LEARNING-AUGMENTED ONLINE OPTICAL INTERCONNECT RECONFIGURATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Optical interconnects are a novel switching technology currently deployed in data center networks due to their extremely low latency and high bandwidth. However, dynamically establishing optical circuits is not free and introduces undesirable delays, disrupting the network during reconfigurations. Existing approaches focus on two ends of a spectrum: either they dynamically reconfigure the network in an oblivious manner, or they optimize for long-lasting static solutions. We study the online optimization problem of such interconnects and present a learning-augmented algorithm that decides when and how to reconfigure. Our algorithm leverages predictions of the incoming traffic to strike a smooth tradeoff between the two previous extremes and shows close-to-optimal performances with regards to the predictions' accuracy.

1 INTRODUCTION

The network traffic of data centers grows explosively, roughly doubling each year, while the bandwidth capacity of standard electrical switches is expected to double only every two years Mellette et al. (2017); Singh et al. (2016). As a result, the time spent routing the network traffic becomes a significant portion of the total execution time of many data center applications. Optical interconnects are a novel switching technology currently deployed in data centers to address this challenge. Their main advantage is an extremely low latency thanks to a routing without computational decision-making, avoiding conversions of optical to electrical signals Ballani et al. (2020). This switching technology is already deployed in data centers to forward the traffic between the computing units Chen et al. (2013; 2017); Farrington et al. (2010); Porter et al. (2013).

Unlike electrical packet switches, optical interconnects however come with a unique set of constraints. At any given time, an optical switch can communicate the traffic only between a restricted set of port pairs that must form a *matching* Mellette et al. (2017). In addition, an optical switch does not suport packet buffering Ballani et al. (2020), which means that any packet whose path falls outside the switch's matching is either lost or handled by a backup switch. Although optical switches can *reconfigure* their matching to adapt to the demand, a reconfiguration can incur a significant delay (up to the order of the microsecond Mellette et al. (2017)) during which the routing of the whole switch is altered Liu et al. (2015a).

These constraints of optical switches, together with the need for modular data centers Hamilton (2006), motivated the development of *hybrid networks* Farrington et al. (2010); Liu et al. (2015a) combining both switch types. This combination works well because datacenter traffic usually has two properties: it follows predictable patterns such as daily cycles or periodic workload spikes Benson et al. (2011); Poutievski et al. (2022), and it exhibits high spatial Kapoor et al. (2013); Roy et al. (2015) and temporal Kandula et al. (2009); Roy et al. (2015); Avin et al. (2020b) locality, with a large fraction concentrated on a few input/output port pairs over long time windows. A well-configured optical switch can therefore handle a large share of the traffic, with the electrical switch routing only the remaining traffic.

1.1 THE CRUX TO HANDLE AN OPTICAL SWITCH: WHEN AND HOW TO RECONFIGURE

The main algorithmic challenge is to determine when and how to reconfigure the connection matching to serve a maximal amount of traffic with low latency. Offline approaches assume full knowledge of

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

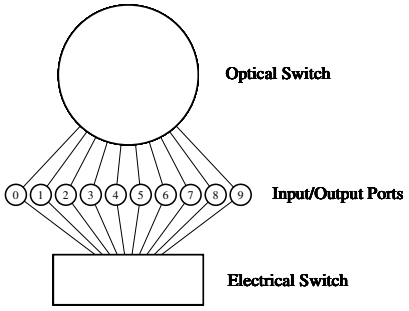


Figure 1: The hybrid switch is made of two switches, an optical and a standard, electrical one. If the optical switch can forward the incoming packet to the correct output port then the packet goes through it, otherwise the packet goes through the electrical switch, introducing additional latency.

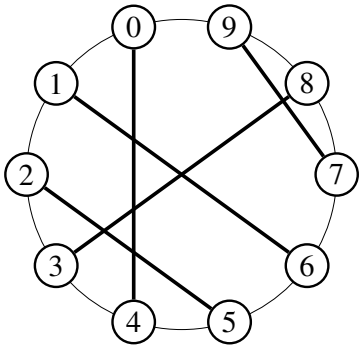


Figure 2: A representation of an internal matching in the optical switch.

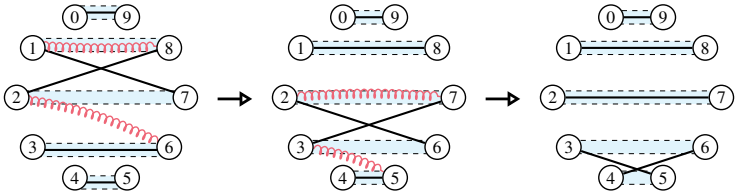


Figure 3: A possible evolution of the configuration inside the optical switch. The black, solid lines represent the configuration at the depicted time; the blue, horizontal zones are an optimal static configuration that would minimize the overall latency; the red, curly lines represent a set of edges that are requested intensively at the corresponding time intervals. This example gives an intuition that reconfiguring to the current most requested edges may not be sufficient to converge towards the optimal configuration even if, like here, the initial (predicted) configuration was almost accurate.

future traffic Porter et al. (2013); Liu et al. (2015b) and schedule reconfigurations ahead of time; we instead design *online* algorithms that operate in real time without such trustworthy knowledge.

Known online algorithms fall into two types. Algorithms of the first type are based on oblivious schedules Mellette et al. (2017); Ballani et al. (2020); Addanki et al. (2023); Amir et al. (2024): the switches reconfigure themselves regularly in a round-robin manner to provide all-to-all connectivity, regardless of actual traffic patterns. Algorithms of the second type periodically compute and reconfigure to a static network configuration that performs well under the current traffic workload Porter et al. (2013); Liu et al. (2015a); Bojja Venkatakrishnan et al. (2016); Kulkarni et al. (2021); Poutievski et al. (2022); Wang et al. (2023). The time period for reconfiguration is constant and chosen at the initialization, which leads to suboptimal configurations if the traffic changes are too fast.
We therefore ask: *Can we decide both when and how to reconfigure the optical switch matching in a fully autonomous yet reliable manner?*

1.2 Contributions

This paper first introduces an optimization model of the hybrid switch aiming to minimize the latency. We show that in the purely online, worst-case oriented setting, all algorithms (even randomized) incur a latency at least $O(n)$ times worse than that of an optimal, offline algorithm that knows the incoming traffic, where n is the number of ports of the optical switch (Theorem 2).

The main contribution of this paper is an online, learning-augmented algorithm that can leverage predictions of the incoming traffic to help make its decisions (Theorem 4). As a building block, we first derive an offline algorithm Off for our problem (in Section 4) and show it is constant competitive (Theorem 3). We then use Off as a target for our learning-augmented algorithm Pred. We show that Pred is both *consistent* and *robust* Lykouris & Vassilvitskii (2021): when

the predictions are accurate PRED inherits the competitive ratio of OFF, and when the predictions are incorrect PRED is $O(n \log n)$-competitive at worst. In addition, PRED is resilient to small error, i.e., it is *smooth*: a small number of mispredicted edges still guarantees a good competitive ratio (Theorem 5).

1.3 CHALLENGE

The primary algorithmic challenge is to effectively integrate both the predictions and the online processing into a coherent, competitive algorithm. Once the algorithm queried a prediction that suggests a set of edges for reconfiguration, the online processing part ideally takes advantage of the accurate predictions while gradually correcting the wrong predictions as more requests are revealed. We must design the online processing to provably always improve upon the predicted configuration and to prevent the destruction of a well-predicted configuration. We show that our online, learning-augmented algorithm (PRED, Algorithm 3) has this property in Lemma 1: after it queried the prediction, PRED can only improve on the predicted configuration, i.e., the error function strictly decreases with time. Figure 3 illustrates one of the cases we encountered while designing our algorithm, where the online decisions lead to non-optimal configurations, resulting in poor performances.

2 MODEL

We consider an optical switch with n (even) ports (called nodes) together with an electrical switch which are used to route incoming packets (see Figure 1). We model the possible connections inside the optical switch with a complete, undirected graph $G = (V, E)$ with n nodes. Each node can both send to and receive from other nodes[1]. An arbitrary sequence of communication requests (packets) $\sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_T\}$ comes over time, where each request $\sigma_t = (x_t, y_t)$ specifies a communication between nodes x_t and y_t at discrete time t.

An algorithm should maintain a dynamic matching $M(t)$ between the nodes at every discrete time t (see Figure 2). Each request σ_t must be served using the matching configuration $M(t-1)$ from the previous time step. After serving σ_t, the matching can be reconfigured from $M(t-1)$ to $M(t)$ in order to adapt to future requests. The objective is to minimize the overall latency (cost). A cost can occur when we serve a request or when we reconfigure the matching. At time t and matching $M(t-1)$, the costs are:

(1) Serving σ_t costs 0 if σ_t belongs to the current matching $M(t-1)$
(2) Serving σ_t costs 1 if σ_t does not belong to the current matching $M(t-1)$
(3) Updating the matching costs α, with $\alpha > 1$.[2]

The goal is to design an online algorithm that dynamically updates the matching in response to the sequence σ in a way that minimizes the total cost, defined as:

$$\text{Total Cost} = \sum_{t=1}^{T} (\text{serving cost for } \sigma_t) \quad + \quad \sum_{t=1}^{T} (\text{cost } \alpha \text{ for each reconfiguration}) .$$

In the literature, our model is almost similar to that of Liu et al. (2015b) except for two differences: (1) we assume that the forwarding time of the optical switch is simply negligible compared to that of the electrical switch (the worst case for our measure of competitive analysis), and (2) we model discrete packet arrivals while they consider a fixed demand matrix.

Competitive Analysis. Competitive analysis is the standard tool to evaluate online algorithms Borodin et al. (1992). Competitive analysis compares the online algorithm ALG with an optimal offline algorithm, OPT, that has full knowledge of the request sequence in advance. We assume both algorithms begin in the same initial configuration and define ALG_σ and OPT_σ as their costs, respectively, on a request sequence σ. For a deterministic algorithm, ALG is c-competitive if for every possible input sequence σ, ALG works not worse than c times the cost of OPT, $c = \sup_\sigma \text{ALG}_\sigma / \text{OPT}_\sigma$.

[1]The bi-directional links model Poutievski et al. (2022); Liu et al. (2015b).
[2]The constant reconfiguration cost model Liu et al. (2015b)

Prediction Model. We design a learning-augmented, online algorithm (PRED, Algorithm 3) that uses predictions to improve its competitive ratio Purohit et al. (2018). PRED can query a *predictor* at any time. When queried, the predictor suggests a configuration (a matching) for the optical switch that is likely good for the future requests. The learning-augmented algorithm is free to use the predicted matching at its discretion. We do not assume any guarantee on the predictor's side: the predictions may not correlate with the requests or may actively try to degrade the algorithm's cost. Under this (standard) prediction model, we calculate two metrics for our learning-augmented algorithm Lykouris & Vassilvitskii (2021): (i) the *consistency* is the competitive ratio when the predictions are perfect (i.e., the error function is zero, see Definition 2) and (ii) the *robustness* is the competitive ratio when the predictions are arbitrarily bad. In this paper, we go further and show that our algorithm exhibits a trade-off between consistency and robustness by providing an upper-bound on the competitive ratio as a function of the predictor's error (*smoothness*, see Theorem 5): our upper-bound slowly degrades as the error grows, which demonstrates resilience to small errors.

Preliminaries and Notation. We denote by ALG_σ the total cost of serving the sequence σ using the algorithm ALG. The cost of an optimal offline algorithm which has full knowledge of the request sequence in advance is denoted by OPT_σ. For any algorithm ALG, a phase is defined as a subsequence of requests during which the algorithm maintains the same matching configuration. The first phase starts with the first request, and each phase ends when ALG reconfigures its matching or the sequence ends. The i-th phase is denoted by P_i, and the cost incurred by ALG during this phase is ALG_{P_i}.

3 Online Algorithms Without Predictions

This section covers the performance of online algorithms for the optical matching problem in the absence of predictions. We have two contributions: an $O(n)$-competitive online deterministic algorithm (Theorem 1) and a lower bound of $\Omega(n)$ for competitiveness of any (randomized or deterministic) online algorithm (Theorem 2). The proofs of the theorems can be found in Appendix A.1 and A.2.

The algorithm GREEDY serves the requests one by one and tracks the number of requests between each pair of nodes. When the counter for a pair (x, y) reaches a threshold $\lambda = \alpha/n$, GREEDY makes x and y connected by swapping their neighbours and resets the counter for (x, y). The pseudocode is provided in Algorithm 1.

Algorithm 1 Online Algorithm (Greedy)

Input: initial matching $M(0)$, reconfiguration cost α, the threshold λ

```
C(e) ← 0 for all e ∈ E
for request σ_t = (x_t, y_t) do
    if σ_t ∉ M(t − 1) then
        Increment the counter: C(σ_t) ← C(σ_t) + 1
        if C(σ_t) ≥ λ then
            Reset the counter: C(σ_t) ← 0
            Reconfigure the matching by swapping x_t with the node currently matched with y_t
        end if
    end if
end for
```

Theorem 1. GREEDY *is* $6n$*-competitive for* $\lambda = \alpha/n$.

Then, we can prove the lower bound of $\Omega(n)$ for competitiveness of any randomized or deterministic online algorithm. As we will see in the next section, much better competitive ratios can be achieved in practical scenarios where communication patterns are to some extent predictable and not arbitrary.

Theorem 2. *No randomized (or deterministic) online algorithm for the optical interconnect reconfiguration problem can achieve competitive ratio better than* $n/12$.

4 A Polynomial-Time 4-Approximation

In this section, we present an offline algorithm (OFF, Algorithm 2) and show it is a 4-approximation of the offline version of our problem. Considered alone, OFF is not optimal and we present it solely as a building block for our main algorithm PRED (Algorithm 3). In particular, we will define the

Algorithm 2 Offline Algorithm (OFF)

Input: initial matching $M(0)$, reconfiguration cost α

1: $w^*(e) \leftarrow 0$ for all $e \in E$
2: **for** request $\sigma_t = \{x_t, y_t\}$ **do**
3: $\quad w^*(\sigma_t) \leftarrow w^*(\sigma_t) + 1$
4: $\quad$ **if** $w^*(E) - w^*(M(0)$ if this is the first phase, else a maximum matching$) \geq \frac{\alpha}{3}$ **then**
5: $\quad\quad \hat{w}(e) \leftarrow 0$ for all $e \in E$
6: $\quad\quad$ Set Q to the current matching: $Q \leftarrow M(t-1)$
7: $\quad\quad$ **for** $s \in [t+1, T]$ **do**
8: $\quad\quad\quad \hat{w}(\sigma_s) \leftarrow \hat{w}(\sigma_s) + 1$
9: $\quad\quad\quad Q \leftarrow$ a maximum matching in $\hat{w}$
10: $\quad\quad\quad$ **if** $\hat{w}(E \setminus Q) \geq \frac{\alpha}{3}$ **then**
11: $\quad\quad\quad\quad$ break
12: $\quad\quad\quad$ **end if**
13: $\quad\quad$ **end for**
14: $\quad\quad$ reconfigure to Q
15: $\quad\quad w^*(e) \leftarrow 0$ for all $e \in E$
16: $\quad$ **end if**
17: **end for**

behavior of OFF as the target behavior for PRED. PRED will therefore act similarly as OFF in case of perfect predictions, i.e., in case the error function is zero (see Definition 2).

Our offline algorithm OFF maintains a request graph: each edge e has a weight $w^*(e)$ that equals the number of requests to e since the last reconfiguration. When $\frac{\alpha}{3}$ requests fall outside a maximum matching of the request graph, the system reconfigures. To choose the next matching, the system iterates over the future requests one by one (incrementing variables $\hat{w}(e)$) and stops whenever $\frac{\alpha}{3}$ requests fall outside the maximum matching for weights $\hat{w}$ (called P). Then, the system goes back to the present moment and reconfigures to P. In case the system reaches the last request without meeting the stopping condition, it picks a maximum matching P, goes back to the present moment and reconfigures to P.

The following theorem about the algorithm holds while its proof is presented in Appendix A.3.

Theorem 3. *Algorithm 2 is a 4-approximation on any of its phases except the first and the last one. Algorithm 2 (OFF) is a 4-approximation taken over the whole input sequence. It can be implemented to run in $O(T \cdot n^3)$ time.*

5 A LEARNING-AUGMENTED ONLINE ALGORITHM

Fortunately, data center traffic is often far from worst-case and exhibits spatial and temporal structure that can be exploited. We introduce PRED (Algorithm 3), a learning-augmented online algorithm that is 4-consistent and $O(n \log n)$-robust.

As the requests arrive, PRED may trigger a reconfiguration for two reasons: either it calls the predictor and reconfigures to the predicted matching with no consideration for the previous requests, or otherwise it reconfigures solely based on the previous requests without the predictor's help. We analyze PRED on subsequences of requests separated by calls to the predictor, we call those subsequences *superphases*. After reconfiguring to a predicted matching, PRED evaluates the quality of that predicted matching as more requests are revealed through the superphase, and may reconfigure by itself (without the predictor) if it assesses that the current matching is not optimal for the arriving traffic. Our key insight is to design PRED such that, for any request sequence, the number of reconfigurations of PRED during a superphase is *bounded by the initial error of the predicted matching* (Lemma 1), where the error counts the number of wrongly matched edges (see Definition 2); in particular, this fact enables us to derive our final smoothness bound (Theorem 5) since it implies that small errors cause small cost gap with an optimal offline solution.

To make its decisions, PRED tracks two request graphs: one with weight w^*, identical to OFF (Algorithm 2) and used to decide when to call the predictor, and one with weight w used to decide when to reconfigure on its own without the predictor. Whenever OFF would trigger a reconfiguration with weight w^* (line 8), PRED resets all the weights and starts a new superphase: it queries the

Algorithm 3 Learning-Augmented, Online Algorithm (PRED)

Input: initial matching $M(0)$, reconfiguration cost α

1: $w(e) \leftarrow 0$ for all $e \in E$
2: $w^*(e) \leftarrow 0$ for all $e \in E$
3: **for** request $\sigma_t = (x_t, y_t)$ **do**
4: $\quad w^*(\sigma_t) \leftarrow w^*(\sigma_t) + 1$
5: $\quad$ **if** σ_t is not in the current matching **then**
6: $\quad\quad w(\sigma_t) \leftarrow w(\sigma_t) + 1$
7: $\quad$ **end if**
8: $\quad$ **if** $w^*(E) - w^*(M(0)$ if this is the first superphase, else a maximum matching$) \geq \frac{\alpha}{3}$ **then**
9: $\quad\quad$ Reconfigure to the predicted matching
10: $\quad\quad w(e) \leftarrow 0$ for all $e \in E$
11: $\quad\quad w^*(e) \leftarrow 0$ for all $e \in E$
12: $\quad$ **end if**
13: $\quad$ **if** $\exists$ matching $M', \forall e' \in M', w(e') \geq \frac{\alpha}{|M'|} + 1$ **then**
14: $\quad\quad$ Unmatch the edges that are in conflict with M'
15: $\quad\quad$ Match the edges of M'
16: $\quad\quad w(e) \leftarrow 0$ for all $e \in E$
17: $\quad$ **end if**
18: **end for**

prediction and reconfigures to it. Within each superphase, PRED constantly checks whether there exists a matching M' whose weight w satisfy a threshold condition (line 13); if so, it reconfigures to M' and begins a new phase — recall a phase is a request subsequence between two reconfigurations. We designed this new-phase condition so that the error (Definition 2) strictly decreases after a reconfiguration (Lemma 1): the threshold is (1) *per edge*, which ensures an even distribution of the weights within M' and certifies that even OPT must incur a cost for each edge if it did not match it, and (2) the threshold is *inversely proportional to* $|M'|$ which ensures a consistent loss for OPT for any size of M'. This division by $|M'|$ is responsible for the logarithmic multiplicative overhead compared to GREEDY in case of wrong predictions; this logarithmic gap is not an artifact of the analysis and is reached when the end of each phase is delayed as much as possible to maximize the weight w (see Lemma 2). We present the pseudocode in Algorithm 3.

We now state our main result on the consistency and robustness of PRED.

Theorem 4 (Consistency and Robustness)**.** PRED *(Algorithm 3) is* 4*-consistent and* $O(n \log n)$*-robust.* PRED *can be implemented to run in time* $O(T \cdot n^4)$.

Proof. The claims on consistency and robustness are directly implied by Theorem 5 and the fact that η is no greater than n. We explain the claimed running time in Appendix A.4. □

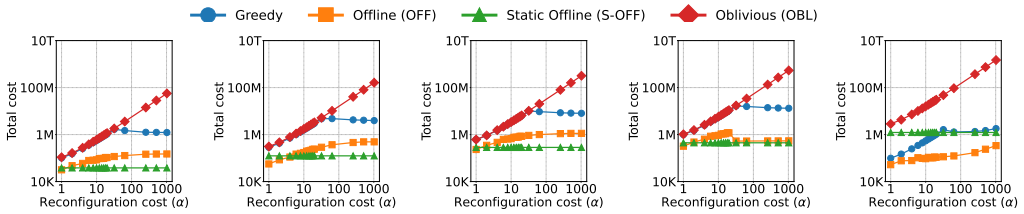
Definition 1 (Superphase)**.** *A* superphase *is the set of requests issued between two consecutive reconfiguration events involving the prediction, that is, at which line 9 of Algorithm 3 is called.*

Definition 2 (Error Function)**.** *Let S be a superphase of* OFF*. Let M^{OFF} be the configuration of* OFF *during S and let M^{PRED} the predicted matching at the beginning of S. We define the error as $\eta = |M^{\text{OFF}} \setminus M^{\text{PRED}}|$.*

Lemma 1. *Consider a superphase S made of phases $P_1, P_2 \ldots$ For a given phase P_i of S, we call η_i the error function during P_i — η_0 is the error function at the beginning of S. The series $\eta_0, \eta_1 \ldots$ is strictly decreasing.*

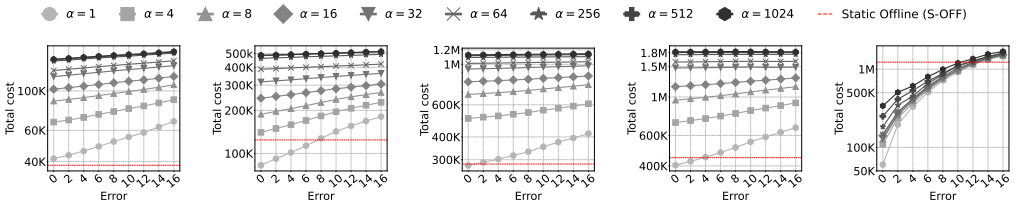
Proof. We prove the claim by contradiction: suppose that there exists two consecutive phases P_{i-1} and P_i in S such that $\eta_{i-1} \leq \eta_i$. We start by proving that strictly more than $\alpha/3$ requests fall outside of M^{OFF} during P_{i-1}. As in PRED, we call M' the matching whose edges are rematched at the end of P_{i-1} at line 15 — notice that M' may not cover the whole the graph. We call $B \subseteq M^{\text{OFF}}$ the set of edges in M^{OFF} that were unmatched from P_{i-1} to P_i, and $G \subseteq M^{\text{OFF}}$ the set of edges in M^{OFF} that were newly matched from P_{i-1} to P_i; it holds that $|G| \leq |B|$ since the error increases: $\eta_{i-1} \leq \eta_i$. Let $D = M' \setminus M^{\text{OFF}}$. Notice that $2|D| \geq |B|$ as a rematched edge unmatches at most two edges. Using the two previous inequalities on the equality $|G| + |D| = |M'|$ (definition of D) gives

$$|G| + |D| = |M'| \implies |D| \geq \frac{|M'|}{3}$$



(a) HPC Mocfe (b) HPC Nekbone (c) HPC Boxlib (d) HPC Combined (e) pFabric

Figure 4: Performance of Greedy, OFF, Static-OFF, and oblivious (OBL) across different workloads and reconfiguration costs.



(a) HPC Mocfe (b) HPC Nekbone (c) HPC Boxlib (d) HPC Combined (e) pFabric

Figure 5: Performance of our learning-augmented algorithm PRED under different prediction errors and reconfiguration costs.

Using this inequality and the saturation condition of M' on line 13, we obtain that D receives no fewer than $\frac{\alpha}{3}$ requests during P. We now prove that before the superphase S ends, there exists a matching different from M^{OFF} triggering the stopping condition at line 8, a contradiction. M^{OFF} is the maximum matching at the end of S, so there exists a request in $P_i \cup \cdots \cup P_q$ after which all edges of M^{OFF} are in a maximum matching with regard to w^*. Let t be the index of the first such request, $w^*(M^{\text{OFF}}(t))$ be the weight of M^{OFF} at time t and $w^*(E \setminus M^{\text{OFF}}(t))$ be the rest weight. It holds that $w^*(M^{\text{OFF}}(t)) > \frac{\alpha}{3}$, otherwise D would be a matching with weight strictly greater than M^{OFF}'s. Also, it holds that $w^*(E \setminus M^{\text{OFF}}(t)) \geq w^*(D) > \frac{\alpha}{3}$. One request earlier at index $t-1$, a maximum matching had weight at least $w^*(M^{\text{OFF}}(t)) - 1 \geq \frac{\alpha}{3}$ and the rest of the graph had weight at least $w^*(E \setminus M^{\text{OFF}}(t)) - 1 \geq \frac{\alpha}{3}$; the superphase should have ended at request $t-1$, a contradiction. □

Lemma 2. *At the end of any phase P_i, no matching has a total weight $w(.)$ greater than $\alpha \cdot H(n/2)$, where $H(n/2) = \sum_{j=1\ldots n/2} \frac{1}{j}$ is the harmonic sum.*

Theorem 5 (Smoothness)**.** PRED *is $4 \cdot (1 + \eta \cdot [2 + \ln n])$ - competitive during any superphase that is not the first one (where the predictor was not called) or the last one (since it may be not complete).*

6 EVALUATIONS

Our evaluation focuses on empirically validating our theoretical findings under real-world datacenter workloads, addressing the following key questions:

(Q1) *How does the performance of our* GREEDY *algorithm vary with the reconfiguration cost?*

(Q2) *Can the learning-augmented algorithm* PRED *improve performance?*

(Q3) *How does the performance of* PRED *vary with the error in the prediction?*

6.1 SETUP

We developed a custom simulator for our evaluation, which will be open-sourced. Our experiments were run on AMD EPYC 9335 with 768 GB RAM (code in supplementary material). Below, we detail the evaluation setup.

Network: We consider a 32-port optical circuit switch (e.g., Polatis Polatis (2025)) interconnecting 32 nodes by establishing an optical matching at each time instance. The switch is reconfigurable and can be reconfigured with a fixed cost of α. Additionally the cluster is also connected via an electrical switch.

Workloads: We evaluate our algorithms on five datacenter workloads: Mocfe, Nekbone, Boxlib, "Combined" that concatenates the previous three workloads after shuffling the source-destination indices for each separately, and pFabric. These workloads are representative of the traffic patterns observed in HPC clusters CESAR (NA) and large-scale datacenters Alizadeh et al. (2013). We consider the traffic generated by these workloads as the requests that need to be routed.

Comparisons: We compare four algorithms: *(i)* GREEDY with $\lambda = \frac{\alpha}{n}$, *(ii)* Offline OFF, *(iii)* a static offline algorithm (denoted by S-OFF) that fixes the best matching for all future requests, and *(iv)* an oblivious algorithm (OBL) similar to RotorNet Mellette et al. (2017) that reconfigures periodically with a fixed schedule that emulates a complete graph over time and one reconfiguration is triggered for each request.

Cost: The performance of each algorithm is based on the cost model described earlier. If a request is served via the optical switch, its cost is 0 and 1, otherwise. Intuitively, the cost captures the latency profile of each algorithm. For instance, sending a packet over an optical circuit switch has no processing, queueing, and electrical-optical conversion delays that are typical over electrical packet-switched network. A lower cost indicates better performance. We report each algorithm's *total cost* values for each workload.

Predictions: We evaluate the performance of our algorithm PRED (Algorithm 3) in a separate set of experiments. The modelled predictions mimic the error function η as defined in Definition 2. We first compute the matching sequence of OFF for each workload. PRED queries predictions based on Algorithm 3. For each prediction, we return the current matching of OFF with the introduced error by swapping η edges. We evaluate its performance across various error values and reconfiguration costs.

6.2 RESULTS

GREEDY adapts to reconfiguration costs. Figure 4 shows the performance of GREEDY across different reconfiguration costs. For all workloads, we see that the total cost of OBL increases with the reconfiguration cost. This is because OBL is not only oblivious to the traffic but also, importantly, oblivious to the reconfiguration costs. In contrast, GREEDY adapts to the reconfiguration costs as clearly seen in its total cost that saturates beyond $\alpha = 64$ in our evaluations. For instance, even at $\alpha = 1$, GREEDY outperforms OBL by $1.07\times$ for the HPC traces, and by $28.9\times$ for the pFabric trace. The offline algorithm OFF and Static-OFF perform significantly better than GREEDY as they have complete visibility over the entire request sequence and choose the best matchings to serve it. HPC workloads exhibit strong spatial structure, largely driven by highly regular collective communication patterns. In particular, the ring-based AllReduce algorithm induces near-permutation traffic matrices, enabling Static-OFF to perform well even in the absence of reconfigurations. In contrast, OFF significantly outperforms Static-OFF on the pFabric workload, highlighting the importance of adapting the interconnect to the workload. Unlike HPC collectives, pFabric exhibits little inherent spatial structure, a characteristic that is increasingly common in modern mixture-of-experts training workloads Li et al. (2024). Our results underscore the potential benefits of our approach not only for traditional HPC workloads, but also for emerging, less structured communication patterns. However, OFF and Static-OFF are not practical in real-world scenarios as it requires complete knowledge of the sequence, whereas GREEDY adapts in an online manner.

PRED greatly improves the performance. Figure 5 shows the performance of the prediction-augmented algorithm PRED across different error values and reconfiguration costs. We observe that PRED significantly outperforms GREEDY on all workloads, upto an error as high as 8 (i.e., 50% of the edges are mispredicted). First, we observe that the performance of PRED with zero error (which is exactly OFF algorithm) is close to that of Static-OFF, indicating that PRED is consistent. Second, we observe that the performance of PRED gradually degrades as the error increases[3]. Even at large prediction errors, PRED's performance (Figure 5) remains significantly better than oblivious (Figure 4), effectively answering our main question, i.e., *When and how to reconfigure optical circuits, given a certain reconfiguration delay?*

PRED is robust to prediction errors. We observe from Figure 5 that PRED even with high prediction error[4] outperforms GREEDY with the same reconfiguration cost α (Figure 4), e.g., for the Mocfe trace, PRED with $\alpha = 8$ outperforms GREEDY with $\alpha = 8$ by $4.2\times$, even under large prediction error

[3]Note that PRED's competitive ratio (similarly to GREEDY) is independent of the reconfiguration cost α.

[4]A perfect matching consists of 16 edges in our setup with 32 nodes. So, an error of 16 is the maximum error.

($= 8$), similarly outperforms GREEDY by $7.1\times$ and $5.4\times$ for the Nekbone and Boxlib traces. Two key observations are worth noting: *(i)* HPC traces considered in our evaluation are driven mostly by AllReduce operations (usually implemented with the Ring collective algorithm) which can be efficiently served by a single matching and *(ii)* HPC workloads present a high locality in traffic patterns which makes the global counting and resetting used in PRED effective for these workloads. Even without such structure in pFabric trace, PRED still performs close to GREEDY within $0.97\times$ of its cost (for $\alpha = 8$) even at large prediction errors, indicating that PRED is robust to prediction errors. We also observe that PRED degrades gracefully as the error in the prediction increases, supported by Theorem 5.

7 RELATED WORK

Online Algorithms with Predictions. Online algorithms with predictions, also known as learning-augmented or algorithms with advice, aim to improve online decision-making by leveraging potentially inaccurate forecasts about future inputs. This hybrid model bridges the gap between worst-case guarantees and empirical performance. Foundational works Lykouris & Vassilvitskii (2021) introduce desirable properties such as *consistency* and *robustness*, enabling algorithms to perform well when predictions are accurate while gracefully degrading under adversarial errors. Recent research has extended this paradigm to a variety of online graph problems Azar et al. (2022), which are particularly relevant in networking. These include online minimum spanning tree Erlebach et al. (2022), facility location problems Xu & Moseley (2022), and many other, often with provable improvements over classical baselines.

Practical Approaches to Optical Network Reconfiguration. Real-time algorithms for optical switch reconfiguration fall into two categories. The first category comprises oblivious algorithms based on fixed schedules, such as round-robin configurations, which guarantee all-to-all connectivity regardless of actual traffic Ballani et al. (2020); Mellette et al. (2017); Amir et al. (2024); Addanki et al. (2023). These approaches are practical and mitigate reconfiguration delays but may lead to suboptimal performance when traffic patterns vary significantly. The second category involves demand-aware algorithms, which periodically compute static configurations that align with the current workload Wang et al. (2023); Henzinger et al. (2019); Poutievski et al. (2022); Liu et al. (2015a); Bojja Venkatakrishnan et al. (2016); Porter et al. (2013). While effective, these algorithms often rely on offline traffic knowledge, which may not be available or accurate.

Online Network Reconfiguration Algorithms. A related online rematching problem was introduced in Avin et al. (2020a), and it was considered in the context of the online graph partitioning problem Avin et al. (2016), as a special case with clusters of capacity 2. The major difference to our model is the reconfiguration cost, paid for each edge individually. Extensions of the rematching problem have been explored in the context of competitive b-matching Bienkowski et al. (2022), where the problem is generalized to a setting where each node can have optical connections with b nodes.

The online rematching model significantly differs from the classical online matching model, where a sequence of vertices of a bipartite graph arrives with corresponding edge weights, and the goal is to immediately and irrevocably match the incoming vertex to maximize the total weight of the matching. This problem was first introduced in the context of online advertisement auctions Mehta et al. (2007), where efficiently matching ads to users is critical. Further work, e.g., Buchbinder et al. (2007), has expanded the scope and solutions for this problem. In contrast, the rematching problem operates on a fixed set of nodes, where edges incrementally appear over time. Here, the objective is to minimize the total weight of edges that are not matched at any point, leading to fundamentally different challenges.

8 CONCLUSIONS

This work addressed the critical problem of reconfiguration scheduling for optical switches in networks to minimize total communication latency. We showed the existence of a fundamental limitation: any online algorithm without additional knowledge is $\Omega(n)$-competitive. To alleviate this limitation, we proposed a *learning-augmented algorithm* that leverages predictions of future requests: our algorithm is 4-consistent and $O(n \log n)$-robust. Importantly, the algorithm exhibits a smooth degradation as prediction quality decreases, balancing performance and resilience. We confirm our theoretical results with the evaluation on five real-world workloads. Our work leaves open several avenues for future research. On the theoretical front, it would be interesting to close the remaining gap between lower and upper bounds on the competitive ratio. On the applied front, we plan to implement our prediction oracles on the switching hardware.

REFERENCES

Vamsi Addanki, Chen Avin, and Stefan Schmid. Mars: Near-optimal throughput with shallow buffers in reconfigurable datacenter networks. *Proc. ACM Meas. Anal. Comput. Syst.*, 7(1), mar 2023. doi:10.1145/3579312. URL https://doi.org/10.1145/3579312.

Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. pfabric: minimal near-optimal datacenter transport. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pp. 435–446, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450320566. doi:10.1145/2486001.2486031. URL https://doi.org/10.1145/2486001.2486031.

Daniel Amir, Nitika Saran, Tegan Wilson, Robert Kleinberg, Vishal Shrivastav, and Hakim Weatherspoon. Shale: A practical, scalable oblivious reconfigurable network. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, pp. 449–464, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400706141. doi:10.1145/3651890.3672248. URL https://doi.org/10.1145/3651890.3672248.

Chen Avin, Andreas Loukas, Maciej Pacut, and Stefan Schmid. Online balanced repartitioning. In *Distributed Computing - 30th International Symposium, DISC 2016, Paris, France, September 27-29, 2016. Proceedings*, volume 9888 of *Lecture Notes in Computer Science*, pp. 243–256. Springer, 2016.

Chen Avin, Marcin Bienkowski, Andreas Loukas, Maciej Pacut, and Stefan Schmid. Dynamic balanced graph partitioning. *SIAM J. Discret. Math.*, 34(3):1791–1812, 2020a.

Chen Avin, Manya Ghobadi, Chen Griner, and Stefan Schmid. On the complexity of traffic traces and implications. In *Abstracts of the 2020 SIGMETRICS/Performance Joint International Conference on Measurement and Modeling of Computer Systems, Boston, MA, USA, June, 8-12, 2020*, pp. 47–48. ACM, 2020b. ISBN 978-1-4503-7985-4. doi:10.1145/3393691.3394205. URL https://doi.org/10.1145/3393691.3394205.

Yossi Azar, Debmalya Panigrahi, and Noam Touitou. Online graph algorithms with predictions. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pp. 35–66. SIAM, 2022.

Hitesh Ballani, Paolo Costa, Raphael Behrendt, Daniel Cletheroe, Istvan Haller, Krzysztof Jozwik, Fotini Karinou, Sophie Lange, Kai Shi, Benn Thomsen, et al. Sirius: A flat datacenter network with nanosecond optical switching. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pp. 782–797, 2020.

Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Microte: Fine grained traffic engineering for data centers. In *Proceedings of the seventh conference on emerging networking experiments and technologies*, pp. 1–12, 2011.

Marcin Bienkowski, David Fuchssteiner, and Stefan Schmid. Online b-matchings for reconfigurable datacenters: The power of randomization. *CoRR*, abs/2209.01863, 2022. doi:10.48550/ARXIV.2209.01863. URL https://doi.org/10.48550/arXiv.2209.01863.

Shaileshh Bojja Venkatakrishnan, Mohammad Alizadeh, and Pramod Viswanath. Costly circuits, submodular schedules and approximate carathéodory theorems. In *Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science*, SIGMETRICS '16, pp. 75–88, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342667. doi:10.1145/2896377.2901479. URL https://doi.org/10.1145/2896377.2901479.

Allan Borodin, Nathan Linial, and Michael E. Saks. An optimal on-line algorithm for metrical task system. *J. ACM*, 39(4):745–763, 1992.

Niv Buchbinder, Kamal Jain, and Joseph Naor. Online primal-dual algorithms for maximizing ad-auctions revenue. In *Algorithms - ESA 2007, 15th Annual European Symposium, Eilat, Israel, October 8-10, 2007, Proceedings*, volume 4698 of *Lecture Notes in Computer Science*, pp. 253–264. Springer, 2007.

CESAR. Center for exascale simulation of advanced reactors (cesar) hpc traces, NA. https://portal.nersc.gov/project/CAL/doe-miniapps.htm.

Kai Chen, Ankit Singla, Atul Singh, Kishore Ramachandran, Lei Xu, Yueping Zhang, Xitao Wen, and Yan Chen. Osa: An optical switching architecture for data center networks with unprecedented flexibility. *IEEE/ACM Transactions on networking*, 22(2):498–511, 2013.

Li Chen, Kai Chen, Zhonghua Zhu, Minlan Yu, George Porter, Chunming Qiao, and Shan Zhong. Enabling wide-spread communications on optical fabric with megaswitch. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pp. 577–593, 2017.

Thomas Erlebach, Murilo Santos de Lima, Nicole Megow, and Jens Schlöter. Learning-augmented query policies for minimum spanning tree with uncertainty. In *30th Annual European Symposium on Algorithms, ESA 2022, September 5-9, 2022, Berlin/Potsdam, Germany*, volume 244 of *LIPIcs*, pp. 49:1–49:18, 2022.

Nathan Farrington, George Porter, Sivasankar Radhakrishnan, Hamid Hajabdolali Bazzaz, Vikram Subramanya, Yeshaiahu Fainman, George Papen, and Amin Vahdat. Helios: a hybrid electrical/optical switch architecture for modular data centers. In *Proceedings of the ACM SIGCOMM 2010 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, New Delhi, India, August 30 -September 3, 2010*, pp. 339–350. ACM, 2010. ISBN 978-1-4503-0201-2. doi:10.1145/1851182.1851223. URL https://doi.org/10.1145/1851182.1851223.

James Hamilton. Architecture for modular data centers, 2006. URL https://arxiv.org/abs/cs/0612110.

Monika Henzinger, Stefan Neumann, and Stefan Schmid. Efficient distributed workload (re-)embedding. In *Abstracts of the 2019 SIGMETRICS/Performance Joint International Conference on Measurement and Modeling of Computer Systems, Phoenix, AZ, USA, June 24-28, 2019*, pp. 43–44. ACM, 2019.

Srikanth Kandula, Sudipta Sengupta, Albert Greenberg, Parveen Patel, and Ronnie Chaiken. The nature of data center traffic: measurements & analysis. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, IMC '09, pp. 202–208, 2009.

Rishi Kapoor, Alex C. Snoeren, Geoffrey M. Voelker, and George Porter. Bullet trains: a study of nic burst behavior at microsecond timescales. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, pp. 133–138. Association for Computing Machinery, 2013.

Harold W. Kuhn. The hungarian method for the assignment problem. In *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*, pp. 29–47. Springer, 2010. ISBN 978-3-540-68274-5. doi:10.1007/978-3-540-68279-0_2. URL https://doi.org/10.1007/978-3-540-68279-0_2.

Janardhan Kulkarni, Stefan Schmid, and Pawel Schmidt. Scheduling opportunistic links in two-tiered reconfigurable datacenters. In *SPAA '21: 33rd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, 6-8 July, 2021*, pp. 318–327. ACM, 2021. ISBN 978-1-4503-8070-6. doi:10.1145/3409964.3461786. URL https://doi.org/10.1145/3409964.3461786.

Wenxue Li, Xiangzhou Liu, Yuxuan Li, Yilun Jin, Han Tian, Zhizhen Zhong, Guyue Liu, Ying Zhang, and Kai Chen. Understanding communication characteristics of distributed training. In *Proceedings of the 8th Asia-Pacific Workshop on Networking*, APNet '24, pp. 1–8, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400717581. doi:10.1145/3663408.3663409. URL https://doi.org/10.1145/3663408.3663409.

He Liu, Matthew K. Mukerjee, Conglong Li, Nicolas Feltman, George Papen, Stefan Savage, Srinivasan Seshan, Geoffrey M. Voelker, David G. Andersen, Michael Kaminsky, George Porter, and Alex C. Snoeren. Scheduling techniques for hybrid circuit/packet networks. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '15, New York, NY, USA, 2015a. Association for Computing Machinery. ISBN 9781450334129. doi:10.1145/2716281.2836126. URL https://doi.org/10.1145/2716281.2836126.

He Liu, Matthew K Mukerjee, Conglong Li, Nicolas Feltman, George Papen, Stefan Savage, Srinivasan Seshan, Geoffrey M Voelker, David G Andersen, Michael Kaminsky, et al. Scheduling techniques for hybrid circuit/packet networks. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, pp. 1–13, 2015b.

T. Lykouris and S. Vassilvitskii. Competitive caching with machine learned advice. *J. ACM*, 68(4): 24:1–24:25, 2021. https://doi.org/10.1145/3447579.

Aranyak Mehta, Amin Saberi, Umesh V. Vazirani, and Vijay V. Vazirani. Adwords and generalized online matching. *J. ACM*, 54(5):22, 2007.

William M. Mellette, Rob McGuinness, Arjun Roy, Alex Forencich, George Papen, Alex C. Snoeren, and George Porter. Rotornet: A scalable, low-complexity, optical datacenter network. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 2017, Los Angeles, CA, USA, August 21-25, 2017*, pp. 267–280. ACM, 2017.

Polatis. Oem solutions - optical switch modules from 16x16 up to 48x48 ports, 2025. URL https://www.polatis.com/switch-modules-for-oem-all-optical-switch-module-solutions-original-equipment-manu asp.

George Porter, Richard Strong, Nathan Farrington, Alex Forencich, Pang Chen-Sun, Tajana Rosing, Yeshaiahu Fainman, George Papen, and Amin Vahdat. Integrating microsecond circuit switching into the data center. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pp. 447–458, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450320566. doi:10.1145/2486001.2486007. URL https://doi.org/10.1145/2486001.2486007.

Leon Poutievski, Omid Mashayekhi, Joon Ong, Arjun Singh, Mukarram Tariq, Rui Wang, Jianan Zhang, Virginia Beauregard, Patrick Conner, Steve Gribble, et al. Jupiter evolving: transforming google's datacenter network via optical circuit switches and software-defined networking. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pp. 66–85, 2022.

Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. *Advances in Neural Information Processing Systems*, 31, 2018.

Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. Inside the social network's (datacenter) network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, SIGCOMM 2015, London, United Kingdom, August 17-21, 2015*, pp. 123–137. ACM, 2015. ISBN 978-1-4503-3542-3. doi:10.1145/2785956.2787472. URL https://doi.org/10.1145/2785956.2787472.

Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, Anand Kanagala, Hong Liu, Jeff Provost, Jason Simmons, Eiichi Tanda, Jim Wanderer, Urs Hölzle, Stephen Stuart, and Amin Vahdat. Jupiter rising: a decade of clos topologies and centralized control in google's datacenter network. *Commun. ACM*, 59(9):88–97, 2016. doi:10.1145/2975159. URL https://doi.org/10.1145/2975159.

Weiyang Wang, Moein Khazraee, Zhizhen Zhong, Manya Ghobadi, Zhihao Jia, Dheevatsa Mudigere, Ying Zhang, and Anthony Kewitsch. Topoopt: Co-optimizing network topology and parallelization strategy for distributed training jobs. In *20th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2023, Boston, MA, April 17-19, 2023*, pp. 739–767, 2023.

Chenyang Xu and Benjamin Moseley. Learning-augmented algorithms for online steiner tree. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022*, pp. 8744–8752, 2022.

A Deferred proofs

A.1 Proof of Theorem 1

Theorem 1. Greedy *is* $6n$*-competitive for* $\lambda = \alpha/n$.

Proof. We refer to Algorithm 1 as Greedy. We call *saturation* the event when Greedy changes its matching, that is, when line 5 is triggered. We call a *phase* a sequence of consecutive requests between two saturations. Let P be a phase, we define the total cost of Greedy on P, $C(P)$ into two costs: $C(P) = C_M(P) + C_{\overline{M}}(P)$ where $C_M(P)$ is the cost of the reconfiguration at the end of P plus the cost of serving the requests for the saturating matching, and where $C_{\overline{M}}(P)$ is the remaining cost for serving requests outside of the saturating matching.
Let Opt be an optimal, offline algorithm. Starting from the beginning of the input request σ, we select consecutive phases $P_1, P_2 \dots P_m$ and stop when either (1) Opt reconfigured it matching (hence Opt reconfigured during P_m) or (2) when we reach n phases, i.e., $m = n$. Once we constructed such a sequence of phases, we repeat this process until the end of the sequence — for simplicity, we will assume that σ is large and that the cost paid during the last sequence of phases (potentially unfinished) is negligible. We will derive the claimed competitive ratio on any (complete) sequence of phases, which implies that the competitive ratio holds on the entire input sequence.

We first note that, for any phase P it holds that $C_M(P) \leq 3\alpha$, as Alg pays at most α for the reconfiguration and at most 2α for the serving costs (α to reach the saturation threshold, plus another α for the rest in the euclidean division by $\frac{\alpha}{n}$ across the edges to be matched).
We will now prove that, in both cases (1) and (2), Opt pays more than $\frac{\alpha}{2} + \sum_{i=1}^{m} C_{\overline{M_i}}(P_i)$ during those phases which implies the desired claim combined with the above statement.

(1) **We assume that Opt reconfigures its matching during the phase P_m.** Take a phase $P_i \in \mathcal{P}$ with $i < m$. Notice that Opt pays more than $C_{\overline{M}}(P_i)$ as Opt does not change its matching and the saturating matching is the one that concentrates the most requests during P_i.
Now during the last phase P_m, Opt pays at least α for the reconfiguration plus $C_{\overline{M}}(P_i)$, giving the claim.

(2) **We assume that Opt never reconfigures its matching during any phase of P.**

We now prove that Opt pays an additional $\frac{\alpha}{2}$ during those phases. The proof is over if Opt rematches at least once during those phases, we therefore assume that Opt does not rematch in the rest of the proof. Let M_{Opt} be Opt's matching, let M_i be Alg's matching once Alg rematched at the end of P_i for some $i \in [n]$. For all $i \in [n]$, we define the function $f(i) = |M_{\text{Opt}} \cap M_i|$. We relate the cost of Opt during a phase P_i with the change of $\Delta f(i) = f(i) - f(i-1)$. We show that if $\Delta f(i) \leq 0$ then Opt pays at least $\max\{\frac{\alpha}{n}, \frac{\Delta f(i)}{2}\frac{\alpha}{n}\}$ during P_i: if f decreases between $i-1$ and i, it means that Alg unmatched edges in common with M_{Opt} and matched other edges instead. Alg can only unmatch an edge e and match another edge e' instead if e' was requested more than $\frac{\alpha}{n}$ times and if e and e' share exactly one node. As a result, if f decreases or stays constant it means that Opt pays at least $\frac{\alpha}{n}$ for an edge. Requests to an edge that is not in M_{Opt} can unmatch maximum two edges that are in M_{Opt}, hence the desired lower bound. By design, $f(i)$ cannot be greater than $n/2$; as a result, f must decrease by one or stay constant at least $n/2$ times and Opt thus pays at least $\frac{\alpha}{2}$ during the n phases.

Putting the previous results together, we proved that Alg is $6n$-competitive. □

A.2 Proof of Theorem 2

Theorem 2. *No randomized (or deterministic) online algorithm for the optical interconnect reconfiguration problem can achieve competitive ratio better than* $n/12$.

Proof. We use Yao's principle to prove the lower bound. We construct a distribution over the input sequences of requests such that the expected cost of the optimal offline algorithm is at least $n/12$ times the expected cost of any deterministic algorithm that has the knowledge of the input distribution,

but not the random choices of the adversary who produces the input sequence. This gives a lower bound of the competitiveness of any randomized algorithm.

Consider the following distribution of input sequences. We partition the nodes into $n/4$ sets $S_1, S_2, S_3, \ldots, S_{n/4}$ of four nodes so that each set $\{a, b, c, d\}$ has the pairs (a, b) and (c, d) matched in the initial configuration.

We issue requests in phases as follows. For each set S_i, we pick a pair of nodes (x, y) from S_i^2 uniformly at random and issue α requests between x and y. We first issue all requests within the set S_1, then within S_2, until $S_{n/4}$. The input sequence consists of m phases, for some large m (the repetition is intended so that the cost of the algorithm cannot be bounded by a constant, as we desire a lower bound on the competitive ratio, regardless of the competitive ratio additive).

Fix any set S_i. We claim that for any of the requests to the nodes of the set S_i any deterministic algorithm pays at least $\frac{\alpha}{3}$ in expectation. The probability that the deterministic algorithm has the requested pair of nodes in the matching at the moment of arrival of the first request to the nodes in S_i in this phase is at most $1/3$: the algorithm has at most two pairs of nodes of S_i in the matching (otherwise the configuration of the algorithm would not be a matching). There are six possible pairs of nodes in S_i that can be requested, and the algorithm has at most two of them in the matching. Thus, the probability that the requested pair is in the matching of the algorithm at the moment of issuing of the first request to S_i in the current phase is at most $2/6 = 1/3$. If the algorithm reconfigures to collocate the requested pair, the cost of the algorithm is at least α (the cost of reconfiguration). If the algorithm does not reconfigure throughout the sequence of requests within S_i, then it pays the expected cost of $1/3$ for each of the α requests to the nodes of S_i, and the claim holds.

Summing over all $n/4$ sets S_i gives the expected cost in the phase of $n \cdot \alpha/12$. In each phase, the optimal offline algorithm pays at most α: at the beginning of the phase, it reconfigures to a configuration that collocates all the pairs requested throughout the phase. Summing over all phases gives us the desired lower bound. □

A.3 Proof of Theorem 3

Theorem 3. *Algorithm 2 is a 4-approximation on any of its phases except the first and the last one. Algorithm 2 (Off) is a 4-approximation taken over the whole input sequence. It can be implemented to run in $O(T \cdot n^3)$ time.*

Proof. We will compare the cost of Off with that of Opt, which is an optimal offline algorithm.

Before we start, since we work with integers we will define $\beta = \lceil \frac{\alpha}{3} \rceil$.

We define a phase as a subsequence of requests between two consecutive reconfigurations by algorithm Off, where the first phase starts with the initial configuration and the last phase ends with the last request.

To start, we prove that the competitive ratio on any phase of Off that is not the first one or the last one is at most 4. Let S be a phase, let M^{Off} be the matching of Off during S and let $w_S^*(e)$ be the weight of edge e at the end of S. Off pays $\alpha + \beta$ during S: α for the reconfiguration at the start and β to serve requests to edges outside of M^{Off}, while the total number of served requests is $w_S^*(M^{\text{Off}}) + \beta$. Now, we consider two cases to show that Opt pays at least $beta$.

- Opt does not reconfigure throughout the phase S and stays in the configuration M^{Opt}. The cost of Opt during the phase is $w_S^*(E \setminus M^{\text{Opt}})$. As M^{Off} is a matching of maximum weight during S, it holds that $w_S^*(M^{\text{Off}}) \geq w_S^*(M^{\text{Opt}})$. From that we derive that $w_S^*(E \setminus M^{\text{Off}}) \leq w_S^*(E \setminus M^{\text{Opt}})$, Opt therefore pays more than $beta$ during S.
- Opt does at least one reconfiguration leading to the cost of at least α, which is larger than β since $\alpha > 1$ as stated in the model.

The above reasoning proves that the competitive ratio of Off is at most 4 on all phases but the first one and the last one. To prove that the competitive ratio of the whole execution does not exceed 4, we

will show that the competitive ratio to serve the requests of the first and the last phases taken together also does not exceed 4. For that we again consider two cases.

- OFF has only one phase. Then, the competitive ratio is 1 since OFF and OPT start with the same configuration and at most $beta$ requests are done outside of the original matching making it unreasonable to make a reconfiguration by OPT for cost α.
- OFF has at least two phases. We consider the first and the last phases together. During these phases OFF pays $beta$ during the first phase and $\alpha + c$ during the last phase for some constant $c \leq \alpha$, where α is paid for the reconfiguration and c is paid for requests outside the matching of OFF during the last phases. OPT pays at least α during the first phase: either it does not reconfigure as OFF or it reconfigures paying at least α. During the last phase, OPT pays at least c if there is no reconfiguration (as the matching of OFF is maximum during that phase) or at least α, otherwise. Thus, OPT pays at least $\alpha + c$ for both phases. Summarizing, OFF pays $\beta + \alpha + c$ while OPT pays at least $\beta + c$ during these two phase giving 4-approximation in total, as $\beta \geq \frac{\alpha}{3}$.

Now, we will discuss the complexity of the algorithm. The trivial analysis can give us $O(T^2 \cdot n^3)$ — at each iteration ($O(T)$ in total) of outer loop we run no more than one iteration of the inner loop (at most $O(T)$), while at each step of the inner loop we calculate the maximum matching in $O(n^3)$ using Hungarian algorithm.

However, this analysis is too loose. On the high level, the inner loop finds the next reconfiguration moment in the future, and the outer loop just serves the requests until that reconfiguration moment.

In more detail, suppose that the algorithm calls the inner loop at request σ_X and the inner loop finds the request σ_Y at which the condition at Line 10 holds. When the outer loop continues it will call the inner loop next time only when the condition at Line 4. However, since conditions at Line 4 and Line 10 are identical, this call will happen on the request σ_Y.

Thus, the algorithm calculates the maximum matching twice per each request: in the inner loop and in the outer loop, providing $O(T \cdot n^3)$ complexity. □

A.4 PROOF OF THEOREM 4. COMPLEXITY $O(T \cdot n^4)$.

Theorem 4 (Consistency and Robustness)**.** PRED *(Algorithm 3) is* 4*-consistent and* $O(n \log n)$*-robust.* PRED *can be implemented to run in time* $O(T \cdot n^4)$.

Proof. At first, we find the maximum weighted matching on each request using the Hungarian algorithm Kuhn (2010) which runs in $O(n^3)$, giving $O(T \cdot n^3)$.

Then, we need to check the condition at Line 13. For that, we will iterate over the size of M' from 1 to n, take all the edges with weight at least $\frac{\alpha}{|M'|} + 1$, and compare the announced size of M' with the size of the maximum matching using Kuhn's algorithm which runs in $O(n^3)$.

This gives us $O(T \cdot n^4)$ complexity.

Note that one can improve it to $O(T \cdot n^3)$ by carefully calculating the time spent on the iterative search of augmented paths in Kuhn's algorithm after adding each set of edges. □

A.5 PROOF OF LEMMA 1

Lemma 1. *Consider a superphase* S *made of phases* $P_1, P_2 \ldots$ *For a given phase* P_i *of* S*, we call* η_i *the error function during* P_i *—* η_0 *is the error function at the beginning of* S*. The series* $\eta_0, \eta_1 \ldots$ *is strictly decreasing.*

Proof. We prove the claim by contradiction: suppose that there exists two consecutive phases P_{i-1} and P_i in S such that $\eta_{i-1} \leq \eta_i$. We start by proving that strictly more than $\alpha/3$ requests fall outside of M^{OFF} during P_{i-1}. As in PRED, we call M' the matching whose edges are rematched at the end of P_{i-1} at line 15 — notice that M' may not cover the whole the graph. We call $B \subseteq M^{\text{OFF}}$ the set of

edges in M^{OFF} that were unmatched from P_{i-1} to P_i, and $G \subseteq M^{\text{OFF}}$ the set of edges in M^{OFF} that were newly matched from P_{i-1} to P_i; it holds that $|G| \leq |B|$ since the error increases: $\eta_{i-1} \leq \eta_i$. Let $D = M' \setminus M^{\text{OFF}}$. Notice that $2|D| \geq |B|$ as a rematched edge unmatches at most two edges. Using the two previous inequalities on the equality $|G| + |D| = |M'|$ (definition of D) gives

$$|G| + |D| = |M'| \implies |D| \geq \frac{|M'|}{3}$$

Using this inequality and the saturation condition of M' on line 13, we obtain that D receives no fewer than $\frac{\alpha}{3}$ requests during P. We now prove that before the superphase S ends, there exists a matching different from M^{OFF} triggering the stopping condition at line 8, a contradiction. M^{OFF} is the maximum matching at the end of S, so there exists a request in $P_i \cup \cdots \cup P_q$ after which all edges of M^{OFF} are in a maximum matching with regard to w^*. Let t be the index of the first such request, $w^*(M^{\text{OFF}}(t))$ be the weight of M^{OFF} at time t and $w^*(E \setminus M^{\text{OFF}}(t))$ be the rest weight. It holds that $w^*(M^{\text{OFF}}(t)) > \frac{\alpha}{3}$, otherwise D would be a matching with weight strictly greater than M^{OFF}'s. Also, it holds that $w^*(E \setminus M^{\text{OFF}}(t)) \geq w^*(D) > \frac{\alpha}{3}$. One request earlier at index $t-1$, a maximum matching had weight at least $w^*(M^{\text{OFF}}(t)) - 1 \geq \frac{\alpha}{3}$ and the rest of the graph had weight at least $w^*(E \setminus M^{\text{OFF}}(t)) - 1 \geq \frac{\alpha}{3}$; the superphase should have ended at request $t-1$, a contradiction. □

A.6 Proof of Lemma 2

Lemma 2. *At the end of any phase P_i, no matching has a total weight $w(.)$ greater than $\alpha \cdot H(n/2)$, where $H(n/2) = \sum_{j=1\ldots n/2} \frac{1}{j}$ is the harmonic sum.*

Proof. Let P_i be a phase and let M be a matching. At the end of P_i, for all $j \in [1, n]$ it holds that no more than j edges of M have weight greater than $\frac{\alpha}{j}$ as this is a stopping condition of the ongoing phase. Among the edges of M, the edge of maximum weight has a weight no greater than α; the one with the second largest weight is no larger than $\frac{\alpha}{2}$, etc. As no matching has more than $n/2$ edges, M has a weight no greater than $\alpha \cdot H(n/2)$. □

A.7 Proof of Theorem 5

Theorem 5 (Smoothness)**.** Pred *is $4 \cdot (1 + \eta \cdot [2 + \ln n])$ - competitive during any superphase that is not the first one (where the predictor was not called) or the last one (since it may be not complete).*

Proof. We call Pred the Algorithm 3, Opt is an optimal offline algorithm, Off is Algorithm 2. We will compare the cost of Pred with that of Off to conclude as Off is 4-competitive (Theorem 3). Given a superphase S, we define M^{OFF} as Off's matching during that superphase, let M^{PRED} be the predicted matching for that superphase. We define the error function for S as $\eta = |M^{\text{OFF}} \setminus M^{\text{PRED}}|$. We call a *phase* the set of requests between two reconfiguration events of Pred, either at line 9 or line 15. Let P be a phase in S. We call Pred_P the *cost* of Pred during P, Pred_P entails all serving costs by Pred during P as well as the final reconfiguration cost at the end of P (if P is not the last phase) — the reconfiguration cost at the beginning of P is accounted for the previous phase. In the rest of the proof, we work on a superphase S which is made of consecutive phases $P_1 \ldots P_q$.

From Lemma 1, we deduce that there are at most $\eta + 1$ phases in S. Let P_i be a phase. We call $w_{P_i}(.)$ the weight of edges counting the number of costly requests during phase P_i. By Lemma 2, a maximum weight matching $w_{P_i}(.)$ at the end of P_i has total weight at most $\alpha H(n/2)$. Off is static, Off thus saves at most $\alpha H(n/2)$ during P_i compared to Pred. Calling Pred_{P_i} the cost of Pred during P_i, we have $\text{Pred}_{P_i} = \alpha + l_i + r_i$, where α is the reconfiguration cost at the end of P_i, $l_i = \min\{\sum_{e \in E} w_{P_i}(e), \alpha H(n/2)\}$ and $r_i = \sum_{e \in E} w_{P_i}(e) - l_i$; in other words, $w_{P_i}(e)$ is the number of requests paid by Pred to edge e during P_i, l_i is the serving cost capped by $\alpha H(n/2)$ and r_i is the remaining serving cost. At last, Off and Pred have the same matching in case there is a phase $P_{\eta+1}$ (Lemma 1), Off therefore pays for $l_{\eta+1} + r_{\eta+1}$ during $P_{\eta+1}$. Overall, Off therefore pays more than $\alpha + [\sum_{i=1}^{\eta} r_i] + l_{\eta+1} + r_{\eta+1}$ during S. Breaking the cost of Pred into reconfiguration and serving costs gives:

$$
\begin{aligned}
\textsc{Pred}_S &\le \alpha \cdot (1+\eta) + \sum\nolimits_{i=1}^{\eta+1} (l_i + r_i) && \text{(Lemma 1: } \le \eta + 1 \text{ phases)} \\
&\le \alpha \cdot (1+\eta) + \alpha\eta H(n/2) + \sum\nolimits_{i=1}^{\eta} r_i + l_{\eta+1} + r_{\eta+1} && \text{(Lemma 2)} \\
&\le \alpha \cdot (1+\eta) + \alpha\eta H(n/2) + \textsc{Off}_S - \alpha \\
&= \alpha\eta + \alpha\eta H(n/2) + \textsc{Off}_S \\
&\le (1 + \eta \cdot [1 + H(n/2)]) \cdot \textsc{Off}_S \\
&\le 4 \cdot (1 + \eta \cdot [2 + \ln n]) \cdot \textsc{Opt}_S. && \text{(Theorem 3, } H(n) \le (\ln n) + 1) \qquad \square
\end{aligned}
$$