. Mathematically, we choose an $(\alpha, M)$ pair such that $\frac{\|\min(\alpha M, T_{\text{rem}})\|_1}{\alpha+\delta}$ is maximized. In the Appendix we have given a proof that the maximum (for $\alpha$) occurs on the support of $T_{\text{rem}}$. Hence this can be easily found by looking at

**Algorithm 3:** Finding the greedy maximum

**Input** : Traffic demand $T$, reconfiguration delay $\delta$
**Output**: $(\alpha, M) \in \mathbb{Z} \times \mathcal{M}$ such that $(\alpha, M) = \text{argmax}_{M \in \mathcal{M}, \alpha \in \mathbb{Z}} \frac{\|\min(\alpha M, T)\|_1}{(\alpha+\delta)}$
$H \leftarrow$ distinct entries of $T$ sorted in ascending order;
$i_{\text{lb}} \leftarrow 1$ and $i_{\text{ub}} \leftarrow \text{length}(H)$;
**while** $i_{\text{lb}} < i_{\text{ub}}$ **do**
  $i \leftarrow (i_{\text{lb}} + i_{\text{ub}})/2$ ;
  $T_1 \leftarrow \min\{T, H(i)\}$ ; // thresholding $T$ to $H(i)$
  $T_2 \leftarrow \min\{T, H(i+1)\}$ ;
  $v_1 \leftarrow$ (max. weight matching in $T_1$)$/(H(i)+\delta)$ ;
  $v_2 \leftarrow$ (max. weight matching in $T_2$)$/(H(i+1)+\delta)$ ;
  **if** $v_1 < v_2$ **then**
    $i_{\text{lb}} \leftarrow i$;
  **else if** $v_1 > v_2$ **then**
    $i_{\text{ub}} \leftarrow i$;
  **else**
    return ($H(i)$, max. weight matching in $T_1$);
  **end**
**end**

the support of the (sparse) matrix $T_{\text{rem}}$. We also propose a simple binary-search procedure, discussed in Algorithm 3, that finds only a local maximum but performs extremely well in our evaluations (Section 5). This process of selecting a matching is repeated in each round until the sum-duration of the matchings exceeds the scheduling window $W$, when the last chosen matching is discarded and the remaining set of matchings are returned. Eclipse is simple and also fast, a fact the following calculation demonstrates.

**Complexity:** We begin with the complexity of Algorithm 3. Since $i_{\text{ub}}$ is no more than the number of distinct entries of $T$, we have $i_{\text{ub}} \leq n^2$. In each iteration, the algorithm only considers entries of $H$ that have indices between $i_{\text{lb}}$ and $i_{\text{ub}}$. However, binary-search halves the effective size of $H$ (i.e., those numbers in $H$ with array indices $i_{\text{lb}}, i_{\text{lb}+1}, \ldots, i_{\text{ub}}$), and the number of iterations of the `while` loop is bounded by $\log n^2 = 2 \log n$. Within the `while` loop, computing the maximum weight matching can be done in $O(dn^{3/2} \log(W\epsilon))$ time (a basic fact of submodular optimization [15,52]) where $dn$ is the number of edges in bipartite graph formed by $T$ (i.e., $d$ is the average sparsity). Further $(1-\epsilon)$ approximate maximum weight matching can be computed in linear time, e.g. $O(dn\epsilon^{-1} \log \epsilon^{-1})$ [14], and efficient implementations in practice have been studied extensively in the literature [44,46,18]. Hence the overall time complexity is $O(dn^{3/2} \log n \log(W\epsilon))$. Now, in Algorithm 2 the number of iterations in the `while` loop is bounded by $W/\delta$. As such the total complexity of the algorithm is $\tilde{O}(dn^{3/2} \frac{W}{\delta})$. An exact search over the support of $T_{\text{rem}}$ in the maximization step results in a overall complexity of $\tilde{O}(d^2 n^{5/2} \frac{W}{\delta})$.

**Approximation Guarantee:** Since the proposed direct routing algorithm is connected to submodular maximization with linear constraints, we can adapt standard combinatorial optimization techniques to show an approximation factor of $1 - 1/e$. Let `OPT` denote the sum-throughput of the optimal algorithm

for given inputs $T, \delta$ and $W$. Let ALG2 denote the sum-throughput achieved by Eclipse. We then have the following.

**Theorem 2** *If the entries of $T$ are bounded by $\epsilon W + \delta$ then Eclipse approximates the optimal algorithm to within a factor of $1 - 1/e^{(1-\epsilon)}$, i.e.,* $\texttt{ALG2} \geq (1 - 1/e^{(1-\epsilon)})\texttt{OPT}$.

The proof of the above Theorem is deferred to the Appendix. As a concluding remark, we note that the constant $\epsilon$ in the approximation factor comes from the requirement that $\alpha + \delta \leq \epsilon W$ hold. We observe that this mild technical condition, required to show that Eclipse is a constant factor approximation of the optimal algorithm, has an added implication. Informally, it ensures that no single matching occupies the bulk of the scheduling window.

## 4 Indirect Routing

In the previous section, we focused on direct routing where packets are forwarded to their destination ports only if a link *directly* connecting the source port to the destination port appeared in the schedule – this is essentially a "single-hop" protocol. In this section, we explore allowing packets to be forwarded to (potentially) multiple intermediate ports before arriving at its final destination. In terms of implementing this more involved protocol, we note that there is no extra overhead needed: the destination of any received packet is read first upon reception and since the queues are maintained on a per-destination basis at each ToR port, any received packet can be diverted to the appropriate queue. The key point of allowing indirect routing is the vastly increased range of ports that can be reached from a small number of matchings.

Consider Fig. 3 which illustrates a 6-port network and a sequence of 3 consecutive matchings in the schedule. With direct routing, port 3 can only forward packets to ports 2, 5 and 4 in rounds 1, 2 and 3 respectively, i.e., the set of egress ports *reachable* by port 3 is $\{2, 4, 5\}$. In the indirect routing framework of this section, port 3 can also forward packets to port 1. This can be achieved by first forwarding the packets to port 2 in the first round where the packets are queued. Then in the second round we let port 2 forward those packets to the destination port 1. Thus the *reachability* of the nodes is enhanced by allowing for indirect routing. Indirect routing can also be viewed as "multi-hop" routing.

Traditionally multi-hop routing has been used as a means of *load balancing*. This is known to be true in the context of networks such as the Internet where the benefits of "Valiant load-balancing" are legion [50, 57, 26]. The benefits of load balancing are also well known in the switching context – a classic example is the two-stage load-balancing algorithm in crossbar switches without reconfiguration delay [55]. The benefit of multi-hop routing in our context is *markedly different*: the reachability benefits of indirect routing are especially well suited to the setting where input ports are directly connected to only a few output ports due to the *small number* of matchings in the scheduling window.

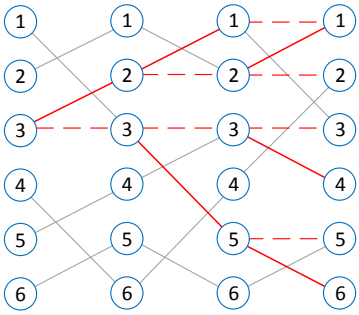



**Fig. 3** Reachability of nodes under multi-hop routing.

In fact, an elementary calculation shows that over a period of $k$ matchings in the schedule, indirect routing can allow a node to forward packets to $O(2^k)$ other nodes, compared to only $O(k)$ nodes possible with direct routing. This is because of the recursion $f(k) = 2f(k-1)+1$ where $f(k)$ denotes the number of nodes reachable by any node in $k$ rounds. If a node (say, node 1) can reach $f(k-1)$ nodes in $k-1$ rounds, then in the $k$-th round (i) there is a new node directly connected to node 1 and (ii) each of the $f(k-1)$ nodes can be connected to a new node. Thus the number of nodes connected to node 1 in the $k$-th round becomes $f(k-1) + (f(k-1)+1)$. Fig. 3 also illustrates this phenomenon where reachability from node 3 is shown. As a corollary we observe that $O(\log_2 n)$ rounds of matchings are sufficient to reach *all* other nodes in a $n$-port network.

As in the direct routing case, computing the optimal schedule remains a challenging problem. While it is clear that we can achieve a performance at least as good as with direct routing, the gain is different for each instance of the traffic matrix – precisely quantifying the gain in an instance-specific way appears to be challenging. Our main result here is that the submodularity property of the objective function continues to hold, provided the variables are considered in an appropriate format. However, unlike direct routing, we are able to use submodularity in this case and obtain fast, natural solutions only under settings where some of the variables are restricted (matchings and durations).

Notice that in general specifying a schedule entails computing (i) a sequence of matchings, (ii) their durations and (iii) a multi-hop routing policy. As steps towards jointly computing all the three quantities listed, we also present a sequence of heuristics, where we restrict different classes of the variables. In Eclipse++ we present a simple and fast greedy algorithm to compute the switching schedule under fixed matchings and durations. This heuristic is borne naturally from the submodularity of the problem and is shown to be approximately optimal for *each* instance of the traffic matrix. Next, in EclipseX we restrict only the matchings and jointly compute their durations and the routing policy. Finally, in Eclipse# we present a heuristic where none of

| Matchings | Durations | Routing | Algorithm |
|---|---|---|---|
| yes | yes | no | Eclipse |
| no | no | yes | Eclipse++ |
| no | yes | yes | EclipseX |
| yes | yes | yes | Eclipse# |

**Table 1** Summary of the solution variables that are computed by our proposed algorithms.

the variables are restricted. The restrictions followed in the above algorithms have been summarized in Table 1. We present these results, following the same format as in the direct routing section, leaving numerical evaluations to a later section. We follow the model as discussed in Section 2.

### 4.1 Submodularity of objective function

We first adopt an alternative way of describing the switch schedule by specifying the multi-hop *path* taken by each packet. Such a formulation serves us well in the *causal* structure of the routed traffic patterns that naturally occur here.

For simplicity let us fix the number of rounds $k$ in the schedule. Consider a fully connected $k$-round time-layered directed graph $G$ consisting of $k+1$ partites, $V_0, V_1, \ldots, V_k$ (of $n$ nodes each), with nodes in each partite $i$ having directed edges to all the nodes in partite $i+1$. Let $\mathcal{P}$ denote the set of all paths in $G$ that begin at a node in $V_0$ and end at a node in $V_k$. Any such $p \in \mathcal{P}$ describes a multi-hop route for a packet in the system. If we are able to choose a path for every packet in the traffic matrix $T$, subject to capacity constraints, then we have a valid sequence of switch configurations and routing policy for the schedule. Now, for a set of paths $(\beta_1, p_1), \ldots, (\beta_m, p_m)$, where $\beta_i$ denotes the number of packets sharing the same path $p_i$, consider the sum-throughput given by a function $f : 2^{\mathbb{Z}\times\mathcal{P}} \to \mathbb{Z}$ defined as $f(\{(\beta_1, p_1), \ldots, (\beta_m, p_m)\}) \triangleq$

$$\sum_{i,j\in[n]} \min\left(\sum_{l=1}^{m} \beta_l \mathbf{1}_{\substack{p_l(0)=i,\\ p_l(k+1)=j}}, T_{ij}\right), \tag{3}$$

where $p(0)$ and $p(k+1)$ denote the starting and ending nodes of path $p$ and $\mathbf{1}_{\{\cdot\}}$ is the indicator function. Then the main observation is that $f$ is submodular.

**Theorem 3** *The function $f : 2^{\mathbb{Z}\times\mathcal{P}} \to \mathbb{Z}$ defined by Equation (3) is submodular.*

The proof is analogous to Theorem 1 and is omitted. So far we have not imposed any restrictions on the set of paths that we choose for the schedule. This can be incorporated in the form of constraints to the problem, thus rephrasing the objective as a constrained submodular maximization problem.

**Constraints:** Since we can choose arbitrary weighted paths, we need constraints to ensure that
(i) the set of paths form a matching in each round and
(ii) the total duration of the matchings is at most $W - k\delta$.
This can be written mathematically as follows for any subset of weighted paths $\{(\beta_1, p_1), \ldots, (\beta_m, p_m)\} \in 2^{\mathbb{Z} \times \mathcal{P}}$ :

$$\sum_{\substack{e: v \in e, \\ e \in E_j}} \mathbf{1}\left\{ \sum_{i=1}^{m} \mathbf{1}_{\{e \in p_i\}} \beta_i > 0 \right\} \leq 1 \ \forall v \in V_{j-1}, j \in [k] \tag{4}$$

$$\sum_{\substack{e: v \in e, \\ e \in E_j}} \mathbf{1}\left\{ \sum_{i=1}^{m} \mathbf{1}_{\{e \in p_i\}} \beta_i > 0 \right\} \leq 1 \ \forall v \in V_j, j \in [k] \tag{5}$$

$$\sum_{j=1}^{k} \left( \left( \max_{e \in E_j} \sum_{i=1}^{m} \mathbf{1}_{\{e \in p_i\}} \beta_i \right) + \delta \right) \leq W \tag{6}$$

where $E_j$ stands for the edges between $V_{j-1}$ and $V_j$ in $G$. Hence we can express the problem as maximization of objective (3) subject to the constraints (4)–(6).

However, the key challenge here is that the constraints (4)–(6) are *nonlinear* – it is not clear whether an efficient (approximation) algorithm exists. The nonlinearities appear only in the sense of membership tests and a corresponding thresholding function – so it is possible that an efficient nearly-optimal greedy algorithm exists, but we leave this study for future work. We do note, however, that for the special case in which the configurations are fixed and we only have to decide on the indirect routing policies, the constraints take on a linear form – in this setting, we are able to construct fast and efficient greedy algorithms. This case represents a composition of direct routing (where switch schedules are computed) and indirect routing (where the multi-hop routing policies are described), and is discussed next.

**Multi-Hop Routing Policies:** Consider a fixed sequence $M_1, \ldots, M_k$ of switch configurations and an input traffic demand matrix $T$. Let $G$ denote the time-layered edge-capacitated graph obtained from the sequence of matchings, i.e., $G$ consists of $k+1$ partites $V_0, \ldots, V_k$ with $n$ nodes each, and $M_i$ is the matching between partites $V_{i-1}$ and $V_i$. In addition to the matching edges, there are also edges, with unlimited edge capacities, connecting the $j$-th nodes of $V_{i-1}$ and $V_i$ for all $j \in [n], i \in [k]$. In this setting, by constraining the total duration of the matchings, we can maximize our required objective by formulating the following *linear program* (LP) relaxation,

$$\begin{aligned} &\text{maximize} \quad \sum_{p \in \mathcal{P}} x_p \\ &\quad \text{s.t.} \quad \sum_{p \in \mathcal{P}_{i,j}} x_p \leq T_{i,j} \quad \forall i,j \in [n] \\ &\qquad \sum_{p: e \in p} x_p \leq \alpha_l \quad \forall\ e \in M_l,\ \forall\ l \in [k] \\ &\quad \alpha_1 + \ldots + \alpha_k \leq W - k\delta \\ &x_p \geq 0\ \forall p \in \mathcal{P}, \alpha_i \geq 0\ \forall i \in [k], \end{aligned} \tag{7}$$

where $\mathcal{P}_{i,j}$ denotes the set of paths starting from a node $i$ in $G$ and terminating at node $j$, $\mathcal{P}$ denotes the set of all paths $\cup_{i,j \in [n]} \mathcal{P}_{i,j}$, $x_p$ is the flow along path $p$ and $\alpha_i$ is the duration of the $i$-th matching. We note that though the present form of the LP can contain an exponential number of variables, equivalent edge-based formulations exist with only a polynomial number of variables and constraints. As such one could use a generic LP solver to obtain the desired schedule. A closely related problem is the classical *multi-commodity flow* problem [1,28,7] that was predominantly solved using linear programming based approaches. However, despite many years of research in this direction the proposed algorithms were often too slow even for moderate sized instances [36]. Since then there has been a renewed effort in providing efficient *approximate* solutions to the multicommodity flow problem [23,4]. The algorithms we present are also a step in this direction, favoring efficiency over exactness of the solution. To do this, we consider the following two settings: (1) where the durations $\alpha_i$ of the matchings are fixed – this case allows for a natural, simple, fast and nearly-optimal algorithm by exploiting the submodularity and is disussed in the following Section 4.2 and (2) under arbitrary durations by using an approach similar to the primal-dual method in LP – this is discussed subsequently in Section 4.3.

### 4.2 Algorithm: Eclipse++

Consider the graph $G$ discussed above under a fixed matching, duration sequence $(M_1, \alpha_1), \ldots, (M_k, \alpha_k)$. Let $R(e)$ denote the capacity of edge $e \in G$. In this setting, the capacity constraints on the end-to-end paths are the sole constraints to the submodular optimization problem – we consider subsets $\{(\beta_1, p_1), \ldots, (\beta_m, p_m)\}$ that obey

$$\sum_{i=1}^{m} \beta_i \mathbf{1}_{\{e \in p_i\}} \leq R(e) \quad \forall e \in G. \tag{8}$$

Notice that the constraints above have a linear form, and there are a total of $kn$ such constraints (one for each edge). Hence, motivated by [5], which presents a fast and efficient multiplicative weights algorithm for submodular

**Algorithm 4:** Eclipse++ : greedy indirect routing algorithm

**Input** : Traffic demand $T$, switch configurations with residue capacities $R_1, \ldots, R_k$, update factor $\lambda$
**Output**: Sequence of paths $p_1, \ldots, p_m$ and corresponding weights $\beta_1, \ldots, \beta_m$

sch $\leftarrow \{\}$ ; `// schedule`
$T_{\text{rem}} \leftarrow T$ ; `// traffic remaining`
$w_e \leftarrow 1/R(e)$ for all $e \in E$;
$m \leftarrow 1$;
**while** $\sum_{e \in E} R(e) w_e \leq \lambda$ *and* $\|T_{\text{rem}}\|_1 > 0$ **do**
  $(\beta_m, p_m) \leftarrow \text{argmax}_{p \in \mathcal{P}, \beta \in \mathbb{Z}} \frac{\min(\beta, T_{\text{rem}}(p(0), p(k+1)))}{\sum_{e \in E} \beta \mathbf{1}_{\{e \in p\}} w_e}$ ;
  sch $\leftarrow$ sch $\cup \{(\beta_m, p_m)\}$ ;
  $T_{\text{rem}}(p_m(0), p_m(k+1)) \leftarrow T_{\text{rem}}(p_m(0), p_m(k+1)) - \beta$ ;
  $w_e \leftarrow w_e \lambda^{\beta_m \mathbf{1}_{\{e \in p\}}/R(e)} \quad \forall e \in G$ ;
  $m \leftarrow m + 1$;
**end**
**if** $\sum_{i=1}^{m-1} \beta_i \mathbf{1}_{\{e \in p_i\}} \leq R(e) \quad \forall e \in E$ **then**
  return sch
**else**
  return sch$\backslash(\beta_{m-1}, p_{m-1})$
**end**

maximization under linear constraints, we propose Eclipse++ in Algorithm 4. The structure of Eclipse++ is similar in spirit to Eclipse (Algorithm 2) in the sense that (a) the algorithm proceeds in rounds, where one new path is added to the schedule in each round and (b) we select a path that offers the greatest utility per unit of cost incurred. However, unlike Algorithm 2 where there was only one linear constraint, we have multiple linear constraints now. This is addressed by assigning weights to the constraints and considering a *linear combination* of the costs as the true cost in each round. In the following, we describe the salient features of Eclipse++.

Recall the capacity constraints in Equation (8) for each edge $e \in G$; let $w_e$ denote the weight assigned to the constraint involving edge $e$. We set $w_e = 1/R(e)$ for all $e$ initially, i.e., edges with a large capacity are assigned a small weight and vice-versa. We can now have another graph $G_w$ (with same topology as $G$) whose edges are weighted by $w_e$. Now, for any path $p$ the "effective cost" of the path per packet is simply the total cost of $p$ in $G_w$. Thus for the path $(\beta, p)$ carrying $\beta$ packets, the effective cost is given by $\sum_{e \in E} \beta w_e \mathbf{1}_{e \in p}$. On the other hand, the benefit we get due to adding path $(\beta, p)$ is given by $\min(\beta, T(p(0), p(k+1)))$ where $p(0)$ and $p(k+1)$ stand for the starting and terminating nodes along path $p$. Thus, the ratio $\frac{\min(\beta, T(p(0), p(k+1)))}{\sum_{e \in E} \beta w_e \mathbf{1}_{e \in p}}$ denotes the benefit of path $p$ per unit cost incurred. In Algorithm 4 we select $p$ such that the utility per unit cost is maximized.

Now, once we have selected a weighted path $(\beta_1, p_1)$ in the first round, we update the weights $w_e$ on the edges. This is done as $w_e \leftarrow w_e \lambda^{\beta_1/R(e)}$ for each edge $e \in p$, where $\lambda$ is an input parameter. For the remaining edges the weights remain unchanged. Thus repeating the above iteratively until the `while` loop condition $\sum_{e \in E} R(e) w_e \leq \lambda$ becomes invalid, we get a schedule that is the

output of the algorithm. It can also be shown that if the schedule returned `sch` violates any of the constraints (Equation (8)) then it must have happened at the very last iteration and hence we return a schedule with the last added path removed from it. It only remains to show how the maximizer of

$$\frac{\min(\beta, T_{\text{rem}}(p(0, p(k+1))}{\sum_{e\in E} \beta \mathbf{1}_{\{e\in p\}} w_e} \tag{9}$$

is computed efficiently in each round (first line inside the `while` loop). Consider the set of shortest paths in $G_w$ (smallest $w_e$-weighted path) from vertices in $V_0$ to vertices in $V_k$. Let $p^*$ denote the shortest among them. Then by setting $\beta^* \leftarrow T_{\text{rem}}(p^*(0), p^*(k+1))$ we claim that Equation (9) is maximized. This is because,

$$\begin{aligned}\frac{\min(\beta, T_{\text{rem}}(p(0, p(k+1))}{\sum_{e\in E} \beta \mathbf{1}_{\{e\in p\}} w_e} &\leq \frac{\beta}{\sum_{e\in E} \beta \mathbf{1}_{\{e\in p\}} w_e} \\ &\leq \frac{1}{\min \sum_{e\in E} \mathbf{1}_{\{e\in p\}} w_e}.\end{aligned}$$

If $T_{\text{rem}}(p^*(0), p^*(k+1)) = 0$ we proceed to the second smallest shortest path and so on. This allows a very efficient implementation of the internal maximization step.

**Approximation Guarantee:** We show, as in the direct-routing scenario, that Eclipse++ has a constant factor approximation guarantee. Specifically, for a fixed instance of the traffic matrix, let `OPT` and `ALG4` denote the value of the objectives achieved by the optimal algorithm (under fixed matchings, durations) and Eclipse++ respectively. Let $\eta := \max_{i,j\in[n],e\in E} T(i,j)/R(e)$. Then one can show that $\texttt{ALG4} = \Omega(1/(nk)^\eta)\texttt{OPT}$ for $\lambda = e^{1/\eta} nk$; the proof is analogous to the direct-routing case and follows [5, Theorem 1.1]. Further, if $\eta = O(\epsilon^2/\log(nk))$ for some fixed $\epsilon > 0$ then we get a approximation ratio of $(1-\epsilon)(1-1/e)$ by letting $\lambda = e^{\epsilon/(4\eta)}$ (using [5, Theorem 1.2]). An interesting regime where this occurs is when the traffic matrices are dense with small skew. For example, we get a constant factor approximation if the sparsity of the traffic matrix grows at least logarithmically fast. This is in stark contrast to direct routing, where sparse matrices generally perform better.

**Complexity:** The proposed algorithm is simple and fast. In this subsection, we explicitly enumerate the time complexity of the full algorithm and show that the complexity is at most cubic in $n$ and nearly linear in $k$. Let $W$ denote a bound on the total incoming or outgoing traffic for a node. In each iteration of the `while` loop at least one packet is sent. Therefore there are at most $W$ iterations of the `while` loop. Now, in each iteration finding the shortest paths between nodes in $V_0$ to nodes in $V_k$ takes $kn^2(\log k + \log n)$ operations using Dijkstra's algorithm [21]. Sorting the computed distances takes $kn^2(\log k + \log n)^2$ time and at most $n^2$ more operations to find a pair $i, j$ such that $T_{\text{rem}}(i,j) > 0$. Finally the weights update step takes $kn$ time. Therefore overall it takes $O(kn^2(\log k + \log n)^2)$ time per iteration. Hence the time complexity of the complete algorithm is $O(Wkn^3(\log k + \log n)^2)$.

### 4.3 Algorithm: EclipseX

While Eclipse++ presents a fast algorithm to compute the multi-hop routing policy, its performance to a large extent depends on the choice of the matching sequence. Yet it is apriori not clear how to choose these matchings and durations. In this section, we provide a fast approximation heuristic to jointly compute the optimal (i) durations of the matchings and (ii) multi-hop routing, under a *fixed* matching sequence. Such an heuristic can be helpful in the search for an optimal matching sequence. Following the same model as before, let $M_1, M_2, \ldots, M_k$ denote a sequence of $k$ matchings. Then using these matchings, our goal is to maximize the total amount of traffic sent. To do this, let us consider a time-layered directed graph $G$, as before, that is constructed by cascading the $k$ matchings $M_1, \ldots, M_k$. Let $E = \cup_{i=1}^{k} M_i$ denote the set of all matching edges in $G$.[4] In Equation (7) we have seen a linear programming relaxation for maximing our required sum-throughput objective. Such an LP has a form similar to multicommodity flow except that the matching durations also have to be decided. This additional constraint unfortunately violates the packing structure of the constraints under which case techniques from fractional packing problems could have been used. Hence as an alternative approach, we consider the dual of the path-flow LP as shown in the box below,

$$\begin{aligned} \text{minimize} \quad & (W - k\delta)\beta + \sum_{i,j \in [n]} T(i,j) z_{i,j} \\ \text{s.t.} \quad & z_{i,j} + \sum_{e : e \in p \cap E} y_e \geq 1 \quad \forall p \in \mathcal{P}_{i,j}, \forall i,j \in [n] \\ & \beta - \sum_{e \in M_l} y_e \geq 0 \quad \forall l \in [k] \\ & z_{i,j} \geq 0 \ \forall i,j \in [n], y_e \geq 0 \ \forall e \in E, \beta \geq 0. \end{aligned} \tag{10}$$

Here the $y(e)$ represent weights on the edges $e \in E$, $z_{i,j}$ is a weight on each $(i,j)$ source-destination pair and $\beta$ is a variable bounding the total weight of any matching. Further the first set of constraints, which impose a minimum path-weight condition, can be verified quickly using the all-pairs shortest path algorithm (e.g. Dijkstra's). The second set of constraints can also be verified quickly as this involves simply checking the weight of each of the $k$ matchings. As such there exists an efficient polynomial time separation oracle that, given a solution $(y(e), z_{i,j}, \beta)$, is able to either (i) decide that the solution satisfies all the constraints or (ii) output a constraint that is violated by the solution.[5] On the other hand, the state-of-the-art technique in maximum multicommodity flow [23,20] involves a primal-dual based algorithm which depends heavily on

[4] Note that in addition to the matching edges $E$, $G$ also contains edges connecting nodes representing the same server across multiple matching rounds.

[5] This also implies the LP can be solved in polynomial time using the Ellipsoidal method [27].

**Algorithm 5:** EclipseX: a template for jointly computing durations and indirect routing under a fixed input sequence of matchings.

**Input** : Traffic demand $T$, reconfiguration delay $\delta$ and scheduling window size $W$, sequence of matchings $M_1, \ldots, M_k$, parameters $\epsilon, \lambda$
**Output**: Sequence of durations $\alpha_1, \ldots, \alpha_k$; sequence of paths and their weights $(p_1, \beta_1), \ldots, (p_m, \beta_m)$
Initialize primal: $x_p \leftarrow 0\ \forall p \in \mathcal{P}, \alpha_l \leftarrow \delta\ \forall l \in [k]$;
Initialize dual: $y(e) \leftarrow \delta/W\ \forall e \in E, z_{i,j} \leftarrow 1/(T(i,j))\ \forall i,j \in [n], \beta \leftarrow n\delta$;
$p(i,j) \leftarrow$ shortest $y$-path from node $i$ to node $j$ in $G$;
$(i^*, j^*) \leftarrow \text{argmin}_{i,j \in [n]}(y(p(i,j)) + z_{i,j})$;
$l^* \leftarrow \text{argmax}_{l \in [k]}(y(M_l))$;
**while** $y(p(i^*, j^*)) < 1$ *or* $y(M_{l^*}) > \beta$ **do**
  **if** $y(p(i^*, j^*)) < 1$ **then**
    $\beta(p(i^*, j^*)) \leftarrow \beta(p(i^*, j^*)) + \epsilon$ ; `// augment flow along` $p(i^*, j^*)$
    $y(e) \leftarrow y(e)(1 + \lambda\epsilon/\alpha(e))\ \forall e \in p(i^*, j^*) \cap E$ ; `// increase edge weights`
    $z_{i^*,j^*} \leftarrow z_{i^*,j^*}(1 + \epsilon/T(i^*, j^*))$ ; `// increase` $z_{i^*,j^*}$
  **else**
    $\alpha_{l^*} \leftarrow \alpha_{l^*}(1 + \lambda)$ ; `// augment duration` $\alpha_{l^*}$ `of` $M_{l^*}$
    $\beta \leftarrow \beta(1 + \lambda)$ ; `// increase` $\beta$
  **end**
  compute the new $i^*, j^*, l^*$ as before;
**end**

the existence of an efficient separation oracle in the dual program. Hence motivated by these observations, we propose a primal-dual based heuristic that is able to approximately solve the LP and is fast. Our proposed solution, which we call EclipseX, is also a greedy algorithm proceeding in rounds wherein at each round either (i) the flow is augmented along a path or (ii) the duration of a matching is increased.Such an approach is in tune with the general principle of primal-dual methods wherein at each round either the objective value is increased or the complementary slackness conditions are improved. We conjecture that our proposed heuristic has a constant factor approximation guarantee to the optimal.

**Variable capacity multicommodity flow:** The specific instance of the problem discussed in EclipseX also leads to a more general problem of multicommodity flow with variable edge capacities. Consider a directed graph $G = (V, E)$ with $k$ source-destination pairs $(s_i, t_i), i \in [k]$. Let the $i$-th com-

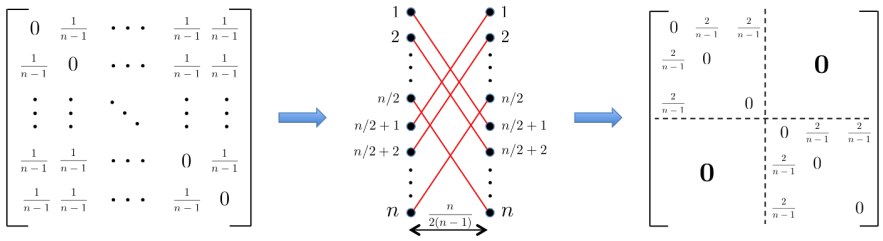


**Fig. 4** An example showing a matching to reduce the sparsity of the traffic matrix by half under uniform load.

modity have a total traffic of $d_i$ for each $i$. Then the LP

$$\begin{aligned} \text{maximize} \quad & \sum_{p \in \mathcal{P}} x_p \\ \text{s.t.} \quad & \sum_{p \in \mathcal{P}_i} x_p \leq d_i \quad \forall i \in [k] \\ & \sum_{p: e \in p} x_p \leq c_e \quad \forall\ e \in E \\ & A\mathbf{c} \leq \mathbf{b} \\ & \mathbf{x} \geq 0, \mathbf{c} \geq 0, \end{aligned} \tag{11}$$

where $\mathcal{P}_i$ is the set of all paths from node $s_i$ to node $t_i$ and $\mathcal{P} = \cup_{i \in [k]} \mathcal{P}_i$, represents a maximum multicommodity flow problem with linear cost constraints on the edge capacities. Such a problem is of canonical interest both theoretically from an approximation algorithms point-of-view and practically for network design and capacity planning (e.g. of core data center networks). While special cases of this problem, such as flow assignment with capacity expansion [41,19,9], have been studied the general problem remains largely open. We defer analysis and experimentations on EclipseX and the general multicommodity flow problem as important topics for future research.

### 4.4 Algorithm: Eclipse#

So far we have discussed heuristics (Eclipse++ and EclipseX) that operate on a fixed input sequence of matchings – such as that computed by Eclipse. While in many common cases this strategy seems to suffice, in general the performance can vary dramatically depending on the choice of matchings. Hence it becomes imperative to have an algorithm that can select matchings carefully such that the multi-hop throughput is optimized. Unfortunately a polynomial time solution to this general problem appears to be hard.[6] One

[6] Despite fast algorithms to compute the schedule under fixed matchings, it is not clear how to perform the search over the space of matching sequences efficiently.

could also use the submodularity in the objective, but as mentioned before, the nonlinearity in the constraints make this approach challenging as well.

In this section, we propose an alternative heuristic (called Eclipse#) that is able to outperform both Eclipse and Eclipse++ under certain key regimes (Section 5.2). Owing to the difficulty of the problem, we have motivated Eclipse# by intuition and empirical observations rather than mathematical structures. The main differentiating factors here are that (i) the number of matchings is restricted to be small ($O(\log n)$) and (ii) the matchings are chosen randomly to provide good reachability between source-destination pairs. While the reduced number of matchings minimize capacity loss due to reconfiguration delay, it is also well known that random matchings have good expansion properties [47, 30]. These properties are in contrast to the schedule computed by Eclipse which could contain a large number of matchings. To illustrate this difference, let us consider the following example.

**Example:** Let $W = \log_2 n$ and $T$ be the uniform traffic matrix in which all the off-diagonal entries are $1/(n-1)$ (the diagonal entries are 0). For simplicity, let us assume than $n$ is an integer power of 2. This example represents an extreme case in which the load is only a $1/\log n$ fraction of the capacity (e.g. 10% load for a network with 1000 servers). Since indirect routing inherently entails multiple transmissions of the same flow across time, we also assume an enhanced network capacity (or a reduced traffic load) in order to feasibly schedule traffic demand matrices.[7] Now, scheduling this demand matrix $T$ using direct routing we would be able to send at most

$$\frac{W}{(\delta + \frac{1}{n-1})} \frac{n}{n-1} \leq \frac{2W}{\delta} = O(\log n),$$

a logarithmic amount of traffic. Indirect routing performed using the matchings returned by Eclipse would also give a similar performance. On the other hand, by using a different sequence of matchings we can essentially transmit all of the traffic in $\log_2 n$ time-units. To see this, consider the first round of matching of duration $1/2$ where the $i$-th node connects to the $((i + n/2) \mod n)$-th node. In this round, let nodes $1, \ldots, n/2$ transmit all of their traffic destined to nodes $n/2+1, \ldots, n$ through their respective outgoing edges. Similarly, let nodes $n/2+1, \ldots, n$ transmit all of their traffic destined to nodes $1, \ldots, n/2$ through their edge. Then at the end of this round, the resultant traffic matrix has only two blocks (along the diagonal) each with $n/2$ non-zero entries of value $2/(n-1)$. Thus we have reduced the initial sparsity of $n-1$ by a factor of half (see Figure 4). Now, each of the two blocks has a structure similar to the initial traffic matrix. Hence by repeating the same process as above we can reduce the sparsity by another half factor. Thus repeating $k$ times, we end up with a traffic matrix whose sparsity is roughly $n/2^k$. For $k = \log n$ we essentially have a matrix with constant sparsity. At this point, we can use direct routing to schedule the traffic as we have done in Eclipse.

[7] This requirement can be accommodated in practice since networks are often severely over-provisioned.

**Algorithm 6:** Eclipse# : general case greedy indirect routing algorithm

**Input** : Traffic demand $T$, reconfiguration delay $\delta$ and scheduling window size $W$
**Output**: Sequence of matchings and their durations $(M_1, \alpha_1), \ldots, (M_k, \alpha_k)$;
sequence of paths and their weights $(p_1, \beta_1), \ldots, (p_m, \beta_m)$
$d \leftarrow$ maximum number of non-zero entries in any row or column of $T$;
$k \leftarrow 2\lceil \log_2 d \rceil$ ;
$D \leftarrow T$;
**for** $i = 1$ *to* $k$ **do**
  $M_i \leftarrow$ random matching on support of $D$;
  $\alpha_i \leftarrow (W - k\delta)/k$;
  $D(i,j) \leftarrow 0 \ \ \forall (i,j) \in M_i$;
**end**
Run Eclipse++ on the schedule $(M_1, \alpha_1), \ldots, (M_k, \alpha_k)$ and traffic matrix $T$;
Return $(p_1, \beta_1), \ldots, (p_m, \beta_m)$ as computed by Eclipse++;

Thus using a right sequence of matchings can offer significant gains under multi-hop routing. In particular, for settings where the reconfiguration delay is large, the fixed sequence of matchings schedule should contain only a small number of matchings. We have enforced this by restricting the number of matchings to $2\log_2 d$ where $d$ is the sparsity of the input (intuitively, we need roughly $\log_2 d$ matchings to *sparsify* the traffic matrix; then on the sparse matrix we need an additional constant number of matchings to send the traffic). For simplicity, we have also set the durations of each of the matchings to be the same. This seems to be a reasonable choice for traffic matrices with a small skew (Section 5.2). Finally, to ensure connectivity between different pairs of nodes, a matching is chosen randomly on the support of the demand matrix at each round.

### 4.5 Discussion

In a schedule with a small number of matchings, the packets inherently have to take longer paths (more hops) and thus consume more capacity to reach their destinations. On the other hand, having a large number of matchings introduces capacity wastage in the form of reconfiguration delays. Thus, balancing this trade-off between the number of matchings and the average number of hops packets take to reach their destinations is a key challenge to finding a good matching sequence. The algorithms Eclipse++ and Eclipse# seem to operate well at regimes where the optimum lies near either extreme of this spectrum respectively (as we will also see from evaluations in Section 5.2). It would be interesting to characterize this trade-off and design a general algorithm that is able to discern and make these design choices by itself.

Throughout our discussion so far, we have adopted a model where the scheduling window is of a fixed duration $W$ and has at least one switching operation per window. From a throughput perspective, this inherently causes a rate loss of at least $\delta/W$ fraction of capacity. Practical optical circuit switches,

on the other hand, are often able to retain their previous switching state across adjacent time windows (i.e., without requiring a new configuration at the start of each window). This suggests a natural modification of our current algorithms where in the first round of each window we either (i) retain the last matching of the previous round or (ii) switch to a new matching. We believe such a modification will further improve performance; the precise changes and an evaluation under continuous traffic (such as Bernoulli arrivals) are left for future work.

## 5 Evaluation

In this section, we complement our analytical results with numerical simulations to explore the effectiveness of our algorithms and compare them to state-of-the-art techniques in the literature. We empirically evaluate both the direct routing algorithm (Eclipse : Algorithm 2) and the indirect routing algorithms (Eclipse++ : Algorithm 4, Eclipse# : Algorithm 6).

**Metric:** We consider the total fraction of traffic delivered via the circuit switch (sum-throughput) over the duration of a fixed scheduling window as the performance metric throughout this section. Evaluating our algorithms under continuous traffic arrival models remains an important future direction.

**Schemes compared:** Our experiments compare Eclipse against two existing algorithms for direct routing:
*(1) Solstice [40]:* This is the state-of-the-art hybrid circuit/packet scheduling algorithm for data centers. The key idea in Solstice is to choose matchings with 100% utilization. This is achieved by thresholding the demand matrix and selecting a perfect matching in each round. The algorithm presented in [40] tries to minimize the total duration of the window such that the entire traffic demand is covered. In this paper, we have considered a more general setting where the scheduling window $W$ is constrained. To compare against Solstice in this setting, we truncate its output once the total schedule duration exceeds $W$.
*(2) Truncated Birkhoff-von-Neumann (BvN) decomposition:* The second algorithm we compare against is the truncated BvN decomposition algorithm [11]. BvN decomposition refers to expressing a doubly stochastic matrix as a convex combination of permutation matrices and this decomposition procedure has been extensively used in the context of packet switch scheduling [40,34, 11]. However BvN decomposition is oblivious to reconfiguration delay and can produce a potentially large ($O(n^2)$) number of matchings. Indeed, in our simulations BvN performs poorly.

Indirect routing is relatively new and to the best of our knowledge our work is the first to consider use of indirect routing for centralized scheduling.[8] In our second set of simulations, we show that the benefits of indirect routing are

[8] Indirect routing in a distributed setting but without consideration of switch reconfiguration delay was studied in a recent work [10].

in addition to the ones obtained from switch configurations scheduling. To this end, we compare Eclipse with Eclipse++ to quantify the additional throughput obtained by performing indirect routing (Algorithm 4) on a schedule that has been (pre)computed using Eclipse. We also compare Eclipse# (Algorithm 6) with Eclipse++ to demonstrate the influence of the choice of matchings on performance.

**Traffic demands:** We consider two classes of inputs: (a) single-block inputs and (b) multi-block inputs (explained in Section 5.1). Intuitively, single-block inputs are matrices which consist of one $n \times n$ 'block' that is sparse and skewed, and are similar to the traffic demands evaluated in the Solstice paper [40]. Multi-block inputs, on the other hand, denote traffic matrices that are composed of many sub-matrices each with disparate properties such as sparsity and skew.

**Network size:** The number of ports is fixed in the range of 50–200. We find that the relative performances stayed numerically stable over this range as well as for increased number of ports.

### 5.1 Direct Routing

While maintaining the sum-throughput as the performance metric, we vary the various parameters of the system model to gauge the performance in different situations.

*Single-Block Inputs*

For a single-block input, our simulation setup consists of a network with 100 ports. The link rate of the circuit switch is normalized to 1, and the scheduling window length is also 1 ($W = 1$). We consider traffic inputs where the maximum traffic to or from any port is bounded by $W$. Further, we let the reconfiguration delay $\delta = W/100$. The traffic matrix is generated similar to [40] as follows. We assume 4 large flows and 12 small flows to each input or output port. The large flows are assumed to carry 70% of the link bandwidth, while the small flows deliver the remaining 30% of the traffic. To do this, we let each flow be represented by a random weighted permutation matrix, i.e., we have

$$T = \sum_{i=1}^{n_L} \frac{c_L}{n_L} P_i + \sum_{i'=1}^{n_S} \frac{c_S}{n_S} P_{i'} + N, \tag{12}$$

where $n_L(n_S)$ denotes the number of large (small) flows and $c_L(c_S)$ denotes the total percentage of traffic carried by the large (small) flows. In this case, we have $n_L = 4, n_S = 12$ and $c_L = 0.7, c_S = 0.3$. Further, we have added a small amount of noise $N$ — additive Gaussian noise with standard deviation equal to 0.3% of the link capacity — to the non-zero entries to introduce some perturbation. Each experiment below has been repeated 25 times.

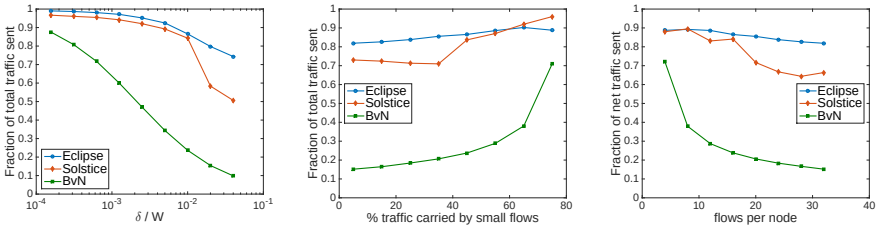



**Fig. 5** Performance comparison of Eclipse under single-block inputs.

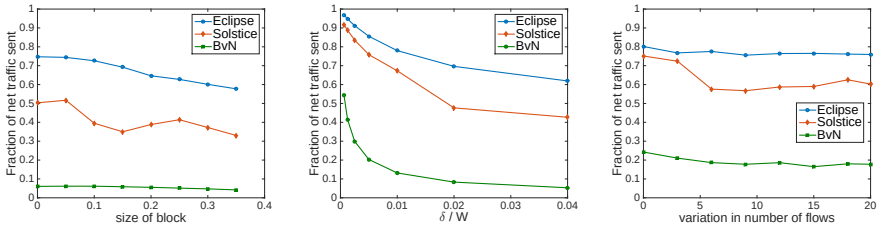
**Reconfiguration delay:** In Fig. 5(a) we plot sum-throughput while varying the reconfiguration delay from $W/3200$ to $4W/100$. Eclipse achieves a throughput of at least 90% for $\delta \leq W/100$. We observe Eclipse to be consistently better than Solstice although the difference is not pronounced until $\delta > W/100$. The BvN decomposition algorithm has a large throughput when the reconfiguration delay is small. As $\delta$ increases, its performance gradually worsens.
**Skew:** We control the skew by varying the ratio of the amount of traffic carried by small and large flows in the input traffic demand matrix ($c_L/c_S$ in Equation (12)). Fig. 5(b) captures the scenario where the percentage traffic carried by the small flows is varied from 5 to 75. We observe that Eclipse is very robust to skew variations and is able to consistently maintain a throughput of about 85%. Solstice has a slightly better performance at low skew (when small-flows carry $\sim 75\%$ of traffic); but overall, is dominated by Eclipse.
**Sparsity:** Finally, we tested the algorithms' dependence on sparsity and plotted the results in Fig. 5(c). The total number of flows is varied from 4 to 32, while fixing the ratio of the number of large to small flows at 1:3. As the input matrix becomes less sparse, the performance of algorithms degrade as expected. However, for Eclipse, the reduction in the throughput is never more than 10% over the range of sparsity parameters considered. Solstice, on the other hand, is affected much more severely by decreased sparsity.

*Multi-Block Inputs*

Next, we consider a more complex traffic model for a 200 node network with block diagonal inputs of the form

$$T = \begin{bmatrix} B_1 & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & B_m \end{bmatrix},$$

where each of the component blocks $B_1, B_2, \ldots, B_m$ can have its own sparsity (number of flows) and skew (fraction of traffic carried by large versus small flows) parameters. The different blocks model the traffic demands of different




(a) (b) (c)

**Fig. 6** Performance comparison of Eclipse under multi-block inputs.

tenants in a shared data center network such as a public cloud data center. To begin with, we consider inputs with two blocks $T = \begin{bmatrix} B_1 & \mathbf{0} \\ \mathbf{0} & B_2 \end{bmatrix}$ where $B_1$ is a $n_1 \times n_1$ matrix with 4 large flows (carrying 70% of the traffic) and 12 small flows (carrying 30% of the traffic) and $B_2$ is a $(200 - n_1) \times (200 - n_1)$ matrix with uniform entries (up to sampling noise).

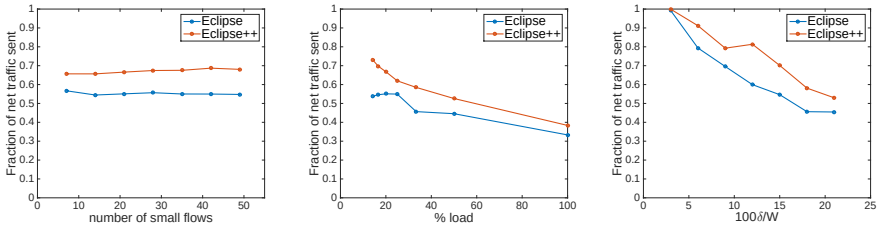
**Size of block:** Fig. 6(a) plots the throughput as the block size of $B_2$ is increased from 0 to 70. We observe a very pronounced difference in the performance of Eclipse and Solstice: Eclipse has roughly $1.5-2\times$ the performance of Solstice. These findings are in tune with the intuition discussed in Section 3.1 — the deteriorated performance of Solstice is due its insistence on perfect matchings in each round.

**Reconfiguration delay:** Fig. 6(b) plots throughput while varying the reconfiguration delay, for fixed size of $B_2$ to be $50 \times 50$. As expected, the throughput of Solstice and Eclipse both degrade as the reconfiguration delay $\delta$ increases. However, Eclipse throughput degrades at a much slower rate than Solstice. The gap between the two is particularly pronounced for $\delta/W \geq 0.02$, a numerical value that is well within range of practical system settings.

**Varying numbers of flow:** In the final experiment we consider block diagonal inputs with 8 blocks of size $25 \times 25$ each. Each block carries $10 + \lfloor \sigma * (U - 0.5)) \rfloor$ equi-valued flows where $U \sim$ unif $(0, 1)$ and $\sigma$ is a parameter that controls the variation in the number of flows. When increasing $\sigma$ from 0 to 20 we see from Fig.6(c) that Eclipse is more or less able to sustain its throughput at close to 80%; whereas Solstice is significantly affected by the variation.

## 5.2 Indirect Routing

In this section, we consider a 50 node network with traffic matrices having varying number of large and small flows as before. We compare the performance of the direct routing algorithm and the indirect routing algorithm that is run on the schedule computed by Eclipse. To understand the benefits of indirect routing, we focus on the regime where the reconfiguration delay $\delta/W$ is relatively large and the scheduling window $W$ is relatively long compared




(a) (b) (c)

**Fig. 7** Performance of Eclipse++ and Eclipse. Here Eclipse++ uses the schedule computed by Eclipse.

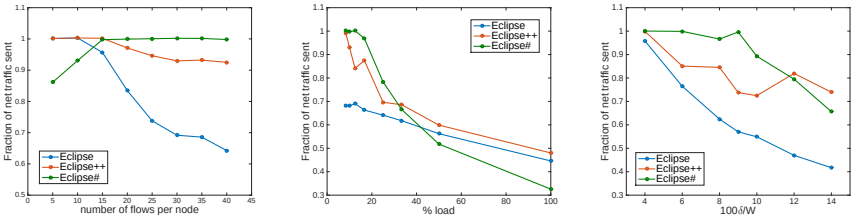
to the traffic demand. This regime corresponds to realistic scenarios where the circuit switch is not fully utilized (Real data center networks often have low to moderate utilization; e.g, 10–50% [8]), but the reconfiguration delay is large. In this setting of relatively large $\delta/W$, switch schedules are forced to have only a small number of matchings, and indirect routing is critical to support (non-sparse) demand matrices. The following experiments numerically demonstrate the added gains of indirect routing.

**Sparsity:** Fig. 7(a) considers a demand with 5 large flows and number of small flows varying from 7 to 49. The large and the small flows each carry 50% of the traffic. We let $\delta = 16W/100$ and consider a load of 20% (i.e., $W = 5$, and traffic load at each port is 1). We observe that the performance of the Eclipse++ is roughly 10% better than Eclipse.

**Load:** As the network load increases (Fig. 7(b)), we see that indirect routing becomes less effective. This is because at high load, the circuits do not have much spare capacity to support indirect traffic. However, at low to moderate levels of load, indirect routing provides a notable throughput gain over direct routing. For example, we see close to 20% improvement with Eclipse++ over Eclipse at 15% load.

**Reconfiguration delay:** Finally, we observe the effect of $\delta/W$ on throughput by varying $\delta$ from $3W/100$ to $21W/100$. At smaller values of reconfiguration delay $\delta$ both Eclipse and Eclipse++ are able to achieve near 100% throughput. With increasing $\delta$ both algorithms degrade with Eclipse++ providing an additional gain of roughly 20% over Eclipse.

Thus having the capability for indirection can offer a significantly improved performance over direct routing schemes. However in our present framework of proposed solutions we have largely left open the choice of the matching sequence. In our next set of experiments we show that a careful choice of matchings can further benefit the performance of indirect routing. For this we compare the three schemes of Eclipse, Eclipse++ and Eclipse# in a large $W$ regime and where the traffic matrices are dense. We continue to consider a 50 node network as before.




(a) (b) (c)

**Fig. 8** Performance of Eclipse, Eclipse++ and Eclipse#. Eclipse++ uses the schedule computed by Eclipse while Eclipse# uses random matchings.

**Sparsity:** In Fig. 8(a) we consider a demand with the number of flows increasing from 5 to 40. Within each instance, the flows share the total load (approximately) evenly. We let $\delta = W/15$ and a total load of 10% (i.e., $W = 10$ and the aggregate per-port traffic is 1). As previously, we observe that Eclipse++ is consistently better than Eclipse. However as the traffic matrix becomes denser, we observe that Eclipse# is able to achieve 100% throughput (a roughly 8% increase over Eclipse++).
**Load:** Next we evaluate the performace under traffic loads varying from 10 to 100%. If the load is smaller than 30%, we see that Eclipse# has a significantly better performance than Eclipse. However as the load increases Eclipse# degrades while Eclipse and Eclipse++ sustain their throughput. This is because, the (relatively) small number of matchings in Eclipse# minimize capacity loss due to reconfiguration delay while at the same time the random matchings provide sufficient connectivity between the source and destination port-pairs.
**Reconfiguration delay:** In this last experiment, we vary the reconfiguration delay $\delta$ in the range $4W/100$ to $10W/100$. We observe that when the reconfiguration delay is small, all three algorithms have a near-optimal performance. However as $\delta$ increases, Eclipse and Eclipse++ suffer a performance drop – again owing to an excessive number of matchings – while Eclipse# with its minimalistic matching sequence is able to sustain its performance.

## 6 Final Remarks

We have studied scheduling in hybrid switch architectures with reconfiguration delays in the circuit switch, by taking a fundamental and first-principles approach. The connections to submodular optimization theory allows us to design simple and fast scheduling algorithms and show that they are near optimal — these results hold in the direct routing scenario and indirect routing provided switch configurations are calculated separately. However, we note that the proposed algorithms are not throughput optimal. This is because the matchings selected in Eclipse are chosen by first thresholding the traffic matrix and then choosing a maximum weight matching. Such a scheme is analogous to the maximum size matching studied in [35, 43] and can be shown to achieve

strictly less than 100% throughput. Nevertheless from a practical point-of-view, where traffic loads are often only a fraction of the network capacity, the algorithm is still interesting and could offer potentially a near-optimal delay performance. A systematic study comparing the delay properties of Eclipse to the state-of-the-art is left for future work.

The problem of jointly deciding the switch configurations *and* indirect routing policies also remains open. While submodular function optimization with nonlinear constraints is in general intractable, the specific constraints discussed in Section 4.1 perhaps have enough structure that they can be handled in a principled way.

In between the scheduling windows of $W$ time units, traffic builds up at the ToR ports. This dynamic traffic buildup is known locally to each of the ToR ports and perhaps this local knowledge can be used to pick appropriate indirect routing policies in a distributed, dynamic fashion. Such a study of indirect routing policies is initiated in a recent work [10], but this work omitted the switching reconfiguration delays. A joint study of distributed dynamic traffic scheduling in conjunction with a static schedule of switch configurations that account for reconfiguration delays is also an interesting direction of future research.

**Acknowledgements** The authors would like to thank Prof. Chandra Chekuri, Prof. George Porter and Prof. R. Srikant for the many helpful discussions. This work was partially supported by NSF grants CCF-1409106, NeTS-1718270 and Army grant W911NF-14-1-0220.

## References

1. Ahuja, R., Magnanti, T., Orlin, J.: Network flows: theory, algorithms, and applications. Prentice Hall (1993). URL https://books.google.com/books?id=WnZRAAAAMAAJ
2. Al-Fares, M., Radhakrishnan, S., Raghavan, B., Huang, N., Vahdat, A.: Hedera: Dynamic flow scheduling for data center networks. In: NSDI, vol. 10, pp. 19–19 (2010)
3. Alizadeh, M., Greenberg, A., Maltz, D.A., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., Sridharan, M.: Data center tcp (dctcp). SIGCOMM (2011)
4. Arora, S., Hazan, E., Kale, S.: The multiplicative weights update method: a meta-algorithm and applications. Theory of Computing **8**(1), 121–164 (2012)
5. Azar, Y., Gamzu, I.: Efficient submodular function maximization under linear packing constraints. In: Automata, Languages, and Programming, pp. 38–50. Springer (2012)
6. Barman, S.: Approximating nash equilibria and dense bipartite subgraphs via an approximate version of caratheodory's theorem. In: Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, pp. 361–369. ACM (2015)
7. Barnhart, C., Sheffi, Y.: A network-based primal-dual heuristic for the solution of multicommodity network flow problems. Transportation Science **27**(2), 102–117 (1993)
8. Benson, T., Akella, A., Maltz, D.A.: Network traffic characteristics of data centers in the wild. In: SIGCOMM (2010)
9. Bienstock, D., Chopra, S., Günlük, O., Tsai, C.Y.: Minimum cost capacity installation for multicommodity network flows. Mathematical programming **81**(2), 177–199 (1998)
10. Cao, Z., Kodialam, M., Lakshman, T.: Joint static and dynamic traffic scheduling in data center networks. In: INFOCOM (2014)
11. Chang, C.S., Chen, W.J., Huang, H.Y.: Birkhoff-von neumann input buffered crossbar switches. In: INFOCOM (2000)

12. Chang, C.S., Lee, D.S., Jou, Y.S.: Load balanced birkhoff-von neumann switches. In: High Performance Switching and Routing, 2001 IEEE Workshop on, pp. 276–280. IEEE (2001)
13. Dasylva, A., Srikant, R.: Optimal wdm schedules for optical star networks. IEEE/ACM Transactions on Networking (TON) **7**(3), 446–456 (1999)
14. Duan, R., Pettie, S.: Linear-time approximation for maximum weight matching. J. ACM **61**(1), 1:1–1:23 (2014). DOI 10.1145/2529989. URL http://doi.acm.org/10.1145/2529989
15. Duan, R., Su, H.H.: A scaling algorithm for maximum weight matching in bipartite graphs. In: SODA (2012)
16. Farrington, N.: Optics in data center network architecture. Ph.D. thesis, Citeseer (2012)
17. Farrington, N., Porter, G., Radhakrishnan, S., Bazzaz, H.H., Subramanya, V., Fainman, Y., Papen, G., Vahdat, A.: Helios: a hybrid electrical/optical switch architecture for modular data centers. SIGCOMM (2011)
18. Felzenszwalb, P.F., Zabih, R.: Dynamic programming and graph algorithms in computer vision. Pattern Analysis and Machine Intelligence, IEEE Transactions on **33**(4), 721–740 (2011)
19. Ferreira, R.P.M., Luna, H.P.L., Mahey, P., Souza, M.C.d.: Global optimization of capacity expansion and flow assignment in multicommodity networks. Pesquisa Operacional **33**(2), 217–234 (2013)
20. Fleischer, L.K.: Approximating fractional multicommodity flow independent of the number of commodities. SIAM Journal on Discrete Mathematics **13**(4), 505–520 (2000)
21. Fredman, M.L., Tarjan, R.E.: Fibonacci heaps and their uses in improved network optimization algorithms. Journal of the ACM (JACM) **34**(3), 596–615 (1987)
22. Fu, S., Wu, B., Jiang, X., Pattavina, A., Zhang, L., Xu, S.: Cost and delay tradeoff in three-stage switch architecture for data center networks. In: HPSR (2013)
23. Garg, N., Koenemann, J.: Faster and simpler algorithms for multicommodity flow and other fractional packing problems. SIAM Journal on Computing **37**(2), 630–652 (2007)
24. Giaccone, P., Prabhakar, B., Shah, D.: Randomized scheduling algorithms for high-aggregate bandwidth switches. Selected Areas in Communications, IEEE Journal on **21**(4), 546–559 (2003)
25. Gopal, I.S., Wong, C.K.: Minimizing the number of switchings in an ss/tdma system. Communications, IEEE Transactions on **33**(6), 497–501 (1985)
26. Greenberg, A., Lahiri, P., Maltz, D.A., Patel, P., Sengupta, S.: Towards a next generation data center architecture: scalability and commoditization. In: Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow, pp. 57–62. ACM (2008)
27. Grötschel, M., Lovász, L., Schrijver, A.: The ellipsoid method and its consequences in combinatorial optimization. Combinatorica **1**(2), 169–197 (1981)
28. Grötschel, M., Lovász, L., Schrijver, A.: Geometric algorithms and combinatorial optimization. Algorithms and combinatorics. Springer-Verlag (1993). URL https://books.google.com/books?id=agLvAAAAMAAJ
29. Hamedazimi, N., Qazi, Z., Gupta, H., Sekar, V., Das, S.R., Longtin, J.P., Shah, H., Tanwer, A.: Firefly: a reconfigurable wireless data center fabric using free-space optics. In: SIGCOMM (2014)
30. Hoory, S., Linial, N., Wigderson, A.: Expander graphs and their applications. Bulletin of the American Mathematical Society **43**(4), 439–561 (2006)
31. Inukai, T.: An efficient ss/tdma time slot assignment algorithm. Communications, IEEE Transactions on **27**(10), 1449–1455 (1979)
32. Kandula, S., Padhye, J., Bahl, P.: Flyways to de-congest data center networks (2009)
33. Keslassy, I., Chang, C.S., McKeown, N., Lee, D.S.: Optimal load-balancing. In: INFOCOM (2005)
34. Keslassy, I., Kodialam, M., Lakshman, T., Stiliadis, D.: On guaranteed smooth scheduling for input-queued switches. In: INFOCOM (2003)
35. Keslassy, I., Zhang-Shen, R., McKeown, N.: Maximum size matching is unstable for any packet switch. IEEE Communications Letters **7**(10), 496–498 (2003)
36. Leighton, T., Makedon, F., Plotkin, S., Stein, C., Tardos, É., Tragoudas, S.: Fast approximation algorithms for multicommodity flow problems. Journal of Computer and System Sciences **50**(2), 228–243 (1995)

37. Li, X., Hamdi, M.: On scheduling optical packet switches with reconfiguration delay. Selected Areas in Communications, IEEE Journal on **21**(7), 1156–1164 (2003)
38. Li, Y., Panwar, S., Chao, H.J.: Frame-based matching algorithms for optical switches. In: High Performance Switching and Routing, 2003, HPSR. Workshop on, pp. 97–102. IEEE (2003)
39. Liu, H., Lu, F., Forencich, A., Kapoor, R., Tewari, M., Voelker, G.M., Papen, G., Snoeren, A.C., Porter, G.: Circuit switching under the radar with reactor. In: NSDI (2014)
40. Liu, H., Mukerjee, M.K., Li, C., Feltman, N., Papen, G., Savage, S., Seshan, S., Voelker, G.M., Andersen, D.G., Kaminsky, M., Porter, G., Snoeren, A.C.: Scheduling techniques for hybrid circuit/packet networks. In: ACM CoNEXT (2015)
41. Mahey, P., Benchakroun, A., Boyer, F.: Capacity and flow assignment of data networks by generalized benders decomposition. Journal of Global Optimization **20**(2), 169–189 (2001)
42. McKeown, N.: The islip scheduling algorithm for input-queued switches. Networking, IEEE/ACM Transactions on **7**(2), 188–201 (1999)
43. McKeown, N., Mekkittikul, A., Anantharam, V., Walrand, J.: Achieving 100% throughput in an input-queued switch. Communications, IEEE Transactions on **47**(8), 1260–1267 (1999)
44. Mekkittikul, A., McKeown, N.: A practical scheduling algorithm to achieve 100% throughput in input-queued switches. In: INFOCOM (1998)
45. Mirrokni, V., Leme, R.P., Vladu, A., Wong, S.C.w.: Tight bounds for approximate carathéodory and beyond. arXiv preprint arXiv:1512.08602 (2015)
46. Pettie, S., Sanders, P.: A simpler linear time 2/3- $\varepsilon$ approximation for maximum weight matching. Information Processing Letters **91**(6), 271–276 (2004)
47. Pinsker, M.S.: On the complexity of a concentrator. In: 7th International Telegraffic Conference, vol. 4, pp. 1–318. Citeseer (1973)
48. Porter, G., Strong, R., Farrington, N., Forencich, A., Chen-Sun, P., Rosing, T., Fainman, Y., Papen, G., Vahdat, A.: Integrating microsecond circuit switching into the data center. SIGCOMM (2013)
49. Prabhakar, B., McKeown, N.: On the speedup required for combined input-and output-queued switching. Automatica **35**(12), 1909–1920 (1999)
50. Rabin, M.O.: Efficient dispersal of information for security, load balancing, and fault tolerance. Journal of the ACM (JACM) **36**(2), 335–348 (1989)
51. Roy, A., Zeng, H., Bagga, J., Porter, G., Snoeren, A.C.: Inside the social network's (datacenter) network. In: SIGCOMM (2015)
52. Schrijver, A.: Combinatorial Optimization - Polyhedra and Efficiency. Springer (2003)
53. Shieh, A., Kandula, S., Greenberg, A.G., Kim, C.: Seawall: Performance isolation for cloud datacenter networks. In: HotCloud (2010)
54. Singla, A., Singh, A., Chen, Y.: Osa: An optical switching architecture for data center networks with unprecedented flexibility. In: NSDI (2012)
55. Srikant, R., Ying, L.: Communication Networks: An Optimization, Control and Stochastic Networks Perspective. Cambridge University Press, New York, NY, USA (2014)
56. Towles, B., Dally, W.J.: Guaranteed scheduling for switches with configuration overhead. Networking, IEEE/ACM Transactions on **11**(5), 835–847 (2003)
57. Valiant, L.G.: A bridging model for parallel computation. Communications of the ACM **33**(8), 103–111 (1990)
58. Wang, C.H., Javidi, T.: Adaptive policies for scheduling with reconfiguration delay: An end-to-end solution for all-optical data centers. arXiv preprint arXiv:1511.03417 (2015)
59. Wang, C.H., Javidi, T., Porter, G.: End-to-end scheduling for all-optical data centers. In: Computer Communications (INFOCOM), 2015 IEEE Conference on, pp. 406–414. IEEE (2015)
60. Wang, G., Andersen, D.G., Kaminsky, M., Papagiannaki, K., Ng, T., Kozuch, M., Ryan, M.: c-through: Part-time optics in data centers. SIGCOMM (2011)
61. Wu, B., Yeung, K.L.: Nxg05-6: Minimum delay scheduling in scalable hybrid electronic/optical packet switches. In: GLOBECOM (2006)
62. Wu, B., Yeung, K.L., Wang, X.: Nxg06-4: Improving scheduling efficiency for high-speed routers with optical switch fabrics. In: GLOBECOM (2006)
63. Zhou, X., Zhang, Z., Zhu, Y., Li, Y., Kumar, S., Vahdat, A., Zhao, B.Y., Zheng, H.: Mirror mirror on the ceiling: Flexible wireless links for data centers. SIGCOMM (2012)

## A Direct Routing - Proofs

### A.1 Proof of Theorem 1

*Proof* We first note that for the throughput of an empty schedule is zero, i.e. $f(\{\}) = 0$. Also for any $S \subseteq S' \in 2^{\mathbb{Z}\times\mathcal{M}}$ we have $\min\left\{\sum_{(\alpha,M)\in S}\alpha M, T\right\} \leq \min\left\{\sum_{(\alpha,M)\in S'}\alpha M, T\right\}$ implying $f(S) \leq f(S')$. Hence $f$ is normalized and monotone. Next, using the identify

$$\min(a+b,c) = \min(a,c) + \min(b, c - \min(a,c))$$

for non-negative reals $a, b, c$, we have for $S \in 2^{\mathbb{Z}\times\mathcal{M}}$ and $(\alpha_0, M_0) \notin S$,

$$\begin{aligned} f(S\cup\{(\alpha_0,M_0)\}) &= \Big\| \min\big(\sum_{(\alpha,M)\in S}\alpha M + \alpha_0 M_0, T\big)\Big\|_1 \\ &= \Big\| \min\big(\sum_{(\alpha,M)\in S}\alpha M, T\big) + \min\big(\alpha_0 M_0, T - \min\big(\sum_{(\alpha,M)\in S}\alpha M, T\big)\big)\Big\|_1 \\ \Rightarrow f_S((\alpha_0,M_0)) &= \Big\| \min\big(\alpha_0 M_0, T - \min\big(\sum_{(\alpha,M)\in S}\alpha M, T\big)\big)\Big\|_1, \end{aligned} \tag{13}$$

where $f_S((\alpha_0, M_0))$ denotes the incremental marginal value of adding $(\alpha_0, M_0)$ to the set $S$ (see Section 3.2). Finally, for $S \subseteq S' \in 2^{\mathbb{Z}\times\mathcal{M}}$ and $(\alpha_0, M_0) \notin S'$ we have

$$T - \min\big(\sum_{i\in S'}\alpha_i M_i, T\big) \leq T - \min\big(\sum_{i\in S}\alpha_i M_i, T\big).$$

Combining the above equation with equation (13) we get

$$f_{S'}(\{(\alpha_0,M_0)\}) \leq f_S(\{(\alpha_0,M_0)\}),$$

or in other words $f$ is submodular.

### A.2 Proof of Theorem 2

*Proof* Recall the submodular sum-throughput function $f$ defined in Equation (2). Let $\{(\alpha_1, M_1), \ldots, (\alpha_k, M_k)\}$ be the schedule returned by Algorithm 2. Let $S_i = \{(\alpha_1, M_1), \ldots, (\alpha_i, M_i)\}$ denote the schedule computed at the end of $i$ iterations of the `while` loop and let $S^*$ denote the optimal schedule. Now, since in the $i+1$-th iteration $(\alpha_{i+1}, M_{i+1})$ maximizes $\frac{\min(\alpha M, T_{\text{rem}}(i+1))\|_1}{(\alpha+\delta)} = \frac{f_{S_i}(\{(\alpha,M)\})}{(\alpha+\delta)}$ we have for any $(\alpha, M) \notin S_i$,

$$\begin{aligned} \frac{f_{S_i}(\{(\alpha,M)\})}{(\alpha+\delta)} &\leq \frac{f_{S_i}(\{(\alpha_{i+1},M_{i+1})\})}{(\alpha_{i+1}+\delta_{i+1})} \\ \Rightarrow f_{S_i}(\{(\alpha,M)\}) &\leq \frac{(\alpha+\delta)}{(\alpha_{i+1}+\delta_{i+1})} f_{S_i}(\{(\alpha_{i+1},M_{i+1})\}). \end{aligned} \tag{14}$$

Now consider $\texttt{OPT} - f(S_i)$ for some $i < k$. Since $f$ is monotone we have

$$\begin{aligned} \texttt{OPT} - f(S_i) = f(S^*) - f(S_i) &\leq f(S_i \cup S^*) - f(S_i) \\ &\leq \sum_{(\alpha,M)\in J^*} f_{S_i}(\{(\alpha,M)\}) \\ &\leq \sum_{(\alpha,M)\in J^*} \frac{(\alpha+\delta)}{(\alpha_{i+1}+\delta_{i+1})} f_{S_i}(\{(\alpha_{i+1},M_{i+1})\}) \qquad (15) \\ &\leq \frac{W}{(\alpha_{i+1}+\delta_{i+1})} f_{S_i}(\{(\alpha_{i+1},M_{i+1})\}), \qquad (16) \end{aligned}$$

where $J^* := S^* \backslash S_i$ denotes the set of matchings that are present in the optimal solution but not in $S_i$, Equation (15) follows from Equation (14), and Equation (16) follows because $\sum_{(\alpha,M)\in J^*}(\alpha+\delta) \leq \sum_{(\alpha,M)\in S^*}(\alpha+\delta) \leq W$. Next, observe that

$$\begin{aligned}
& f(S_{i+1}) = f(S_i) + f_{S_i}(\{(\alpha_{i+1}, M_{i+1})\}) \\
\Rightarrow\ & \texttt{OPT} - f(S_{i+1}) = \texttt{OPT} - f(S_i) - f_{S_i}(\{(\alpha_{i+1}, M_{i+1})\}) \\
& \leq (\texttt{OPT} - f(S_i)) \left(1 - \frac{(\alpha_{i+1}+\delta)}{W}\right) && (17) \\
& \leq (\texttt{OPT} - f(S_0)) \prod_{i'=1}^{i+1} \left(1 - \frac{(\alpha_{i'}+\delta)}{W}\right) \\
& \leq \texttt{OPT} \times e^{-\sum_{i'=1}^{i+1}(\alpha_{i'}+\delta)/W}, && (18)
\end{aligned}$$

where Equation (17) follows from Equation (16) and Equation (18) follows because of the identity $1 - x \leq e^{-x}$. Now, since after the $k$-th iteration the `while` loop terminates, this implies $\sum_{i'=1}^{k}(\alpha_{i'}+\delta) > W$. However, if the entries of the input traffic matrix $T$ are bounded by $\epsilon W + \delta$, then no matching has a duration longer than $\epsilon W$. In particular $\alpha_k + \delta \leq \epsilon W \Rightarrow \sum_{i'=1}^{k-1}(\alpha_{i'}+\delta) \geq W(1-\epsilon)$. Thus, setting $i = k-2$ in Equation (18) we have

$$\begin{aligned}
& \texttt{OPT} - f(S_{k-1}) \leq \texttt{OPT} \times e^{-\sum_{i'=1}^{k-1}(\alpha_{i'}+\delta)/W} \leq \texttt{OPT} \times e^{-(1-\epsilon)} \\
& \Rightarrow \texttt{OPT} - \texttt{ALG2} \leq \texttt{OPT} \times e^{-(1-\epsilon)}.
\end{aligned}$$

Hence we conclude $\texttt{ALG2} \geq \texttt{OPT}(1 - e^{-(1-\epsilon)})$.

## A.3 Correctness

Consider any traffic matrix $T \in \mathbb{Z}^{n\times n}$. Let $\mathcal{T} = \{T(i,j) : i,j \leq [n]\}$ denote the distinct entries in the matrix $T$. Then, in the following, we show that the maximizer in

$$\max_{\alpha\in\mathbb{Z}, M\in\mathcal{M}} \frac{\|\min(T, \alpha M)\|_1}{\alpha+\delta} \tag{19}$$

occurs for $\alpha \in \mathcal{T}$. To do this, for any matching $M \in \mathcal{M}$ let us define $f_M(\alpha) \triangleq \|\min(\alpha M, T)\|_1$ and let $f(\alpha) \triangleq \max_{M\in\mathcal{M}} \frac{f_M(\alpha)}{\alpha+\delta}$. We then have the following proposition.

**Proposition 1** *$f_M(\alpha)$ is (i) non-decreasing, (ii) piece-wise linear where the corner points are from $\mathcal{T}$ and (iii) concave.*

*Proof* It is easy to see (i) because if $\alpha_1 \leq \alpha_2$ then $\min(\alpha_1 M, T) \leq \min(\alpha_2 M, T)$ entrywise and hence $f_M(\alpha_1) \leq f_M(\alpha_2)$. To see (ii) consider any $t_1 < t_2 \in \mathcal{T}$ such that no other element of $\mathcal{T}$ is between $t_1$ and $t_2$. Then for $t_1 \leq \alpha \leq t_2$ we have

$$\begin{aligned}
f_M(\alpha) &= \|\min(\alpha M, T)\|_1 \\
&= \sum_{\substack{(i,j)\in M \\ T(i,j)\leq t_1}} \min(\alpha, T(i,j)) + \sum_{\substack{(i,j)\in M \\ T(i,j)\geq t_1}} \min(\alpha, T(i,j)) \\
&= \sum_{\substack{(i,j)\in M \\ T(i,j)\leq t_1}} T(i,j) + \sum_{\substack{(i,j)\in M \\ T(i,j)\geq t_1}} \alpha \\
&= \sum_{\substack{(i,j)\in M \\ T(i,j)\leq t_1}} T(i,j) + |\{(i,j)\in M : T(i,j) \geq t_1\}|\,\alpha. && (20)
\end{aligned}$$

Thus $f_M(\cdot)$ is linear for $t_1 \leq \alpha \leq t_2$ and (ii) follows. (iii) also follows from Equation (20) by observing that

$$|\{(i,j) \in M : T(i,j) \geq t_1\}| \geq |\{(i,j) \in M : T(i,j) \geq t_2\}|$$

for any $t_1 < t_2 \in \mathcal{T}$. Hence the slope of the piece-wise linear function $f_M(\alpha)$ is non-increasing as $\alpha$ increases. In other words, $f_M(\alpha)$ is concave.

Next, we consider a case where the matching is fixed.

**Proposition 2** *For a fixed matching $M$, we have* $\arg\max_\alpha \frac{f_M(\alpha)}{\alpha+\delta} \in \mathcal{T}$.

*Proof* This follows from Proposition 1-(ii). Let $f_M(\alpha)$ be linear for $\alpha \in [t_1, t_2]$. Then it can be written as $f_M(\alpha) = f_M(t_1) + m(\alpha - t_1)$ for some slope $m \geq 0$. Now, consider the derivation of the function $f_M(\alpha)/(\alpha+\delta)$ in the interval $[t_1, t_2]$:

$$\begin{aligned}\frac{d}{d\alpha}\left(\frac{f_M(\alpha)}{\alpha+\delta}\right) &= \frac{d}{d\alpha}\left(\frac{f_M(t_1)+m(\alpha-t_1)}{\alpha+\delta}\right) \\ &= \frac{(\alpha+\delta)(m) - (f_M(t_1)+m(\alpha-t_1))}{(\alpha+\delta)^2} \\ &= \frac{\delta m - f_M(t_1) + mt_1}{(\alpha+\delta)^2}. \end{aligned} \tag{21}$$

Note that the numerator of Equation (21) is independent of $\alpha$ and the denominator is strictly positive. Hence the sign (i.e., $> 0, < 0$ or $= 0$) of the slope of $f_M(\alpha)/(\alpha+\delta)$ is the same in the interval $[t_1, t_2]$. This proves that the maximum must occur at either of the extreme points $t_1$ or $t_2$. By Proposition 1-(ii) we know that the $f_M(\alpha)$ is piece-wise linear with the corner points from the set $\mathcal{T}$. Thus we can conclude that the maximum must occur at one of the points in $\mathcal{T}$.

We are now ready to show that the maximizer of Equation (19) occurs for $\alpha \in \mathcal{T}$.

**Theorem 4** $\arg\max_\alpha f(\alpha) \in \mathcal{T}$.

*Proof* This follows directly from Proposition 2. Notice that

$$\max_\alpha f(\alpha) = \max_\alpha \max_M \frac{f_M(\alpha)}{\alpha+\delta} = \max_M \left(\max_\alpha \frac{f_M(\alpha)}{\alpha+\delta}\right).$$

But the maximizer of $f_M(\alpha)/(\alpha+\delta)$ belongs to $\mathcal{T}$ for any $M$. Hence we conclude that the maximizer of $f(\alpha)$ also belongs to $\mathcal{T}$ and the Theorem follows.