CBB-Net: Towards a demand-aware periodic microsecond-switching RDCN for skewed traffic

Thatthep Rukpanich*†, Teeruch Sintusiri*†, Worames Phadungcharoen†
Chantana Chantrapornchai‡, Sucha Supittayapornpong†
†Vidyasirimedhi Institute of Science and Technology, Thailand. ‡Kasetsart University, Thailand.

***Abstract*—Recent advances in microsecond-switching reconfigurable datacenter networks (RDCNs) with spine-level optical circuit switches (OCSes) cyclically reconfigure the topology to emulate a complete graph. In this paper, we show that the complete graph with Valiant Load Balancing (VLB) is inefficient under skewed traffic patterns, which are typical of real-world datacenters, where only a subset of top-of-rack (ToR) switches sends or receives large flows at any given time. To address this, we propose the *complete balanced bipartite (CBB)* graph, which better serves skewed traffic. Based on this, we introduce *CBB-Net*, a demand-aware periodic RDCN that reconfigures network capacity in microseconds to efficiently serve large flows while maintaining an ever-changing connected graph for small flows. CBB-Net exploits graph isomorphism to meet stringent time requirements for topology and routing computation, enabling updates within microseconds. Experiments with real-world traffic distributions reveal that CBB-Net yields up to a 1.79× reduction in the average flow completion time of large flows while achieving similar performance for small flows compared to a state-of-the-art demand-oblivious microsecond-switching RDCN.**

I. Introduction

Achieving low latency in datacenter networks while accommodating growing traffic demands is an active research area in datacenter networking. The trend is shifting away from static networks, relying solely on electrical packet switches [1]–[6], toward reconfigurable datacenter networks (RDCNs) [7]–[22] for efficient resource allocation at lower costs.

The primary objective of RDCN designs is to efficiently serve traffic demands, which span a wide range of flow sizes (e.g., 100 bytes to 1 GB [5]). Previous designs typically classify flows into *small* and *large* categories, as they require different handling to minimize flow completion time (FCT). Small flows are *latency-sensitive* and should be forwarded as quickly as possible, ideally with minimal hop counts. In contrast, large flows are *throughput-hungry* and require high bandwidth to transfer data efficiently from the source to the destination. An effective RDCN design must simultaneously meet the requirements of both flow types.

Various RDCN designs have been proposed in recent years, incorporating different switching technologies with varying reconfiguration times: milliseconds [7], [8], microseconds [11], [13]–[16], and nanoseconds [17]–[19]. The choice of switching technology introduces unique challenges and plays a crucial role in RDCN design. For example, small flows cannot tolerate the high latency introduced by millisecond-range switching delays. As a result, millisecond switches are typically deployed alongside a static topology dedicated to small flows. However, such designs inherently lack *capacity flexibility*, as only a portion of the network capacity can be dynamically reconfigured to accommodate large flows, while the rest remains fixed for small flows or unstable traffic. At the other end, nanosecond switching delays are negligible compared to the latency of small flows. This enables nanosecond switches to support a unified topology for both flow types at the cost of specialized high-end hardware [17], [19].

Microsecond-switching RDCNs present unique challenges: while they can reconfigure the network more frequently than millisecond-switching designs, their microsecond-scale delays remain significant for small flows. To mitigate this, many designs allocate a fixed portion of network capacity to form a static topology dedicated to small flows [13], [14], but this approach limits capacity flexibility. Opera [16] and its realizable version [23] overcome this limitation by cyclically reconfiguring the network to emulate a complete graph, leveraging Valiant Load Balancing (VLB) to provide high throughput for large flows. At any given time, the union of connectivity across all switches forms a connected graph to serve small flows. This design effectively balances the needs of both flow types while maintaining capacity flexibility. However, prioritizing low system complexity, demand-oblivious RDCN has suboptimal throughput, particularly under skewed traffic patterns, which are common in production datacenter networks [13], [24].

Our research direction aims to address the following question: **Under the constraint of microsecond switching, can we design a demand-aware RDCN that satisfies the requirements of both small and large flows while also achieving capacity flexibility?** We take an approach by extending the idea of Opera. In particular, instead of repeatedly emulating a complete graph, can we adaptively emulate a demand-aware graph with low overhead to provide higher throughput?

Our approach: We propose CBB-Net, a demand-aware periodic microsecond-switching RDCN. Our approach is as follows: (i) Instead of a complete graph, CBB-Net relies on a *complete balanced bipartite* (CBB) graph, which is more bandwidth-efficient (§III). CBB-Net periodically detects traffic demands with low overhead (§V-B) and forms a demand-aware CBB graph to provide high throughput for large flows (§IV-C). (ii) Concurrently, CBB-Net provides an always-on connected

*Both authors contributed equally to this paper.

This work was supported by Thailand Science Research and Innovation (TSRI) under Grant No. FRB690039/0457.
ISBN 978-3-903176-82-9 © 2026 IFIP

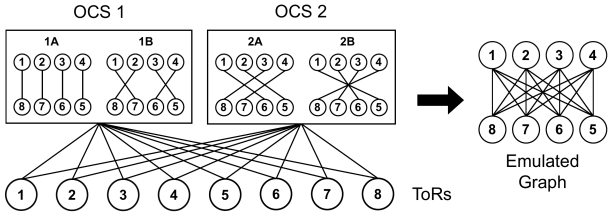


FIGURE 1: A periodic RDCN with 8 ToRs and 2 OCSes. Every OCS rotates over 2 matchings and collectively forms an emulated graph.

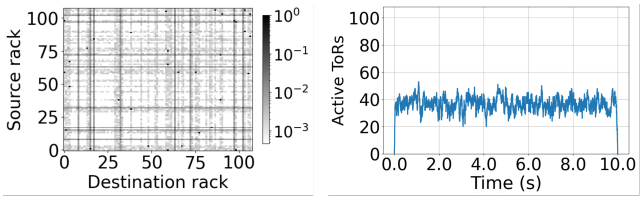


FIGURE 2: (a) Recreated Microsoft traffic pattern. (b) Number of active ToRs simulated from 3:1 Fattree.

graph to promptly handle small flows (§IV-D). (iii) CBB-Net leverages the isomorphism property inherent in a CBB graph to rapidly compute new topology and routing (§IV-E). The contributions of this paper are threefold.

- We analyze network behavior under skewed, real-world traffic and show that instantaneous traffic deviates significantly from all-to-all patterns (§II). To address this, we propose the CBB graph and demonstrate its improved throughput (§III).
- We propose CBB-Net, a demand-aware periodic RDCN based on the CBB graph that reconfigures its topology to meet traffic demands at microsecond timescales.
- We leverage the isomorphism property of the CBB graph to maintain low overhead of topology and routing changes.

We evaluate CBB-Net using a packet-level simulator. The results demonstrate up to a 1.79× reduction in the average flow completion time of large flows compared to a complete-graph-based RDCN, while preserving comparable performance for small flows (§VI-B). Moreover, we will release the source code of this work to research community at: https://github.com/NDS-VISTEC/htsim-cbbnet.

II. Background and Motivation

A. Background on Periodic RDCNs

A periodic RDCN with leaf-spine architecture consisting of ToRs and OCSes is shown in Fig. 1. Every ToR has an uplink connection to every OCS. At any given time, each OCS provides bidirectional connectivity between specific ToR pairs, forming a (ToR-to-ToR) *matching*. Every OCS reconfigures cyclically, following a predefined schedule of its matchings, and each matching spends an equal duration of time, hence the term *periodic*. This cyclical reconfiguration of matchings collectively forms an *emulated graph*.

For example, Opera [16] is a periodic RDCN that emulates a complete graph[1]. To achieve this, Opera decomposes a complete graph into a set of disjoint matchings, which are equally distributed across OCSes. By reconfiguring every OCS according to its assigned matchings, Opera emulates a complete graph and completes one cycle within milliseconds.

For throughput analysis, while it is challenging to analyze the throughput of periodic RDCNs, previous work [22] has proven that *a periodic RDCN whose topology periodically changes over time and its emulated graph have the same throughput.*

[1]We say that a periodic RDCN *emulates* a graph if the union of all matchings over one reconfiguration cycle induces the graph.

B. Traffic-related definitions

This section defines terms used to describe the network characteristics under instantaneous traffic demand.

ToR-level traffic. At datacenter scale, we consider *ToR-level traffic*, where ToRs act as traffic sources and destinations. A ToR sources traffic when at least one of its connected hosts generates traffic. Similarly, a ToR is considered a traffic destination when one of its hosts receives traffic. Traffic sourced from and destined to hosts connected to the same ToR does not contribute to ToR-level traffic.

Large and small flows. Flows are classified as large or small based on their size. This approach has been used in prior RDCN designs [14]–[16] to treat flows differently as large flows are throughput-bound and small flows are latency-bound. In our configuration, the flow-size cutoff is 60 MB. In practice, flows can be classified by marking based on latency requirements per socket [23].

Active ToR. At any instant, a ToR is called an *active ToR* if it is a source or a destination of any large flows. For example, a large flow from ToR A to ToR B results in two active ToRs. Concurrent large flows from ToR A to ToR B and from ToR A to ToR C induce three active ToRs. In particular, active ToRs identify endpoints in a network fabric that need high-capacity connectivity at any point in time.

C. Motivation I: Skewed traffic induces low active ToRs

We motivate this work from the traffic characteristics of the Microsoft clusters, which are highly skewed [13]. Figure 2a shows a recreated 5-minute ToR-level traffic heat map for a cluster with hundreds of ToRs (original in [13]). They also report that *46%–99% of rack pairs exchange no traffic, while only 0.04%–0.3% account for 80% of the total traffic*. In the heat map, pitch-black dots represent dominant (*primary*) traffic forming a permutation pattern, while vertical and horizontal lines indicate *secondary* one-to-many and many-to-one traffic. The map is plotted on a logarithmic scale. We then simulate this traffic to study active ToR behavior.

We simulate the traffic on a 3:1 Fattree network with 108 ToRs using htsim [25]. Each link capacity in the simulation is set to 40 Gbps. In combination with the recreated traffic pattern in Fig. 2a., we use the Hadoop flow size distribution [26] to generate flows that match the application running in the reported cluster [13]. The traffic load is set to almost saturate the highest utilized link in the 3:1 Fat-tree network.

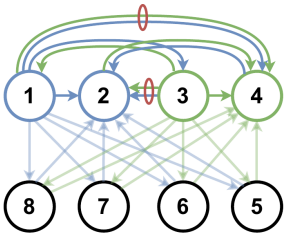


FIGURE 3: A periodic RDCN with eight ToRs emulating a complete graph. Skewed traffic with flows (1⇒2) (blue) and (3⇒4) (green) is uniformly load-balanced; some paths are faded for clarity.

Active ToRs under recreated Microsoft traffic. The heat map in Fig. 2a. suggests a high fan-in/fan-out traffic pattern, indicating that a communication-wide network for arbitrary ToR pairs seems mandatory. However, our simulation results in Fig. 2b reveal that the average number of active ToRs is less than half throughout the simulation period. This suggests that *provisioning high bandwidth for all ToRs simultaneously may be unnecessary, as only a subset of ToRs is active at any given moment.* Next, we depict the inefficiency of a complete graph under skewed traffic.

D. Motivation II: Inefficiency of a complete graph under skewed traffic

In RDCNs based on an emulated complete graph (e.g., Opera), traffic from a source ToR is load-balanced across all other ToRs. Each ToR acts as an intermediary, forwarding received traffic to the ultimate destination unless the destination is the ToR itself. This design, however, results in *throughput degradation* under skewed traffic, as discussed below.

Notation. A flow sourced from ToR u and destined to ToR v is denoted by (u⇒v). A directed link from ToR u to ToR v is denoted by (u→v). A path from ToR u to ToR v to ToR w is denoted by [u → v → w]. In this section, each link has one unit capacity.

Throughput degradation. We analyze the throughput of a periodic RDCN using its emulated graph, as discussed in (§II-A). Fig. 3 shows a toy example in which concurrent flows (1⇒2) and (3⇒4) are load-balanced in the network. Two bottlenecks occur on links (1→4) and (3→2); on each link, the capacity is shared equally between the two flows.

We now consider flow (1⇒2). The bottleneck on link (3→2) limits the traffic rate of the flow that takes the path [1 → 3 → 2] to half the capacity of link (1→3), leaving half of the link capacity unused. The bottleneck on link (1→4) similarly causes the flow to utilize only half the capacity. Overall, ToR 1 sends traffic at line rate to ToRs 2, 5, 6, 7, 8 and at half rate to ToRs 3 and 4, resulting in a total sending rate of 6.

Throughput efficiency. We define the *throughput efficiency* $\theta_{A\Rightarrow B}$ of flow (A⇒B) as the ratio of its total sending rate to the maximum possible sending rate, which equals the aggregate capacity of ToR A's outgoing links. Therefore, in this toy example, the throughput efficiencies are $\theta_{1\Rightarrow 2} = 6/7$ and $\theta_{3\Rightarrow 4} = 6/7$, as each flow forwards 6 units of traffic over all 7 outgoing links. Our further analysis, omitted due to space constraints, reveals that this throughput degradation persists proportionally as the network scales.

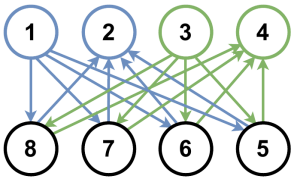


FIGURE 4: A CBB graph provides optimal throughput efficiency when active ToRs fit in one half and use two-hop forwarding.

E. Key Takeaway

Under skewed traffic, only a subset of ToRs is active at any instant. Combining this insight with the inefficiency of uniform load balancing over a complete graph, shown in Fig. 3, raises the question of whether there is a more efficient way to serve concurrent flows. We therefore propose a complete balanced bipartite (CBB) graph in the next section.

III. CBB GRAPH

A. Definition and intuition

Definition. A CBB graph is denoted by $CBB = G(\mathcal{U}, \mathcal{L})$. The nodes of a CBB graph are partitioned equally into two sets: *upper half* and *lower half*, denoted by $\mathcal{U}$ and $\mathcal{L}$ respectively. Edges only exist between every pair of nodes in different halves, resulting in a *complete balanced bipartite* graph. For example, the emulated graph in Fig. 1 is CBB with $\mathcal{U} = \{1, 2, 3, 4\}$ and $\mathcal{L} = \{5, 6, 7, 8\}$. The nodes represent ToRs in the network, while the edges represent logical ToR-to-ToR connectivity.

Intuition. From the complete graph in Fig. 3, we observe that (a) a bottleneck occurs on a link from the source of one flow to the destination of another flow, and (b) forwarding traffic to idle ToRs causes no bottlenecks. To address this, we introduce a CBB graph. Traffic in a CBB graph relies exclusively on two-hop forwarding. A source ToR evenly distributes traffic to all adjacent ToRs in the other half, which then forward the received traffic to a destination ToR in the same half as the source. This design removes the bottleneck in (a) and exploits the opportunity in (b).

Example. Fig. 4 shows an ideal ToR placement, in which all active ToRs are placed in one half, while the other half contains only idle ToRs. All flows sourced from the active ToRs are evenly distributed to idle ToRs in the other half, thereby causing no bottlenecks. In particular, both flows achieve optimal throughput efficiency, $\theta_{1\Rightarrow 2} = \theta_{3\Rightarrow 4} = 1$, as they send at line rate on all outgoing links.

When active ToRs cannot fit in a single half, they are placed across both halves while maximizing the number of active ToRs in one half. In this case, bottlenecks occur in a CBB graph but are less severe than those in a complete graph. Note that we do not place the source and destination of a flow in different halves, since the flow will be served by a single link under one-hop forwarding, severely impacting throughput. This placement intuition is formalized in (§IV-C) to reduce throughput degradation for an arbitrary set of active ToRs.

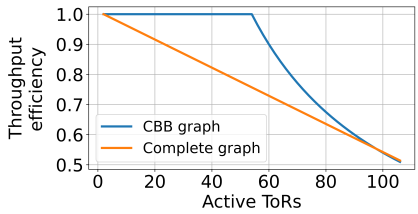
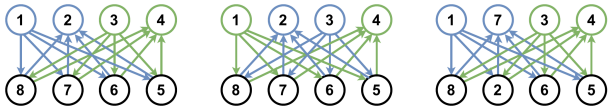


FIGURE 5: Throughput efficiency curve under CBB and complete graphs as active ToRs increases in a 108-ToR network.



(a) Concurrent flows 1 $\{(1 \Rightarrow 2), (3 \Rightarrow 4)\}$ (b) Concurrent flows 2 $\{(1 \Rightarrow 4), (3 \Rightarrow 2)\}$ (c) Concurrent flows 3 $\{(1 \Rightarrow 7), (3 \Rightarrow 4)\}$

FIGURE 6: Three CBB graphs delivering optimal throughput for the three sets of concurrent flows using two-hop forwarding.

B. Advantages of CBB graphs

Higher throughput efficiency. Fig. 5 compares the throughput efficiency of the CBB and complete graphs under permutation traffic, a challenging pattern observed in Microsoft clusters [13]. We compute throughput efficiency using a multi-commodity flow formulation using Gurobi [27]. With strategic placement (§IV-C), the CBB graph achieves full efficiency when at most half of the ToRs are active; as more ToRs become active, efficiency degrades and converges to that of the complete graph when all ToRs are active. Since skewed traffic typically activates only a subset of ToRs, these results highlight the potential of CBB-based RDCNs.

Shorter cycle time. A complete-graph–based RDCN requires N matchings to emulate a complete graph over N nodes. In contrast, a CBB graph can be decomposed into $N/2$ matchings, halving the cycle time needed to emulate the target graph. This shorter cycle enables direct connections to become available sooner, allowing the final packets of a flow to complete earlier and reducing flow completion time.

Isomorphism. This property allows CBB graphs to adapt efficiently to changing flow demands. For example, when the flow set changes from {(1⇒2), (3⇒4)} to {(1⇒7), (3⇒4)} (from Fig. 6a to Fig. 6c), the resulting CBB graph is obtained by simply swapping nodes 2 and 7, as the two graphs are isomorphic. This greatly reduces the computational complexity of recomputing matchings and routing paths (§IV-E).

IV. CBB-Net design

This section presents the design of CBB-Net, an RDCN that emulates CBB graphs to efficiently serve skewed demands.

A. CBB-Net overview

Architecture. Our CBB-Net is a leaf-spine network consisting of N ToRs and S OCSes as shown in Fig. 8. The uplinks of each ToR connect to every OCS, while the downlinks connect to $H = S$ hosts. Traffic information for large flows is periodically collected from ToRs to construct a demand-aware CBB graph. Each OCS reconfigures at microsecond timescales following

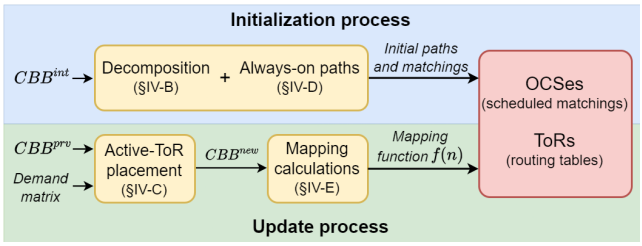


FIGURE 7: Initialization and update processes of CBB-Net.

controller-determined matchings, emulating a CBB graph over time. CBB-Net is *initialized* once and periodically *updated* based on real-time traffic information. An overview of the initialization and update processes is shown in Fig. 7.

Initialization process. CBB-Net pre-computes the sets of matchings and the sets of paths corresponding to the initial emulated CBB graph (CBB^{int}), whose upper half contains the first half of ToR indices. To achieve this, CBB^{int} is decomposed and organized into sets of matchings, each set is assigned to an OCS, to ensure that the union of OCS-deployed matchings at any given time always forms a connected graph capable of delivering small flows (§IV-B). Furthermore, the initial sets of routing paths for small flows are pre-computed (§IV-D).

Update process. Once the central controller receives traffic information from the ToRs and demand matrix changes, it solves the *active ToR placement problem* (§IV-C) to generate a demand-aware CBB graph (CBB^{new}) that serves the demand with higher throughput. Using the isomorphism property of a CBB graph, we design the *mapping function* (§IV-E) that maps the structure of the new CBB to the initial CBB graph, enabling the rapid updates of the matchings in OCSes and routing tables in ToRs within microseconds without any recalculation.

B. Constructing CBB matchings

A CBB graph is a d-regular graph with degree $d = N/2$. To deploy it in CBB-Net, we decompose the CBB graph into d disjoint matchings, as illustrated in Fig. 8 with $d = 8/2 = 4$. Each OCS disjointly stores d/S of these matchings. Constructing these matchings is a combinatorial problem and differs from [15] for a complete graph. We, therefore, propose a new decomposition method.

The key idea is to iteratively extract a matching from the d-regular graph by solving an integer linear program until the graph is empty. We solve the problem using Gurobi [27]. The matchings are then shuffled so the union of consecutive matchings of certain lengths forms a connected graph and assigned to OCSes in a round-robin fashion, ensuring the connected-graph property is preserved during CBB-Net operation.

C. Determining a demand-aware CBB graph for large flows

A demand-aware CBB graph is constructed to efficiently serve concurrent large flows. These flows are modeled by a binary *demand matrix* that is a simple $N \times N$ matrix whose entry is 1 if there exists a large flow from ToR i to ToR j and 0 otherwise. Note that this demand matrix also identifies active ToRs associated with the flows. The efficient collection of such

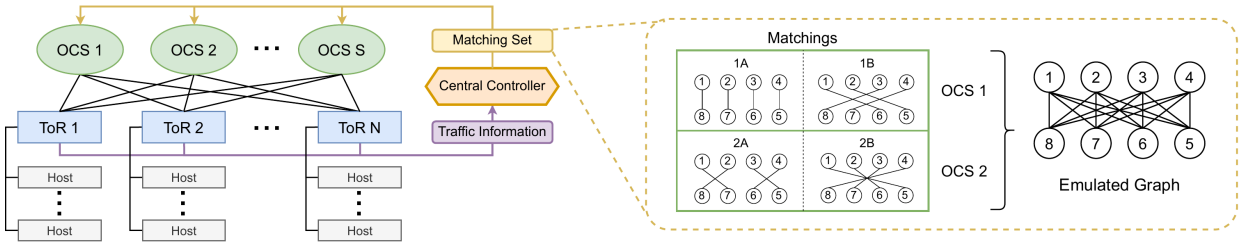


FIGURE 8: CBB-Net with 8 ToRs, 2 OCSes, and 2 hosts per ToR ($N = 8, S = 2, H = 2$). Every ToR connects to every OCS. The central controller receives the traffic information from ToRs regularly. It reassigns new matchings to the OCSes when the demand matrix changes.

demand matrices is detailed in (§V-B). We next construct a demand-aware CBB graph as an active-ToR placement problem.

1) Active-ToR placement problem: Given a demand matrix, a demand-aware CBB graph can be constructed by placing the active ToRs, associated with the demand matrix, on a CBB graph strategically. We define three placement rules based on the insights from (§III):

(R1) Two ends of a demand entry must be on the same half.
(R2) All active ToRs should be on one half if possible.
(R3) A placement should closely resemble the previous one.

Rule (R1) ensures the demand's source can load balance traffic to all intermediate ToRs in the other half under two-hop forwarding. Rule (R2) reduces congestion by ensuring that intermediate ToRs are idle (not the source or destination of any demands) as much as possible. Rule (R3) aims to minimize topology changes between cycles to reduce rerouting overhead, as discussed further in (§IV-C2).

We cast the active-ToR placement as a well-known Knapsack problem, where *items* are selected to maximize their total *value*, constrained by their total *weight* and the *Knapsack capacity*. In our setting, the Knapsack capacity is the size of a CBB half that is $N/2$. Each item is a *connected component*, a group of demand entries sharing source or destination ToRs. For example, a set of ToR-to-ToR demands $\{(1{\Rightarrow}2), (3{\Rightarrow}4), (3{\Rightarrow}5)\}$ induces a set of connected components $\{\{1, 2\}, \{3, 4, 5\}\}$. Intuitively, each connected component must be placed in the same half to satisfy (R1). The item's weight is the size of the corresponding connected component. Clearly, one cannot select connected components whose total size exceeds a CBB half. The last step is to carefully define the value of each item.

Since (R3) suggests the similarity between the placement and a previous CBB graph, we assign the value of each item based on this similarity. Let $CBB^{\text{prv}} = G(\mathcal{U}^{\text{prv}}, \mathcal{L}^{\text{prv}})$ be a previous placement with upper half $\mathcal{U}^{\text{prv}}$ and lower half $\mathcal{L}^{\text{prv}}$. The similarity of a connected component c equals how it overlaps with these halves, which are $|c \cap \mathcal{U}^{\text{prv}}| + 1$ and $|c \cap \mathcal{L}^{\text{prv}}| + 1$. From the two sets of similarity values, we pick the set that has the highest total sum of its similarity values.

Indeed, our Knapsack problem determines the placement by treating connected components as items, where the goal is to pack them into a half, while maximizing alignment with the previous placement. Note that (R1)-(R3) are taken into account in this problem. Solving the Knapsack problem gives a new CBB graph for a given demand matrix. Nonetheless, the placement process could be infeasible when some connected component is larger than a half. This situation is more likely to occur under dense and uniformly random traffic patterns. In this situation, CBB-Net emulates a complete graph, which suits such traffic conditions.

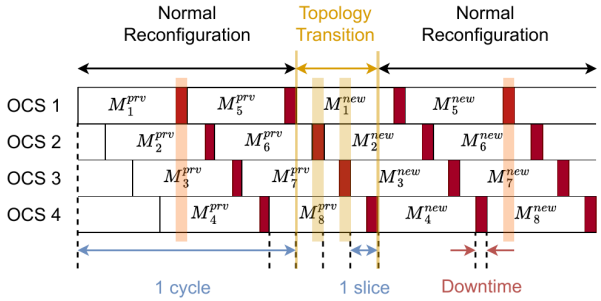


FIGURE 9: Topology transition from "previous" to "new" sets of matchings while performing offset reconfiguration. The M_i^{prv} and M_i^{new} are the i-th matching before and after transition, respectively.

2) Serving of large flows: CBB-Net serves large flows using the RotorLB protocol [15], [16], which employs Valiant Load Balancing (VLB) over two-hop paths. The key distinction of CBB-Net is that large flows are load-balanced over demand-aware CBB graphs, rather than a single complete graph as in [16]. As a result, few packets need to be rerouted after transitions to a new CBB graph.

Specifically, in the RotorLB protocol, each host maintains *local queues* for packets of large flows it originates and *non-local queues* for relayed packets. After a transition, some non-local packets may become stuck if no direct connection to their destination exists, since RotorLB forwards them only over direct links. These packets are moved to the front of the local queue, enabling two-hop forwarding. Our evaluation in (§VI-D) confirms that the affected fraction is negligible.

D. Ensuring always-on connected graph for small flows

Each OCS experiences a *downtime* (10 μs for CBB-Net) during reconfiguration, which can severely affect latency-sensitive small flows if all OCSes reconfigure simultaneously.

1) Offset reconfiguration: To prevent this, CBB-Net employs offset reconfiguration similar to Opera [16], as shown in Fig. 9, where OCSes reconfigure one at a time at a fixed interval, called a *slice*[2]. This introduces additional challenges for CBB-Net, which regularly transitions to new demand-aware graphs.

[2]A slice is the time between two consecutive OCS downtimes, excluding the first; a cycle is the time for one OCS to traverse its allocated matchings.

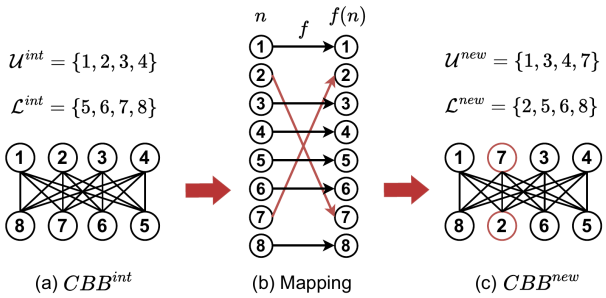


FIGURE 10: Isomorphism of two different CBB topologies: we can remap nodes ($2 \leftrightarrow 7$) to transform (a) to (c) and vice versa.

During *normal reconfiguration*, OCSes cycle through matchings to emulate the current demand-aware graph. When switching graphs (e.g., $CBB^{\text{prv}} \rightarrow CBB^{\text{new}}$), the network enters a *topology transition* phase, during which the topology is derived from two sets of matchings. To ensure always-on paths for small flows, our decomposition (§IV-B) guarantees that: (i) any $S-1$ consecutive matchings (excluding the reconfiguring switch) form a connected graph (orange stripes in Fig. 9); and (ii) First (and last) $S/2$ consecutive matchings form a connected graph (yellow stripes in Fig. 9). (i) ensures connectivity during normal reconfiguration, and (ii) during transitions.

2) Routing of small flows: Small flows are handled by the NDP protocol [28], which uses source routing, designated by the source's NIC, over ECMP multi-paths. ECMP paths are determined *per slice* and are precomputed during the initialization process. To accommodate ever-changing CBB graphs, the paths for a new CBB graph are quickly remapped from the initial CBB graph, as detailed in (§IV-E).

E. Fast topology and routing recalculation

Once a new CBB graph is determined, the entire matching set and routing paths must be updated. We can utilize the isomorphism property of CBB graphs (§III-B) to remap the matchings and routing paths quickly within microseconds.

Mapping function. Let CBB^{new} be a new CBB graph from the placement process (§IV-C), and recall that CBB^{int} is the initial CBB graph from the decomposition (§IV-B). We can construct a *mapping function* $f(n)$ that maps nodes in the new graph to nodes in the initial graph as follows. We first calculate the difference between the new and initial CBB graphs as $\partial\mathcal{U} = \mathcal{U}^{\text{new}} - \mathcal{U}^{\text{int}}$ and $\partial\mathcal{L} = \mathcal{L}^{\text{new}} - \mathcal{L}^{\text{int}}$ for the upper and lower halves respectively. These differences identify the indices that need remapping. We then remap each element in the differences one by one. For the example in Fig. 10, the differences are $\partial\mathcal{U} = \{7\}$ and $\partial\mathcal{L} = \{2\}$. We then remap $f(2) = 7$ and $f(7) = 2$, producing the new mapping function in Fig. 10b. This mapping function is used next to generate new matchings and routing paths.

Fast matching remapping. For every initial matching $M^{\text{int}} = [m_{ij}^{\text{int}}]_{i=1,j=1}^{i=N,j=N}$, we derive a new matching corresponding to a new CBB graph by applying the mapping function f as

$$M^{\text{new}} = [m_{ij} : m_{ij} = m^{\text{int}}_{f^{-1}(i)f^{-1}(j)}, \forall i,j \in \{1,2,\ldots,N\}].$$

This technique enables us to obtain the set of matchings for a new CBB graph quickly without unnecessary decomposition.

Fast path remapping. After changing to a new CBB graph, paths must be updated to reflect the new always-on connected graphs. Instead of recomputing these paths, we quickly derive them from the initial paths associated to the initial CBB graph.

Let $P^{\text{int}}(s,d) = \left[x_1^{sd}, x_2^{sd}, \ldots\right]$ be a path from node s to node d associated with the initial CBB graph where x_k^{sd} is the k^{th} node on the path. The mapping function f associated with a new CBB graph is used to derive the new path as follows. Define a mapping operation $F([x_1, x_2, \ldots]) = \left[f^{-1}(x_1), f^{-1}(x_2), \ldots\right]$. The new path associated with a new CBB graph is derived quickly for every (s,d) by

$$\begin{aligned} P^{\text{new}}(s,d) &= F(P^{\text{int}}(f(s), f(d))) \\ &= \left[f^{-1}(x_1^{f(s)f(d)}), f^{-1}(x_2^{f(s)f(d)}), \ldots\right]. \end{aligned}$$

For example, in Fig. 10, path $P^{\text{new}}(1,7)$ in CBB^{new} is derived from $P^{\text{int}}(1,2) = [1,7,2]$ in CBB^{int} and the mapping f:

$$\begin{aligned} P^{\text{new}}(1,7) &= F(P^{\text{int}}(f(1), f(7))) = F(P^{\text{int}}(1,2)) = F([1,7,2]) \\ &= \left[f^{-1}(1), f^{-1}(7), f^{-1}(2)\right] = [1,2,7]\,. \end{aligned}$$

V. Practical Considerations

A. Hardware enablers

CBB-Net reconfigures topology at microsecond granularity. Various switching technologies with reconfiguration times ranging from microseconds to nanoseconds, such as the Mordia prototype [11], MEMS-based switch [29], and Sirius prototype with AWGR tunable laser [17], could be applied.

B. Large-flow demand collection

The controller computes the topology from a binary ToR-to-ToR demand matrix. Each host encodes its demands using one bit per ToR ($N/8$ bytes for N ToRs) and sends these bits to its local ToR [16], which aggregates and forwards the data to the controller with highest priority.

C. Fault tolerance

Network components failure. CBB-Net uses hello packets, similar to Opera [16]. Adjacent ToRs exchange the packets to detect failures. Missing such a packet, a ToR recomputes its routing table to bypass the failure ToR.

Controller failure. CBB-Net adopts the MegaSwitch [14] approach, using multiple controllers with one designated leader. If the leader fails, a backup controller assumes leadership.

Demand information packet loss. CBB-Net may be affected by temporary inconsistencies between current demands and the controller's demand matrix due to demand information packet loss; however, the impact is minimal since demand updates are sent regularly (§VI-C2).

Mapping function packet loss. Since the mapping function (§IV-E) is critical, mapping packets are sent reliably. If a ToR does not receive a mapping packet within a timeout, it sends a NACK to the controller to request retransmission.

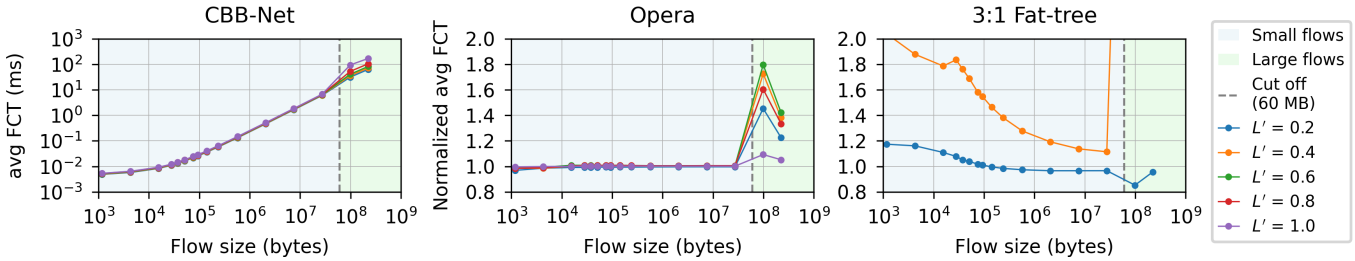


FIGURE 11: Avg FCT of each Hadoop flow size under Microsoft traffic pattern (Microsoft(0.5)).

VI. EVALUATION

We evaluate CBB-Net using the htsim packet-level simulator, which is widely used in RDCN research [16], [22], [28].

A. Experimental setup

1) Network systems: Since our focus is on microsecond-switching RDCNs, we compare CBB-Net with Opera [16], an over-subscribed Fat-tree, and MegaSwitch [14], which represent a demand-oblivious RDCN, a static network, and a demand-aware RDCN, respectively.

CBB-Net. Our network comprises 648 hosts across 108 ToRs (6 hosts per ToR) and 6 OCSes. It reconfigures ToR-to-ToR connectivity to emulate demand-aware CBB graphs, and falls back to a complete graph when no feasible CBB graph exists.

Opera. This is a demand-oblivious RDCN that emulates a complete graph with an ever-changing expander topology. It uses the same configuration as CBB-Net (108 ToRs, 6 hosts per ToR, and 6 OCSes).

Over-subscribed Fat-tree. We use a 3:1 over-subscribed Fat-tree with 648 hosts, where each ToR has 3 uplinks to the upper layer and each rack contains 9 hosts. This network is shown to be cost-equivalent in [16].

MegaSwitch. This is a classical demand-aware network: large flows use demand-aware one-hop circuits, while small or unstable flows traverse a fixed-capacity *basemesh*. Our setup includes 108 ToRs (6 hosts per ToR) each with 6 wavelengths; one wavelength forms a directed circuit per ToR, b wavelengths are reserved for the basemesh, and the remainder support demand-aware circuits. Scheduling takes 10 ms.

All systems use 40 Gbps links and employ priority queues for small flows. For Opera and CBB-Net, the reconfiguration time is set to 10 μs, and each slice lasts 70.6 μs, following Opera's topology generation code. A full cycle takes 7.6 ms for a complete graph (108 matchings) and 3.8 ms for a CBB graph (54 matchings), significantly boosting CBB-Net 's performance (§VI-B1). CBB-Net detects demand every two cycles (7.6 ms).

2) Traffic: Traffic is simulated as follows.

Traffic load. We define the traffic load as $L = \frac{Fn}{TNHr}$, where F is the average flow size, n is the total number of flows generated over period T, $N \times H$ is the total number of hosts, and r is the host link capacity. Intuitively, L captures the ratio between the average flow generation rate and the aggregate host link capacity. Note that even at low load, congestion may still arise at some ToRs under highly skewed traffic.

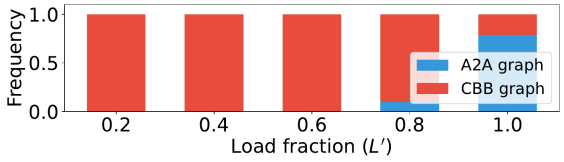


FIGURE 12: Frequency of A2A and CBB graph usage by CBB-Net under varying load fractions.

In the evaluation, we increase L in 2% increments until reaching L_{max}, the maximum load at which the Opera network remains stable. We then vary the *load fraction* $L' = \frac{L}{L_{max}}$.

Flow generation. Flows arrive according to a Poisson process, with arrival rates determined by the target load L. Flow sizes follow real-world Datamining [5] and Hadoop [26] distributions, which are widely used for evaluation in the literature [16], [18], [19], [22]. In both distributions, large flows account for over 90% of the total traffic volume. Each simulation runs for 10 seconds and is repeated five times with different traffic traces to reduce randomness.

Traffic pattern. A traffic pattern characterizes the long-term average traffic volume between source–destination ToR pairs. We generate traffic using a skewed model (§II) that combines *primary* (permutation) and *secondary* (one-to-many/many-to-one) traffic. The resulting pattern, denoted Microsoft(γ), uses γ to control the volume ratio between primary and secondary traffic (default $\gamma = 0.5$). This produces highly skewed ToR-to-ToR traffic. We also evaluated a skewed traffic pattern from [4] and observed similar trends (omitted due to space limit).

B. Performance comparison

Metric. We use average flow completion time (avg FCT) as the performance metric [4], [13], as it reflects network throughput and captures the performance of all flows. Moreover, avg FCT is more consistent across random traffic traces, whereas the 99th-percentile FCT varies significantly. For CBB-Net, we plot the avg FCT for each flow size. For other networks, we plot the avg FCT normalized to that of CBB-Net.

Results. The results for the Microsoft(0.5) traffic pattern with the Hadoop flow size distribution are shown in Fig. 11. The results for the Datamining flow size distribution exhibit a similar trend and are omitted due to the space limit.

1) CBB-Net vs Opera: Fig. 11 shows that, for highly skewed traffic like that from the Microsoft clusters, CBB-Net achieves up to a 1.79× reduction in avg FCT (at $L' = 0.6$)

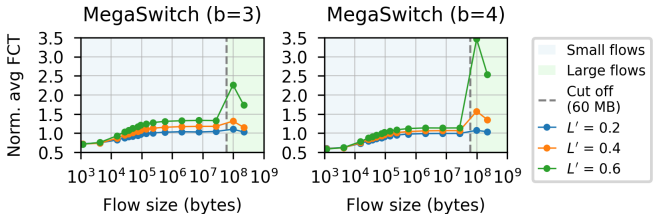


FIGURE 13: Norm. avg FCT of MegaSwitch (Microsoft(0.5)).

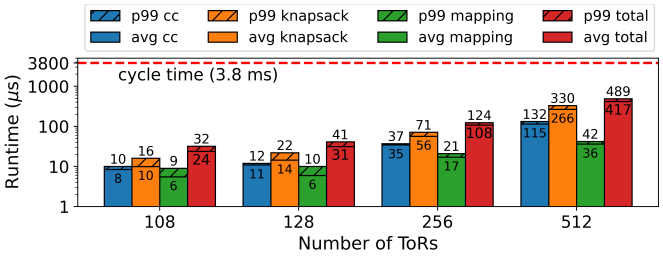


FIGURE 14: Average and p99 execution time of processes at different network sizes. cc: forming connected component, knapsack: solving the Knapsack problem, mapping: calculate mapping function.

while maintaining a similar avg FCT for small flows. This performance gain stems from two key properties of CBB-Net: *(i) Improved emulated graph:* A CBB graph provides higher average throughput than a complete graph, as shown in (§III). *(ii) Shorter cycle time:* Opera requires twice as long to complete a cycle, introducing a 2× delay when packets are forwarded to or collected from intermediaries, affecting both the start and the end of flow delivery. The FCT improvement for smaller large flows is more significant because they benefit proportionally more from (ii). The 99th percentile performance follows a similar trend, with a smaller improvement of up to 1.34×.

When $L' = 1.0$, which is the saturation point, CBB-Net mostly falls back to Opera, shown in Fig. 12, as higher loads activate more ToRs and lead to infeasible placements. We believe that the fallback region can be reduced by modifying the placement rules to support currently infeasible placements; we leave this exploration to future work.

Further analysis shows that CBB-Net achieves its greatest advantage over Opera under highly skewed traffic. When $\gamma > 0.5$ (more skewed traffic), fewer ToRs are active, allowing CBB-Net to outperform Opera across all loads. In contrast, when $\gamma < 0.5$ (less skewed traffic), the traffic becomes closer to all-to-all and triggers an earlier fallback to Opera, although CBB-Net still achieves lower FCT before the fallback. This emphasizes that CBB-Net is designed for highly skewed traffic.

2) CBB-Net vs 3:1 Fat-tree: Fat-tree is a static network and does not incur the reconfiguration delay present in periodic RDCNs when delivering large flows. As a result, Fig. 11 shows that Fat-tree achieves lower avg FCT for large flows at low traffic load ($L' = 0.2$). At higher loads, Fat-tree suffers from severe congestion at ToRs with concentrated traffic, leading to a sharp increase in avg FCT even at $L' = 0.4$ due to the highly skewed traffic. CBB-Net sustains a much higher traffic load.

3) CBB-Net vs MegaSwitch: Since the source code of MegaSwitch is unavailable, we implement a flow-level simulator of MegaSwitch following the design described in its paper, which excludes queuing delay. Fig. 13 shows that CBB-Net generally outperforms MegaSwitch for two main reasons.

First, CBB-Net responds more quickly to changes in demand—particularly for large flows—due to its shorter topology and routing computation times (100 μs vs. 10 ms). This faster response enables CBB-Net to drain traffic more quickly and free bandwidth for newly arriving flows. At high loads, such demand changes occur more frequently, which degrades large-flow performance in MegaSwitch; as a result, MegaSwitch becomes unstable at $L' = 0.6$. CBB-Net sustains a higher load.

Second, as discussed in (§I), MegaSwitch is capacity-inflexible: it reserves a fixed portion of bandwidth for small flows and assigns only the remaining capacity to large flows. This design limits the bandwidth available to small flows even when few large flows are present. Increasing the number of basemesh wavelengths from 3 to 4 improves small-flow performance in MegaSwitch, but comes at the expense of large-flow performance. Note that the superior performance of the few smallest flow sizes in MegaSwitch is partly due to the absence of queuing delay in the flow-level simulator, although transmission and propagation delays are accurately modeled.

C. Sensitivity analysis

The sensitivity analysis uses the Microsoft(0.5) traffic pattern at $L' = 0.8$, where highly congested networks still regularly employ CBB graphs. We use the Datamining distribution, which includes more large flows (100 MB, 250 MB, and 1 GB), to better evaluate sensitivity.

1) Impact of demand collection frequency: We analyze how the demand collection frequency affects the avg FCT of large flows. As the collection interval increases from 2 (default) to 10 cycles (3.8 ms per cycle), the average FCT of 1 GB flows remains stable even at 38 ms, since these long-lasting flows (~300 ms) are largely insensitive to delayed demand collection. Meanwhile, 100 MB flows experience a degradation no more than 8%. This suggests that CBB-Net under skewed traffic is quite robust to demand collection frequency.

2) Impact of demand collection failure: CBB-Net requires each ToR to iteratively forward demand information packets to construct a ToR-level demand matrix. We evaluate CBB-Net's sensitivity to this demand-packet loss during demand collection. In this setup, every ToR sends a demand information packet every two cycles, and the packet drop rate is varied from 0% to 20%. We found that even at an unlikely high drop rate of 20%, large flows of all sizes experience no more than a 4% increase in avg FCT.

D. Computation Overhead and Scalability

Computing a demand-aware CBB graph involves three steps: forming connected components, solving the Knapsack placement, and computing the mapping. We measure each step by generating random concurrent large flows, repeating 10^5 times per network size.

As shown in Fig. 14, the Knapsack step dominates the delay. Nevertheless, total computation remains within the cycle time for all sizes, indicating scalability beyond 108 ToRs. As the network scales, the cycle time can be preserved by proportionally increasing OCSes so that matchings per OCS remain constant.

CBB-Net introduces minor overhead on path length of small flows. Opera achieves slightly shorter average hop counts (3.12 vs. 3.34) due to no topology transitions and higher path diversity. However, this has negligible impact on average FCT (Fig. 11). Additional overhead from packet rerouting during transitions (§IV-C2) is also limited: on average, only 2.8 of 54 ToR pairs switch every two cycles.

E. Effort–Benefit Trade-off in CBB-Net

While it is challenging to directly quantify this trade-off compared to a low-complexity, demand-oblivious approach, we have previously shown that CBB-Net incurs only minimal operational overhead to enable demand-aware operation. In addition, the required control channel between the controller and the OCSes constitutes a one-time infrastructure cost.

Whether the benefits of CBB-Net outweigh the added complexity ultimately depends on the characteristics of traffic patterns. If the pattern is close to all-to-all, a demand-oblivious approach may be more suitable. Conversely, when the pattern is highly skewed, CBB-Net provides a substantial reduction in large-flow FCT.

VII. Conclusion

This work proposes CBB-Net, a demand-aware periodic microsecond-switching RDCN. We build on the properties of the CBB graph to provide a demand-aware emulated graph at microsecond timescales, resulting in higher throughput than a complete graph under skewed traffic. Our evaluation shows that, under highly skewed traffic patterns, large flows in CBB-Net achieve up to a 1.79× reduction in average FCT compared to a complete-graph-based system.

References

[1] Ankit Singla, Chi-Yao Hong, Lucian Popa, and P. Brighten Godfrey. Jellyfish: Networking data centers randomly. In *USENIX NSDI*, 2012.

[2] Nils Blach, Maciej Besta, Daniele De Sensi, Jens Domke, Hussein Harake, Shigang Li, Patrick Iff, Marek Konieczny, Kartik Lakhotia, Ales Kubicek, Marcel Ferrari, Fabrizio Petrini, and Torsten Hoefler. A High-Performance design, implementation, deployment, and evaluation of the slim fly network. In *USENIX NSDI*. USENIX Association, April 2024.

[3] Asaf Valadarsky, Gal Shahaf, Michael Dinitz, and Michael Schapira. Xpander: Towards optimal-performance datacenters. In *ACM CoNEXT*, 2016.

[4] Simon Kassing, Asaf Valadarsky, Gal Shahaf, Michael Schapira, and Ankit Singla. Beyond fat-trees without antennae, mirrors, and disco-balls. In *ACM SIGCOMM*, 2017.

[5] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: a scalable and flexible data center network. In *ACM SIGCOMM*, 2009.

[6] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, Anand Kanagala, 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. In *ACM SIGCOMM*, 2015.

[7] 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 *ACM SIGCOMM*, 2010.

[8] Guohui Wang, David G. Andersen, Michael Kaminsky, Konstantina Papagiannaki, T.S. Eugene Ng, Michael Kozuch, and Michael Ryan. C-through: Part-time optics in data centers. In *ACM SIGCOMM*, 2010.

[9] He Liu, Feng Lu, Alex Forencich, Rishi Kapoor, Malveeka Tewari, Geoffrey M. Voelker, George Papen, Alex C. Snoeren, and George Porter. Circuit switching under the radar with reactor. In *USENIX NSDI*, 2014.

[10] Xia Zhou, Zengbin Zhang, Yibo Zhu, Yubo Li, Saipriya Kumar, Amin Vahdat, Ben Y. Zhao, and Haitao Zheng. Mirror mirror on the ceiling: flexible wireless links for data centers. In *ACM SIGCOMM*, 2012.

[11] 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 *ACM SIGCOMM*, 2013.

[12] Ankit Singla, Atul Singh, and Yan Chen. OSA: An optical switching architecture for data center networks with unprecedented flexibility. In *USENIX NSDI*, 2012.

[13] Monia Ghobadi, Ratul Mahajan, Amar Phanishayee, Nikhil Devanur, Janardhan Kulkarni, Gireeja Ranade, Pierre-Alexandre Blanche, Houman Rastegarfar, Madeleine Glick, and Daniel Kilper. Projector: Agile reconfigurable data center interconnect. In *ACM SIGCOMM*, 2016.

[14] 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 *USENIX NSDI*, 2017.

[15] 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 *ACM SIGCOMM*, 2017.

[16] William M. Mellette, Rajdeep Das, Yibo Guo, Rob McGuinness, Alex C. Snoeren, and George Porter. Expanding across time to deliver bandwidth efficiency and low latency. In *USENIX NSDI*, 2020.

[17] Hitesh Ballani, Paolo Costa, Raphael Behrendt, Daniel Cletheroe, Istvan Haller, Krzysztof Jozwik, Fotini Karinou, Sophie Lange, Kai Shi, Benn Thomsen, and Hugh Williams. Sirius: A flat datacenter network with nanosecond optical switching. In *ACM SIGCOMM*, 2020.

[18] Daniel Amir, Nitika Saran, Tegan Wilson, Robert Kleinberg, Vishal Shrivastav, and Hakim Weatherspoon. Shale: A practical, scalable oblivious reconfigurable network. In *ACM SIGCOMM*, 2024.

[19] Cong Liang, Xiangli Song, Jing Cheng, Mowei Wang, Yashe Liu, Zhenhua Liu, Shizhen Zhao, and Yong Cui. Negotiator: Towards a simple yet effective on-demand reconfigurable datacenter network. In *ACM SIGCOMM*, 2024.

[20] Navid Hamedazimi, Zafar Qazi, Himanshu Gupta, Vyas Sekar, Samir R. Das, Jon P. Longtin, Himanshu Shah, and Ashish Tanwer. Firefly: a reconfigurable wireless data center fabric using free-space optics. In *ACM SIGCOMM*, 2014.

[21] Johannes Zerwas, Csaba Györgyi, Andreas Blenk, Stefan Schmid, and Chen Avin. Duo: A high-throughput reconfigurable datacenter network using local routing and control. *ACM Meas. Anal. Comput. Syst.*, 2023.

[22] Vamsi Addanki, Chen Avin, and Stefan Schmid. Mars: Near-optimal throughput with shallow buffers in reconfigurable datacenter networks. *ACM Meas. Anal. Comput. Syst.*, 2023.

[23] William M. Mellette, Alex Forencich, Rukshani Athapathu, Alex C. Snoeren, George Papen, and George Porter. Realizing rotornet: Toward practical microsecond scale optical networking. In *ACM SIGCOMM*, 2024.

[24] Daniel Halperin, Srikanth Kandula, Jitendra Padhye, Paramvir Bahl, and David Wetherall. Augmenting data center networks with multi-gigabit wireless links. In *ACM SIGCOMM*, 2011.

[25] HT-sim. Ndp simulator. https://github.com/nets-cs-pub-ro/NDP, 2017.

[26] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. Inside the social network's (datacenter) network. In *ACM SIGCOMM*, 2015.

[27] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2026.

[28] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W. Moore, Gianni Antichi, and Marcin Wójcik. Re-architecting datacenter networks and stacks for low latency and high performance. In *ACM SIGCOMM*, 2017.

[29] Tae Joon Seok, Niels Quack, Sangyoon Han, Richard Muller, and Ming Wu. Large-scale broadband digital silicon photonic switches with vertical adiabatic couplers. *Optica*, 2016.