RDC: Energy-Efficient Data Center Network Congestion Relief with Topological Reconfigurability at the Edge

Weitao Wang[†], Dingming Wu[*], Sushovan Das[†], Afsaneh Rahbar[†], Ang Chen[†], and T. S. Eugene Ng[†]
[†]Rice University, [*]Bytedance Inc.

Abstract

The *rackless data center* (RDC) is a novel network architecture that logically removes the rack boundary of traditional data centers and the inefficiencies that come with it. As modern applications generate more and more inter-rack traffic, the traditional architecture suffers from contention at the core, imbalanced bandwidth utilization across racks, and longer network paths. RDC addresses these limitations by enabling servers to logically move across the rack boundary at runtime. Our design achieves this by inserting circuit switches at the network edge between the ToR switches and the servers, and by reconfiguring the circuits to regroup servers across racks based on the traffic patterns. We have performed extensive evaluations of RDC both in a hardware testbed and packet-level simulations and show that RDC can speed up a 4:1 oversubscribed network by 1.78× ∼ 3.9× for realistic applications and more than 10× in large-scale simulation; furthermore, RDC is up to 2.4× better in performance per watt than a conventional non-blocking network.

1 Introduction

The importance of the data center network (DCN) has led to a series of DCN architecture proposals [26, 43, 44, 53, 59, 62, 66, 74, 80, 82, 84–86, 96, 110, 118] over the past decade. Although these proposals have competing designs for the network core, the designs for the network edge are similar: servers organized in racks. The network core connects multiple racks, and each rack hosts tens of servers that are connected via a Top-of-Rack (ToR) switch. Standardized racks enable unified power supply and cooling, as well as significant space and cable savings. This rack-based topology and connectivity pattern is deeply ingrained in the design of existing DCN architectures.

While traffic within a rack experiences no congestion, traffic across racks often has to contend for bandwidth due to oversubscription in the network core[1]. At the same time, traffic across racks is increasing in data center workloads [36, 37, 40, 99]. Firstly, more and more DCN traffic is escaping the rack boundary due to resource fragmentation [61], large-scale jobs [24], specific application placement constraints for fault tolerance [13], and service-based rack organization for operational convenience [99]—e.g., one rack may host storage servers, and another rack may host cache servers. Secondly, there is also an increasing amount of traffic that leaves the pod. For instance, a web-frontend cluster may need to retrieve data from a database cluster or submit jobs to a Hadoop cluster [99].

Thus, the need for efficient handling of cross-rack traffic has motivated numerous approaches; but they have one thing in common – they view the rack design (i.e., a ToR switch connecting tens of servers) as a given. Firstly, the non-blocking network and its alternatives [26, 62, 64, 65, 82, 109] aim to enlarge the capacity of the network core. However, due to the scaling limit of CMOS-based electrical packet switches [6, 33, 34, 49, 50, 57, 91, 104, 105], building such a network while staying within the datacenter power budget is challenging [107]. Secondly, rack-level reconfigurable networks [53, 74, 80, 110, 118] add additional bandwidth between the most intensively communicating racks with extra cables, lasers, or antennas to relieve the bottleneck at the core. However, the performance improvement is constrained by the fact that the number of additional paths is usually limited. Thirdly, smarter job placement and execution strategies [39, 40, 45, 46, 71, 72, 87, 108, 116, 120] can also reduce the inter-rack traffic by arranging the jobs based on their traffic pattern. However, these placement solutions cannot perform well if traffic patterns fluctuate at runtime or if the application dictates placement and forces the traffic to be cross-rack.

This paper studies a complementary and little-explored point in the design space, which we call the *rackless data center (RDC)* architecture. It logically removes the fixed, topological rack boundaries while preserving the benefits of rack-based designs, e.g., organized power supply and cooling, and space efficiency. In RDC, servers are still mounted on physical racks, but they are not bound statically to any ToR switch. Rather, they can move logically from one ToR to another. Under the hood, this is achieved by the use of the circuit switches (CS), which can be dynamically reconfigured to form different connectivity patterns. In other words, servers remain immobile, but circuit changes may shift them to different topological locations. Therefore, this new architecture is not committed to any static configuration, so servers that heavily communicate with each other can be grouped on demand, and they can be regrouped as soon as the pattern changes again. Such dynamic server regrouping enabled by RDC leads to performance benefits in many common, real-world scenarios (details in §2).

[1]The literature suggests that there exists a wide-range of common oversubscription ratios between 4:1 to 20:1 [43, 62, 99, 105].

We make the following contributions: 1) a novel architecture called RDC, which can be reconfigured to connect servers under different racks in the same logical locality group despite physical rack boundaries; 2) a low-latency RDC control plane and algorithms, which continuously optimize the RDC topology based on the traffic patterns; 3) a prototype of RDC in both testbed and simulation settings, demonstrating that RDC boosts the performance of a 4:1 oversubscribed network by $1.78\times \sim 3.9\times$ for realistic applications and more than $10\times$ in large-scale simulation; furthermore, RDC is up to $2.4\times$ better in performance per watt than a conventional non-blocking network.

2 Motivation

RDC is motivated by inefficiencies that stem from the inherent rack boundaries in today's data centers. RDC enables dynamic topological reconfiguration to regroup servers, leading to improved performance for modern workloads. We propose to realize RDC using circuit switching technologies.

2.1 Rack sizes are inherently limited

Today's DCNs are organized in physical racks as the basic unit. Communication within a rack is through a ToR switch and enjoys lower latency and higher throughput than that across racks. This rack boundary is stressed by a combination of two trends. First, applications are becoming data-intensive. DNN training, iterative machine learning, HPC, big data frameworks (MapReduce, Spark, HDFS) and many other workloads require extensive data communication. Second, the advent of domain-specific accelerators (GPUs, TPUs) and non-volatile memories (NVM) is further shifting the major bottleneck from computation to network IO. The convergence of these trends leads to the need to maximize rack-level performance as much as possible. Broadcom's Tomahawk-4 64x400 Gbps—the fastest Ethernet switch ASIC commercially available on the market today [7]—only supports a rack boundary of tens of servers while maintaining maximum rack-level performance. A few years ago, the End-of-Row architecture was proposed as an alternative, where multiple racks of low port speed servers were connected to a high-radix edge switch to form a larger logical rack [1]. However, high-radix switching is not feasible at high port speeds: 400 Gbps ports are common today, and Ethernet standards are growing to terabit level. Therefore, in the foreseeable future, the physical rack boundaries of tens of servers are here to stay. New solutions are necessary to mitigate inter-rack-level bottlenecks.

2.2 Rack boundaries introduce bottlenecks

1. Jobs fragmented across racks. A job may spread across racks if rack resources are fragmented. This is partly because cluster schedulers assign resources to their own jobs locally [5, 11, 12]; also, dynamic job churns ensure that rack resources aren't always neatly packed [60, 95]. Such resource fragmentation leads to heavy inter-rack traffic which contends for bandwidth due to oversubscription.

2. Workloads with dynamic traffic patterns. Many data-intensive applications (e.g., DNN training, HPC) consist of multiple stages, and each stage has a different yet predictable traffic pattern. For example, Distributed Matrix Multiplication (DMM) has broadcast (one-to-many) and shift (one-to-one) traffic patterns among different subsets of servers in every iteration (Fig. 8(e)). When these jobs coexist in a cluster, the overall combined traffic pattern will change dynamically and predictably. For such workloads, no static job allocation is sufficient to localize all the traffic patterns simultaneously.

3. Applications with placement constraints. Applications may intentionally spread their instances across racks to balance load [55] to reduce synchronized power consumption spikes [70], or to achieve fault tolerance [13]. For instance, to increase resilience, some distributed storage systems, like HDFS, require at least one replica to be placed on a different rack. These requirements result in placement constraints that are by design crossing rack boundaries.

4. Imbalanced out-of-pod traffic. In large datacenters, traffic patterns across racks are often skewed, and out-of-pod traffic demand for each rack is different. For example, only 7.3% of the traffic from the frontend servers is inter-pod, comparing to 40.7% for the cache servers [41, 99]. Operationally, data centers tend to group servers based on their types [99]. So, the above heterogeneity of the out-of-pod traffic demand will make some racks' uplinks highly congested (e.g., cache) while other racks still have unused bandwidth (e.g., frontend).

2.3 Facebook trace analysis: A case study

Methodology. We used a public dataset released by Facebook, which contains packet-level traces collected from their production data centers in a one-day period. The traces were collected from the "frontend", "database", and "Hadoop" clusters, sampled at a rate of 1:30 k, and each packet contains information about the source and destination servers [4]. To understand the benefits of removing rack boundaries, we simulate a rackless design by regrouping servers of different racks into "logical" racks using the algorithms presented in §3. We have two major findings.

Observation #1: Intensive inter-rack traffic. The first observation from the traces is that most of the traffic crosses rack boundaries in a pod. Fig. 1(a) shows the heatmap of traffic pattern inside a frontend pod with 74 racks, collected during a 2-minute interval. If a server in rack i sends more traffic to another server in rack j, then the pixel (i, j) in the heatmap will become darker. Intra-rack traffic appears on the diagonal (i.e., $i = j$). The scattered dots show that the traffic does not exhibit rack locality—in fact, 96.26% of the traffic in this heatmap is inter-rack but intra-pod. A similar trend exists for the database trace: 92.89% of traffic is inter-rack but intra-pod. Hadoop trace has more intra-rack traffic but still has 52.49% of traffic being inter-rack but intra-pod.

Implication #1: Regrouping servers improves locality. Fig.

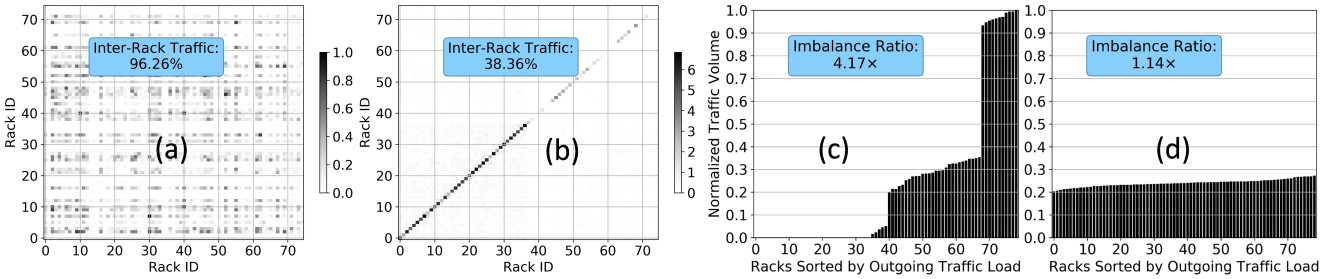

Figure 1: Traffic patterns from the Facebook traces. (a) is the rack-level traffic heatmap of a representative frontend pod. (b) shows the heatmap after regrouping servers in (a). (c) and (d) plot the sorted load of inter-pod traffic across racks in a representative database pod, before and after server regrouping, respectively.

1(b) shows the heatmap if servers are regrouped under different racks based on their communication intensity, simulating the desired effect of RDC. Here, most of the traffic is on the diagonal, and inter-rack traffic is reduced significantly to 38.4%. Assuming a 4:1 oversubscribed network, what used to be inter-rack traffic now enjoys 2.82x higher bandwidth. For the database and Hadoop traces, the inter-rack traffic ratios after regrouping are 28.4% and 41.6%, respectively.

Observation #2: Out-of-pod traffic imbalance. Another notable trend is the heavy imbalance of out-of-pod traffic. Fig. 1(c) sorts the racks based on the amount of out-of-pod traffic they sent (traffic trace: database) in a 20-min interval, where the X-axis is the rack ID, and the Y-axis is the (normalized) out-of-pod traffic volume. As we can see, the top 11 racks account for nearly 50% of the out-of-pod traffic, and almost half of the racks never sent traffic across pods. Therefore, some uplinks of ToR switches are heavily utilized, whereas other links are almost always idle. The load imbalance, defined as $max(L_i)/avg(L_i)$, where L_i is the amount of out-of-pod traffic from rack i, is as much as 4.17. We found qualitatively similar results on other traces.

Implication #2: Grouping servers mitigates load imbalance. Fig. 1(d) shows the results if servers can be regrouped. In the simulated RDC network, the inter-pod traffic is much more evenly load-balanced across racks, achieving a load imbalance of 1.14. Moreover, the aggregated bandwidth for the out-of-pod traffic increases to 1.79x of the previous bandwidth. This would make better use of the ToR uplinks and avoid congesting any particular link due to imbalance.

2.4 The Power of RDC

Driven by the application-level demand and trace-based analysis, we propose the concept of *rackless data center (RDC)*, which logically removes the physical rack boundaries while maintaining the high-speed rack-level performance. In RDC, servers are mounted on the same "physical rack" sharing the power supply and cooling system but can be logically moved across the ToR switches. We call the new groups of servers served by the same ToR a "logical rack". Fig. 2 illustrates the benefits of RDC due to server regrouping.

1. Mitigate the effect of resource fragmentation. RDC can

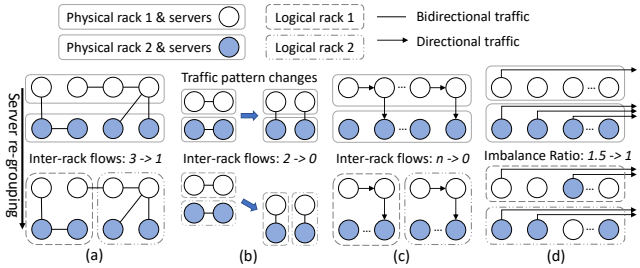

Figure 2: Comparisons between before and after server regrouping for (a) placement optimization, (b) dynamic optimization for evolving patterns, (c) application constraints accommodation, and (d) out-of-pod load balancing.

reduce the effect of resource fragmentation by relocating the heavily communicating server groups under the same logical rack, thus reducing inter-rack traffic. RDC can completely localize smaller jobs that are possible to be packed within one logical rack, like the job on the left-hand side of Fig. 2(a). Even for bigger jobs that cannot be packed within one logical rack, RDC benefits them by (1) localizing as many traffic flows as possible to logical racks, like the job on the right-hand side of Fig. 2(a); and (2) minimizing overall inter-rack traffic from all jobs, leaving the core bandwidth to be shared by much fewer flows that must cross the rack boundaries.

2. Optimize for dynamic traffic patterns. The ability of dynamic server regrouping enabled by RDC can potentially optimize the applications with variable yet predictable traffic patterns. With such changing patterns as shown in Fig. 2(b), RDC is able to dynamically change the topology and minimize the inter-rack traffic for all patterns.

3. Accommodate application placement constraints. As shown in Fig. 2(c), application-level constraints can be accommodated by RDC while localizing traffic. For example, HDFS always requires at least one data block replica to be placed on a different rack. By regrouping the servers from different racks into one logical rack, RDC can place the replicas to a different physical rack but within the same "logical" rack, which provides higher bandwidth and also satisfies the replica placement policy of HDFS.

4. Balance out-of-pod traffic. RDC is able to regroup the servers according to their out-of-pod traffic demands and balance link utilization, hence relieving the bottleneck. In

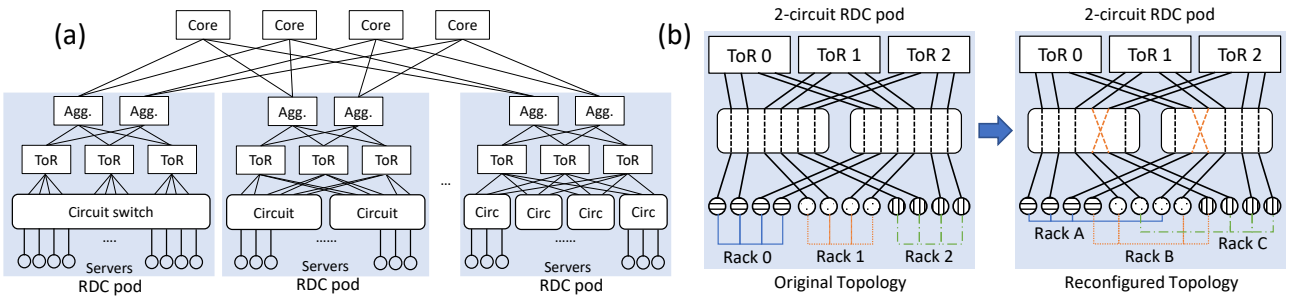

Figure 3: RDC architecture overview. (a) is an example of the RDC network topology. Different numbers of circuit switches can be inserted at the edge between servers and ToR switches. Connectivities for aggregation switches (*agg.*) and core switches remain the same as in traditional Clos networks. (b) shows the original topology and an example reconfigured topology for a 2-CS RDC pod with 3 racks and 4 servers under each rack.

Fig. 2(d), the imbalance ratio has been decreased to 1 from 1.5 after the grouping is changed according to the out-of-pod traffic demand.

2.5 Realizing RDC

Circuit switches (CS) are widely used to provide reconfigurable connections among end points, which is a great fit for the server regrouping functionality of RDC that we discussed above. One realization of RDC is to connect all the servers and all the ToR switches within a pod with a single CS. Alternatively, RDC can also use multiple smaller port count CSes to form a distributed reconfigurable server-to-ToR fabric.

RDC can potentially leverage any kind of CS technologies, including optical and electrical circuit switches alike [104]. However, at high data rates, optical transceivers are the standard interfaces. Therefore, to make the realization long-term sustainable, we consider various optical circuit switching (OCS) technologies. Several OCS technologies are available today such as 3D/2D MEMS, AWGR, etc. Fundamentally, OCS does not perform packet-level processing and forwards the photon beams using mirror rotation, diffraction, etc., which leads to some inherent advantages such as a) agnostic to data-rate (or modulation format), b) negligible power consumption, c) negligible forwarding latency due to no buffering, and d) no need of transceivers at the OCS ports. Additionally, different OCS technologies can provide very fast switching. For example, 2D-MEMS-based OCSes provide microsecond switching [96]), AWGR switches with the latest tunable transceivers can provide nanosecond switching [33, 35, 48, 49, 58, 77]. Moreover, OCS are highly reliable [101] and, due to their simplicity, mostly free from firmware bugs and software misconfigurations.

3 The RDC Architecture

3.1 Connectivity structure

RDC changes the traditional multi-layer Clos topology [26, 62] by inserting one or more circuit switches (CS) at the edge layer between the servers and ToR switches, so that the server can be connected to different ToR switches through circuit reconfiguration. The aggregation and core layers of the network remain the same. Each circuit switch has some ports connected to every ToR switch within the pod to guarantee that the servers could be connected to any ToR switch. For the 1-CS RDC pod, the servers can be grouped without constraints, as long as the number of servers under each ToR switch is the same. If multiple circuit switches are used in one pod, the additional connectivity constraint is that not all the servers under one circuit switch can be connected to the same ToR. With such design, RDC maximizes the flexibility to permute the server-ToR connectivities, allowing the most intensively communicating servers to be localized under the same ToR and enjoy the line rate throughput.

Fig. 3(a) shows an example of the RDC pods. For a pod with m racks and n servers per rack, $2mn$ ports should be provided by all the circuit switches in total to link both servers and ToR switches. For instance, a 16-rack pod with 32 servers can be built with either 1 circuit switch with 1024 ports or k switches with $\frac{1024}{k}$ ports each. Fig. 3(b) gives a detailed example of inserting multiple circuit switches and how to reconfigure for regrouping servers. For a pod with k circuit switches, $\frac{n}{k}$ servers under each ToR are connected to one circuit switch, so that the original topology can keep every server under its own physical ToR switch.

Intuitively, if we increase the number of CSes, the design becomes more distributed which decouples it from a particular CS technology's port count availability; while at the same time, the flexibility of moving servers across the ToRs is slightly reduced. To shed light on this trade-off, we perform trace-based analysis with varying numbers of CSes between the servers and ToR switches. To find a valid server regrouping, we formulate an Integer Linear Programming (ILP) which maximizes the traffic localization given the constraints arising from multiple CSes (more details in §4.3). For the analysis, we consider an RDC pod with 16 ToRs and 32 servers per ToR, having 4 : 1 oversubscription above the ToR level. We vary the number of CS from 1 to 8 and compare the performance with a static 4 : 1 oversubscribed network. Fig. 4 shows a boxplot of the flow completion times (FCT) of these architectures for flow-level Cache traffic trace generated from [99]. We observe that the potential benefit of RDC remains high

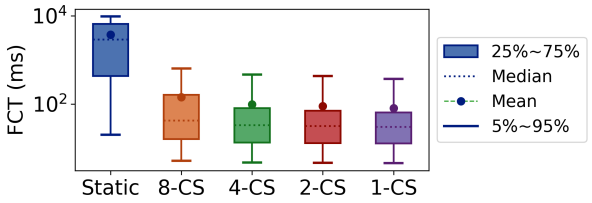

Figure 4: The potential improvement for FCT remains high across a wide range of multi-CS configurations in RDC

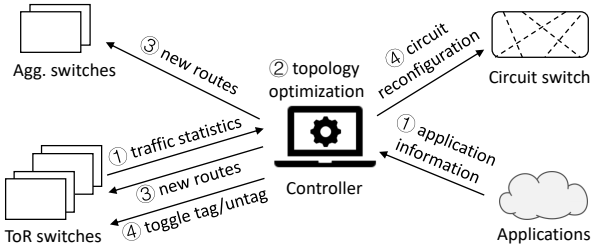

Figure 5: Workflow overview of RDC.

across a wide range of CS configurations, which validates the efficacy of our distributed design.

3.2 The RDC Controller

Today's data centers are constructed from modular pods [3, 14, 19, 22], where a pod typically hosts one type of service. RDC similarly views pods as basic units and uses a per-pod network controller that manages both packet switches and circuit switches within the pod. The controller reconfigures the network at timescales of seconds or longer depending on the traffic pattern. It has two operation modes: it can receive the traffic demands or commands from the applications directly in the proactive mode, or passively monitor the traffic statistics from packet switches in the reactive mode.

We illustrate the workflow for both modes in Fig. 5. The controller 1) first collects the traffic statistics by querying the flow counters on the ToRs, or passively receives the information from the applications; 2) determines the optimized topology with certain optimization goals (§4); 3) generates a set of new routes and pre-installs them on the packet switches; and 4) finally sends the circuit reconfiguration request to the circuit switch and simultaneously activates the new routing rules on packet switches. The first two steps serve as the RDC control plane (discussed later in §4), while the last two steps configure the data plane (discussed in §3.3). Note that only the final step would cause a small amount of disturbance due to the circuit reconfiguration delay.

3.3 Routing

In traditional DCNs, forwarding rules are aggregated based on IP prefixes. In RDC, such aggregation does not work as servers have no fixed locations. Instead, RDC uses per-pod flat IP addressing and exact matching rules on packet switches. Topology changes are captured by updating the routing rules. These rule updates are for intra-pod routing only, as routing mechanisms across pods remain unchanged.

In an RDC pod, each ToR has a flow table entry for every server IP in its rack, and a single default entry for other addresses outside the rack. Each ToR splits traffic to other racks equally across its uplinks using ECMP [69]. All *agg.* switches have the same forwarding table: one entry per destination IP. The flow entries on ToRs and *agg.* switches both need to be updated when topology changes. For ToRs, only the rules for downward traffic need to change; the default ECMP entry for upward traffic remains the same. Therefore, for an RDC pod with m racks and n servers per rack, a topology change could result in n rule updates on ToRs and $m \times n$ updates on *agg.* switches, which is on the order of hundreds to a thousand. Updating this number of rules on an OpenFlow switch could take 100ms to over 1s [68, 76]. Previous works have developed the *two-phase commit* method to reduce disruption during updates [81, 98], which first populate the switches with new routing rules and then flip the packet version at the ingress switches. However, such an approach cannot avoid packet loss in the transient state, like changing the packet version rule [81]. This is because updating the packet version rule at the ingress switch requires two rule changes—removing the old rule and installing the new rule—and therefore is not atomic. Our measurement on a Quanta T3048-LY2R OpenFlow switch shows the transient period could last for 0.5ms.

Instead of changing the packet version, in RDC a switch performs binary changes from VLAN tagging packets to not tagging them, and vice versa. The VLAN-tagged packets will match a group of rules with the VLAN tag as a match field, whereas the packets without VLAN tags will match a more general group of rules without VLAN IDs. In this way, adding and removing a single VLAN tag rule achieves the same goal as changing the packet version, but the operation is atomic and avoids packet loss (more details in §A.1.) We apply this update approach to both ToR and *agg.* switches but only tag or not tag packets on ToR switches. Tag flipping actions are only performed when the new forwarding rules have been populated network-wide. The VLAN tag flipping actions need to be executed at the same time across multiple ToRs; such network-wide changes can be performed using well-known SDN time synchronization [90] and consistent update [42, 73, 100] techniques, so that changes can be synchronized and take effect atomically.

3.4 Discussions

Reducing the path length: Besides the throughput benefits mentioned in §2, localizing the hosts under the same logical rack would effectively reduce the average path length (evaluated in §5.2), and thus reduce the network latency. Therefore, low-latency applications and disaggregated systems [56, 63, 92, 94, 103] may benefit from RDC's design as well.

ToR failure handling: In a traditional data center, a ToR failure disconnects all servers in the rack. ToR failures are handled either by multi-homing servers to several other ToRs

[79, 83] or by replicating applications under multiple ToRs [113, 119]. But in RDC, servers are not tied to any particular ToR, servers under a failed ToR can be migrated to a healthy ToR. To host the relocated servers, we can reserve some "free" ports on each ToR for recovery or install a set of backup ToRs [112, 114]. Specifically, for an RDC pod with m racks and n servers per rack, we only need $\frac{n}{m}$ free ports per ToR (or n ports overall) to recover from any single ToR failure, which incurs a low additional cost and complexity.

CS failure handling: In general, circuit switches are extremely reliable [102]. Commercial OCS products have more than 28.5 years of mean-time-between-failure (MTBF) and come with redundant control processors [2]. However, if a CS failure happens, only a small fraction of servers will be disconnected uniformly under each ToR of RDC, due to its connectivity structure. The most common failure mode for the CS is the power outage. To mitigate this, multiple redundant power supplies can be used for the CS [2]. For further protection, battery backups can be used—since the CS draws only tens of Watts, a battery backup already goes a long way.

4 RDC Control Algorithms

RDC has a general framework to support various topology optimization algorithms, working in two modes to collect the traffic demand matrix and compute the reconfiguration plan.

4.1 Proactive-mode RDC

The *proactive* mode of RDC allows applications to explicitly call the RDC controller via RPC with two APIs: **1) Traffic demand matrix** can be reported by the applications to request reconfigurations. Along with the demand matrix, RDC controller will request the application to specify one topology optimization algorithm from the algorithms described in §4.3 as well. After receiving the request, the RDC controller will calculate an optimal topology with a specified algorithm and conduct the reconfiguration accordingly. **2) Raw configuration commands** can also be given directly from the applications. For this method, formatted data to describe the new circuit connections will be sent to the controller, so that the controller could bypass the calculation of the optimal topology and directly used the received configuration to initiate the reconfiguration. An additional benefit for applications to send raw configuration plans is that it enables network-aware job placement and scheduling since the applications know the future network requirements in advance.

There are several scenarios where applications can benefit from telegraphing their intent to the RDC controller: 1) In a case where applications intentionally spread their deployment across racks—e.g., for fault tolerance [40] or for reducing synchronized power consumption spikes [70]—inter-rack traffic patterns are unavoidable in traditional architectures. In RDC, however, such applications can request relevant servers to be grouped together logically. 2) The cluster applications may be allocated with resources from multiple racks due to fragmentation. By aggregating those fragmented resources to the same logical rack, RDC improves the bandwidth and reduces the average latency. 3) When applications have changing traffic patterns (e.g., distributed matrix multiplication (DMM) algorithms proceed in iterations with shifting traffic patterns), they can request reconfigurations before the next phase starts to ensure locality throughout the job. 4) Last but not least, RDC could rely on the out-of-pod traffic demands reported by applications to balance the load across different ToR uplinks.

We evaluate three different applications in §5.1 to show the performance of proactive-mode RDC, including HDFS, Memcached, and DMM.

4.2 Reactive-mode RDC

The *reactive* mode of RDC does not require to modify application; it collects traffic statistics from the network in one epoch, and reconfigures the network with an optimized topology for the next, based on the statistics and one of the optimization algorithms from §4.3, specified by the network operators.

Traffic statistics. The RDC controller pulls flow counters from ToRs periodically. A flow counter associates the 5-tuple (13 bytes) of a flow to an 8-byte counter value and thus has 21 bytes in total. Switch memory constraint is traditionally the main concern of maintaining per-flow counters, but this constraint is loosening over the years as the switch SRAM size has been continuously growing. The most recent switch ASICs have 50-100MB of SRAM and can store millions of flow states [18, 88]. As recent DCN measurement works show that the number of concurrent flows per server is on the order of hundreds to a thousand [28, 99], each ToR in RDC would then need tens of thousands of flow counters assuming tens of servers per rack. For instance, assuming an RDC pod with 16 racks and 32 servers per rack, and a counter pulling period of 10s, the control channel bandwidth usage is roughly 8.6Mbps, which is low enough to be feasible.

Demand estimation algorithm. Previous works have shown that data center workloads demonstrate certain degrees of stability [38, 99], and RDC similarly relies on this stability to estimate the traffic demand based on historical data. But the *observed* traffic volumes on ToR switches are biased by the *current* topology, so it is important to estimate the true traffic demand, i.e., the traffic demand when flows are not bottlenecked by the network core. Mitigating such observation bias has been studied in previous work, Hedera [27], and we adopt a similar heuristic.

A flow could be bottlenecked either by the network or by the application itself. We call the first class of flows *elastic* and the second *non-elastic*, and RDC only considers elastic flows. The heuristic is to remove flows from the observed traffic matrix whose sizes are smaller than their fair share. The remaining flows are treated as elastic, and RDC calibrates for potential bias in the counters by computing their idealized bandwidth share (i.e., their bandwidth share if they are only bottlenecked by the host NICs' capacity) as the es-

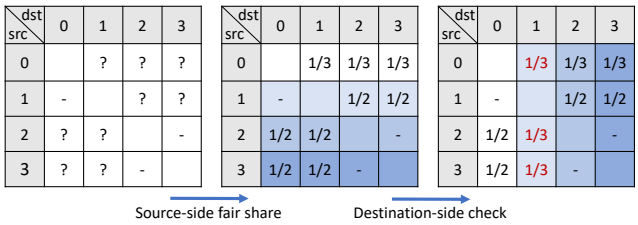

Figure 6: Hedera demand estimation example. Each "?" represents one flow from source host to destination, "-" represents no flow between that source-destination pair, and number "1/2" represents 50% of host bandwidth. This example ends in one iteration, but it takes more iterations for a more complicated traffic matrix.

timated demand [27]. Hedera is an algorithm to calculate the max-min fair share rate of each flow within a network. It performs multiple iterations to firstly increase the flow capacities at the source (no greater than the source host capacity) and then decrease the exceeding capacities (sum of enlarged flow capacities subtracting the actual NIC capacity) on each destination host until the flows' capacities converge. A simple demand estimation example that ends with only one iteration is shown in Fig. 6 (More details in §A.3). After convergence, the estimated flow demands are aggregated into a server-to-server traffic matrix for reconfiguration. The effectiveness of this demand estimation algorithm is evaluated in §5.4.

4.3 Topology optimization algorithms

RDC enables a range of topology optimization and reconfiguration algorithms.

1. Traffic localization algorithm reconfigures the network to localize inter-rack traffic, after obtaining the flow demands proactively or reactively. The objective of the localization algorithm is to minimize the traffic demands across the logical racks of the new topology. With this objective, the localization algorithm can be formulated as an Integer Linear Programming (ILP) problem as described in §A.4. However, finding the optimal solution is NP-hard, so we provide heuristic alternatives with balanced graph partition [75] for 1-CS RDC and a simplified algorithm for multi-CS RDC discussed in §A.4. The heuristic algorithms can find a high-quality regrouping plan within tens of milliseconds as shown in Table 2.

2. Uplink load-balancing algorithm spreads out-of-pod traffic across ToR switches for load balancing, relieving the potential congestion on the over-subscribed uplinks. The objective for uplink load balancing (ULB) is to minimize the maximum out-of-pod traffic from one rack. We provide a formal problem formulation and faster heuristic algorithms in §A.2.

3. Mixed optimizations can be developed in RDC to localize the inter-rack traffic and balance the out-of-pod traffic at the same time, e.g., for a *mix* of workloads or applications. To satisfy this goal, the objective of this problem will be minimizing $\alpha T + \beta R$, where T is the total inter-rack traffic demands within a pod, R is the maximum volume of out-of-pod traffic across ToRs, and α and β are the respective weights [40].

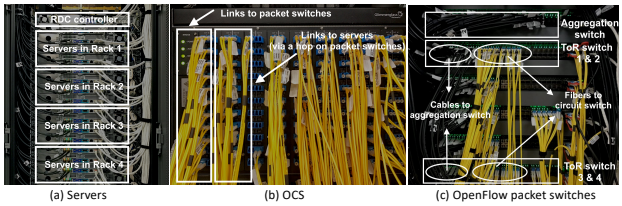

Figure 7: RDC prototype with 4 racks and 16 servers.

4. Scenario-specific optimizations allow applications or network operators to define their own optimization algorithms for regrouping the servers into logical racks. The applications are able to define their own objective function and add more application-specific constraints.

5 Implementation and Evaluation

We conduct comprehensive evaluations using testbed experiments and packet-level simulations. Our experiments focus on several dimensions: a) real-world applications of RDC to HDFS [10], Memcached [23], and MPI-based distributed matrix multiplication (DMM) [54] as use cases, b) packet-level simulations on the latency and throughput improvements at scale, c) packaging, power, and capital cost analysis, and d) microbenchmarks on RDC, including non-disruptive control loop latency.

Testbed. Our RDC prototype consists of 16 servers and 4 ToR switches in 4 logical racks, one *agg.* switch and one circuit switch; Fig. 7 illustrates our hardware testbed. The ToR switches are emulated on two 48-port Quanta T3048-LY2R switches. Each ToR switch has four downlinks connected to the servers, and one uplink to the *agg.* switch, forming an over-subscription ratio of 4:1. We can tune this ratio to emulate a non-blocking network by increasing the number of uplinks to 4. The *agg.* switch is a separate OpenFlow switch. The OCS is a 192-port Glimmerglass 3D-MEMS switch with a switching delay of 8.5 ms. This can also be replaced with other types of OCS. Each server has six 3.5 GHz dual-hyperthreaded CPU cores and 128 GB RAM, running TCP CUBIC on Linux 3.16.5. Most of our experimental results except the large-scale simulation in §5.2 are obtained on this testbed.

Packet-level simulator. In order to simulate a wider variety of experimental settings, we have developed a packet-level simulator based on *htsim*, which was used to evaluate MPTCP [97] and NDP [67]. This simulator has a full implementation of TCP flow control and congestion control algorithms and supports ECMP. We simulate a conservative circuit reconfiguration delay of 8.5 ms, which is what our testbed 3D-MEMS switch achieves. As discussed in §2.5, much faster circuit switching technologies exist [33, 48, 58, 86] that can further improve the performance of RDC. Note that only the circuit that is being reconfigured will experience a disruption; all other circuits continue to function. Packets in flight during reconfiguration will be dropped if they traverse the disrupted links, and unsent packets will be buffered at the servers. We simulate an RDC pod with 512 servers, 32 servers per rack,

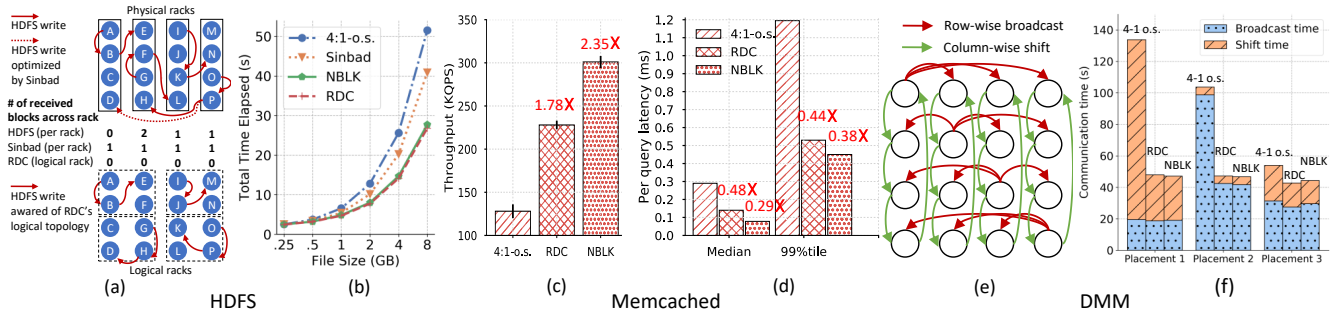

Figure 8: Application performance improvements of RDC compared with the 4:1 oversubscribed network (4:1-o.s.), 4:1-o.s. network powered with Sinbad [45], or the non-blocking network (NBLK). (a) The HDFS write traffic pattern and the number of received blocks per rack. (b) The HDFS transfer time. (c)-(d) Memcached query throughput and latency. (e) The DMM traffic pattern. (f) Average shift time and broadcast time.

and 16 racks overall. The 16 ToR switches are connected to a single *agg.* switch with tunable oversubscription ratios. Results in §5.2 are obtained via simulation.

5.1 Real-world applications

First, we show how RDC can improve the performance of real-world applications for each of its use cases.

HDFS. We set up an HDFS cluster with 16 *datanodes* across 4 racks and 1 *namenode*, with a replication factor of 3 and a block size of 256 MB. All data blocks are cached in the RAM disk to prevent the hard drive from being the bottleneck. The 16 clients initiated concurrent write requests to 16 HDFS files, respectively. According to the default HDFS data block placement policy, when writing a data block to a datanode, a replica of the block will be placed on the same rack of the original copy, and another replica is placed on a remote rack for resilience (Fig. 8(a)). Therefore, a write operation generates an intra-rack flow and an inter-rack flow.

HDFS can localize all the inter-rack traffic (for storing replicas) by using both proactive RDC and network-aware replica placements. Fig. 8(b) shows the performance gain with RDC and compares it with the non-blocking network (NBLK) and an advanced bandwidth-centric replica placement solution, Sinbad [45]). Sinbad keeps track of the paths and links to reach the replicas within the most recent period and assigns the next replica to the least-utilized paths in the recent period. Therefore, Sinbad does not reduce cross-rack traffic, but can relieve bottlenecks at network links by load balancing as shown in Fig. 8(a). Specifically, it detects traffic imbalance for transferring inter-rack replicas and aims to utilize all links roughly equally—i.e., each rack hosts one replica. In the results, we can see that Sinbad improves the total time for HDFS writes, but still underperforms the NBLK network. In contrast, RDC allows the HDFS to regroup servers directly. Moreover, with the new topology, HDFS could change the replica placement scheme to keep all traffic within the logical racks but satisfy fault-tolerance constraints at the same time, as shown in Fig. 8(a). HDFS with RDC achieves similar performance as the NBLK network, reducing the total time to 0.59× on average, compared to the original topology and

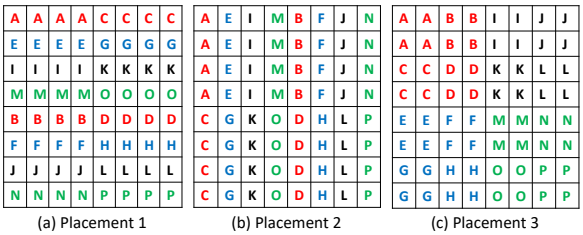

Figure 9: Three different placements for DMM. A-P represent 16 servers and A-D, E-H, I-L, M-P belong to four physical racks separately.

placement policy.

Memcached. We then configured Memcached [23] servers on two racks, and issued read/write requests from two other racks. This emulates the scenario where clients in one pod access cache servers in another pod. Our workload has a) 200 k key-value pairs uniformly distributed across 8 servers, b) a 99%/1% read/write ratio, and c) 512 byte keys and 10 KB values. We adopted a Zipfian query key distribution of skewness 0.99 similar to previous works [31, 93], which led to a load imbalance ratio of ∼1.8 on the server racks.

By reallocating the servers with hot keys equally onto every ToR, RDC improves the query throughput by 1.78× on average and reduces the median latency to 0.48× as shown in Fig. 8(c)-(d). These improvements are close to what a non-blocking network could achieve. RDC also cuts the tail latency significantly, for which network congestion is a major cause [30, 117]. Since in the baseline setting, ToR uplink can easily get congested when several hot keys are coincidentally located in the same rack, even if the overall uplink utilization is low. In contrast, RDC can observe the traffic patterns due to the hot keys, and spread the servers hosting these keys to different racks. This reduces the peak uplink utilization.

OpenMPI DMM. We set up a 16-node OpenMPI cluster across 4 racks and implemented a commonly used DMM algorithm [54] with 64 processes. Matrices are divided into 64 blocks (submatrices). Each server has 4 processes to form an 8 × 8 process layout. Then in each iteration, it performs a "broadcast-shift-multiply" cycle where a process a) broadcasts submatrix row-wise, b) shifts submatrices column-wise, and

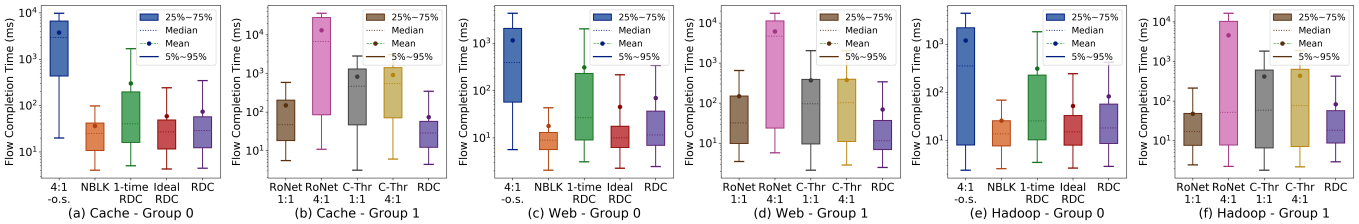

Figure 10: Performance comparison with RDC. Group 0 shows that RDC delivers performance improvements by dynamically reconfiguring the network and achieves similar performance as NBLK; group 1 shows that RDC outperforms alternative designs by benefiting a higher amount of inter-rack traffic.

c) multiplies submatrices as shown in Fig. 8(e). We consider three placements for processes: 1) Fig. 9(a): places them row-wise (no cross-rack traffic for broadcast), 2) Fig. 9(b): places them column-wise (no cross-rack traffic for shift) and 3) Fig. 9(c): places them in a mixed manner, considering both broadcast and shift traffic across racks.

By dynamically configuring the topology for different phases during DMM, RDC shrinks the communication time as well as the end-to-end execution time. Fig. 8(f) shows that RDC improves the overall communication time for placements 1, 2, and 3 by 3.9×, 2.3×, and 1.26× respectively compared to a static 4:1 oversubscribed network, achieving almost the same performance with the NBLK network. Since the applications have evolving traffic patterns, no static process placement is consistently optimal. Out of the three placements, placement 3 jointly minimizes the cross rack traffic for both communication patterns in DMM, outperforming the other two strategies.

5.2 Performance at scale

Next, we evaluate the reactive RDC pods at the data center scale using the packet-level simulator. Our baselines are a) a static non-blocking network (NBLK), b) a static network with 4:1 oversubscription (4:1-o.s.), c) RDC with future traffic-demand information (Ideal RDC), d) a 4:1-o.s. network that applies RDC's reconfiguration algorithm only once over the entire traffic trace (One-time RDC). e) a hybrid network—like C-Through [110] with 16 4:1/1:1 oversubscribed reconfigurable circuit ToR-pair links in addition to a 4:1-o.s. network, which is similar to Firefly [66], and ProjecToR [59] in terms of performance. f) a novel circuit-core network—RotorNet [86] with 4:1/1:1 oversubscribed ToR uplink bandwidth. Note C-through has the same circuit switching delay as RDC and buffers packets at ToRs during the circuit downtime.

We used the Cache, Web, and Hadoop traffic traces from Facebook. Since the original traces do not contain flow-level information, we generated flow-level traffic based on the sampled packet traces from [99]. Specifically, we inferred the source/destination servers of the flows from the trace, and simulated flow sizes and arrival times based on Figures 6 and 14 in the same Facebook paper. The Cache workload has an average flow size of 680 KB, with 87% being inter-rack. The Web workload has an average size of 63 KB with 96% inter-rack. For the Hadoop workload, the average size is 67.18 KB

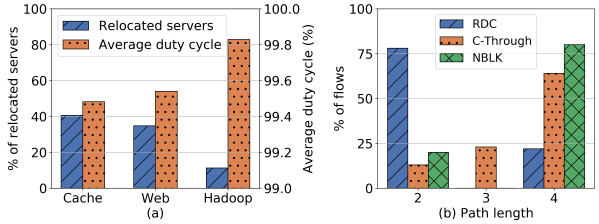

Figure 11: RDC's average circuit duty cycle is >99% even with frequent reconfigurations; RDC has an average path length 35% shorter than NBLK.

but only 60% is inter-rack traffic. All traffic traces last for 30s in the simulation, and RDC's reconfiguration period is 1s.

Fig. 10 shows the boxplot of flow completion times (FCT) for RDC and the baselines using the three traces. We observe that RDC reduces the median FCT by more than an order of magnitude compared to 4:1-o.s. network. Applying RDC's traffic localization algorithm once can bring some improvements on the median FCT but not as significant as RDC and NBLK, since the traffic pattern changes during the simulation. We found that one root cause for the performance improvements is due to TCP dynamics—severe inter-rack congestion causes consecutive packet losses and TCP becomes very conservative in increasing its sending rate. More importantly, we observe that RDC with future knowledge of traffic demands performs consistently close to the non-blocking network, which again demonstrates the power of a rackless network. Without future knowledge, RDC can still achieve similar performance as NBLK with a slightly longer median FCT, because the cache workload is largely stable at the time scale of seconds, similar to that in the Database workload in the original traces. As for other solutions, C-Through's average FCTs are at least 3.21× higher than RDC. Because although C-Through adds extra inter-rack bandwidth, it is provisioned for only 16 ToR pairs. As the traffic traces that motivate our RDC design have more than 16 intensively-communicating ToR pairs (see the heatmap in Fig: 1), C-Through falls short in relieving inter-rack congestion even after enlarging the bandwidths of 16 extra links. 4:1-o.s. RotorNet has the same total uplink bandwidth as RDC, but its performance is much worse than RDC. The non-blocking RotorNet is 2× and 2.17× slower than RDC on the Cache and Web traces; only for the Hadoop traces, it can reduce RDC's average FCT to 0.576×. Since RotorNet provides a dedicated

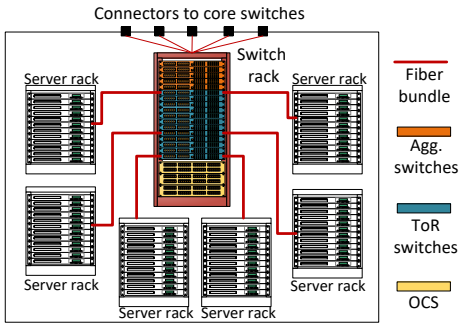

Figure 12: Packaging design of an RDC pod.

link between each ToR pair, if the traffic is skewed between some ToR pairs, it cannot achieve the best performance. So only the Hadoop trace, which has only 60% inter-rack and is quite evenly distributed across different ToRs, enjoys better performance on non-blocking RotorNet.

Fig. 11(a) shows that the average number of servers being relocated in each epoch is different across traces. The duty cycle is an important metric in optical networks to represent the percentage of time that an optical link is up and available for transmission. Assuming one reconfiguration per second, the lowest circuit duty cycle of RDC is 99.2% in theory (details about downtime in §5.4); since not all servers will be relocated in practice, the average circuit duty cycle for all transmissions can be as high as 99.83%. Fig. 11(b) shows the distribution of flow path lengths for RDC, C-Through, and NBLK. (Two C-Through settings have the same distribution; NBLK, 4:1-o.s., and RotorNet also have the same distribution). An intra-rack flow has path length 2 in all networks; and an inter-rack flow has path length 4 in RDC, NBLK, and 4:1-o.s.; the path length could vary in C-Through—3 for the circuit path and 4 for the normal packet-switched path. Overall, RDC localizes more than 70% of the inter-rack traffic and achieves an average path length of 0.75× of C-Through and 0.65× of NBLK.

5.3 Packaging, power, and capital cost

Packaging. Fig. 12 shows the packaging design of an RDC pod, which is somewhat different from that of a traditional pod. RDC has a central switch rack dedicated to hosting ToRs, *agg.* switches, and OCSes. Server racks are connected to OCSes via fiber bundles to reduce wiring complexity. On the central rack, ToRs are connected to OCSes and *agg.* switches using short fibers and cables, respectively. *Agg.* switches provide similar connectivity to core switches outside the pod, just like in traditional data centers. To ensure that centralized switch placement has similar reliability as traditional switch placement, backup power supplies are employed. Similar to the existing modular data centers, RDC supports incremental expansion by adding RDC pods.

Power and capital cost modeling. We show that RDC is more economical by comparing the power and capital cost between RDC and NBLK, at 400 Gbps data rate. They both

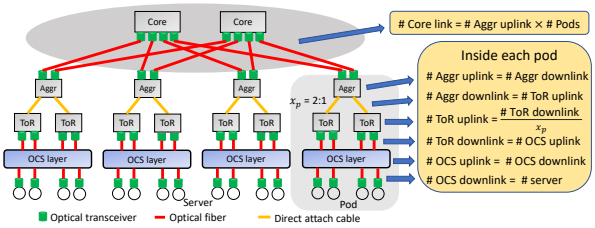

Figure 13: RDC example with detailed components, including the governing equations for power and capital cost model.

Components	Power (Watt)	Cost (USD)	Relative count	
			RDC (x_p : 1)	NBLK
Ethernet port [16]	40.6	312.5	$1+4/x_p$	5
Optical transceiver [15]	10	799	$2+2/x_p$	4
Inter-rack fiber [20]	0	6.9	$1+1/x_p$	2
Intra-rack fiber [21]	0	4.9	1	0
DAC [17]	1.5	249	$1/x_p$	1
OCS port [8]	0.14	400 [52]	2	0

Table 1: Power/cost data and relative count of the components for RDC (x_p:1 o.s.) and NBLK at 400 Gbps.

consist of the following types of networking components: a) 400 Gbps Ethernet port, b) 400 Gbps Optical transceiver, c) inter-rack duplex single-mode fiber (average length 10m), d) intra-rack duplex single-mode fiber (average length 3m), e) 400 Gbps Direct Attach Cables (average length 3m) and f) OCS port. NBLK network can use DAC to directly connect the server-ToR downlinks, while RDC needs fiber-optic cables along with optical transceivers both at the server and ToR ends to connect the OCSes in between.

We assume that RDC has an x_p : 1 oversubscription above the ToR level. Fig. 13 demonstrates a 4-pod RDC network (total 16 servers) with component-level details (where $x_p = 2$) and shows the governing equations to find the component counts across the network. Based on our modeling, given the number of servers and pods are the same, the relative component count for RDC and NBLK network only depends on x_p. Table 1 shows the recent power and cost values of different components along with their relative count for RDC and NBLK network (400 Gbps). On one hand, the power consumption values of the network components are fundamental and well-documented in datasheets. On the other hand, the component cost can vary based on sales volume, and since we have no proprietary industry pricing figures, we do a "best-effort" calculation based on readily available retail pricing (in other words worst-case or no-discount pricing) for all components, so at least it is somewhat objective and unbiased. We consider $400 to be the OCS per-port cost, the worst-case price adopted from a recently reported article from Microsoft [52]. Readers should be aware of the limitation of this pricing assumption and take the capital cost results for general guidance only.

As shown in the governing equations in Fig. 13, an x_p : 1 RDC network with s servers have s ToR downlink ports, $\frac{s}{x_p}$ ToR uplink ports, $\frac{s}{x_p}$ *agg.* switch downlink ports, $\frac{s}{x_p}$ *agg.*

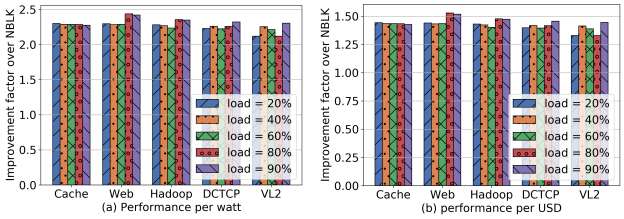

Figure 14: a) RDC (4:1-o.s.) has 2.1-2.4× improvements in performance per watt than NBLK at 400 Gbps data rate; b) RDC (4:1-o.s.) has 1.3-1.5× improvements in performance per dollar than NBLK at 400 Gbps data rate, assuming worst-case component pricing.

switch uplinks ports and $\frac{s}{x_p}$ core switch ports; leading to $(s+\frac{4s}{x_p})$ Ethernet ports in total. Consider a traditional NBLK (fat-tree) network with the same number of servers and pods, where the number of Ethernet switch ports at each layer is the same as the number of servers. This leads to a total of $5s$ (using $x_p = 1$ in RDC) Ethernet ports. Hence, the relative Ethernet port count is $(1+\frac{4}{x_p})$ to 5 (see Table 1). Similar calculation can be applied to other components as well.

Power efficiency. A 4:1-o.s. RDC network consumes 2.29× less power than an NBLK network considering 400 Gbps data rate. RDC significantly improves the performance (median FCT) per watt compared to that of NBLK for diverse traffic patterns across different network loads, as shown in Fig. 14(a). We use five different production traces i.e., Cache [99], Web [99], Hadoop [99], DCTCP [29] and VL2 [62]. For median FCT, RDC has $2.1\times - 2.4\times$ improvements in performance per watt compared to NBLK. RDC also significantly reduces the power consumption of the network because it requires fewer power-hungry packet switches in the core. The optical circuit switch at the RDC edge consumes very little power since it only directs the incoming photon beams using mirror rotation or diffraction.

Capital cost. We again emphasize that readers should take this "best-effort" cost analysis for general guidance only. A 4:1-o.s. RDC network costs 1.4× less than an NBLK network at 400 Gbps. Using the same five production traces, we observe that RDC has $1.3\times - 1.5\times$ improvements in performance (median FCT) per dollar compared to NBLK, as shown in Fig. 14(b). We also estimate OCS per-port cost which would let 4:1-o.s. RDC has an equal performance per dollar as NBLK: it ranges from $1000-$1300.

5.4 RDC reconfigurations

To have a deeper understanding of RDC, we break down this analysis into the effectiveness study of the demand estimation algorithm, the non-disruptive control loop before the reconfiguration, and the hardware transient state during the reconfiguration.

Effectiveness of demand estimation algorithm To show the effectiveness of our demand estimation algorithm, we examine how our heuristic interacts with consecutive topology reconfigurations, by an in-depth study of traffic localization.

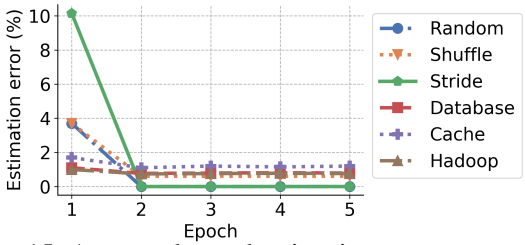

Figure 15: Average demand estimation error over multiple consecutive epochs (epoch duration: 10s).

We use the same packet-level simulation as §5.2 to illustrate this demand estimation technique. For each simulation, RDC performs traffic localization and reconfigures the topology once every 10s according to the algorithm detailed in §4.3. The senders and receivers of elastic flows are determined based on a chosen traffic pattern, while non-elastic flows are generated with randomly chosen senders and receivers with a data rate < 10Mbps. The ratio between the number of non-elastic and elastic flows is 10:1. Besides the three trace-derived traffic patterns – Cache, Web, and Hadoop, we also test three synthetic traces as follows. Each traffic pattern remains the same throughout the simulation.

Random: Each host i sends a flow to one of the other hosts with uniform random probability;

Shuffle: Each host i sends to a set of 31 other hosts with indexes $(i+j*16)\%num\_hosts, j \in [1..31]$;

Stride: Each host i sends a flow to another host with index $(i+32)\%num\_hosts$;

Fig. 15 shows the average demand estimation errors over five consecutive reconfiguration epochs for all server pairs. We observe that while the initial demand estimation errors can be moderately high (10%), the errors decrease as the network reconfigures to adapt to the traffic pattern in subsequent epochs. In the first epoch, many elastic flows congest the oversubscribed network core. As a result, their flow counters can be small and they could be misidentified as non-elastic flows. However, as RDC adapts the topology to localize the identified elastic flows, fewer elastic flows are transmitted across racks, congestion in the network core is reduced, and thus more elastic flows are correctly identified. For example, because the elastic flows in the stride pattern are eventually all localized within racks, the demand estimation errors for these elastic flows drop to nearly zero. Therefore, we can see that the Hedera technique is well-suited to RDC—reconfiguring the topology to suit the traffic patterns helps improve the accuracy of demand estimates for the next epoch.

Non-disruptive control loop. Next, we evaluate the latency of the RDC control loop, which includes four components: 1) collecting flow counters, 2) estimating traffic demands, 3) computing new topologies, and 4) modifying forwarding rules. This latency will affect how fast RDC can respond to changing traffic patterns. Note that the reactive RDC uses all four components; for proactive RDC using traffic demand matrix,

#Racks	4		8		16		32	
	TL	ULB	TL	ULB	TL	ULB	TL	ULB
Counter collection	10.6	2.3	21.3	2.6	42.6	3.4	85.1	4.5
Demand estimation	10.8	0.7	24.9	1.1	80.6	1.3	310.6	1.7
Topo. computation	7.8	0.1	28.2	0.1	40.3	0.3	69.3	0.6
Rule installation	32.5	30.6	45.6	30.8	75.6	41.4	147.6	70.6
Proactive - Command	**32.5**	**30.6**	**45.6**	**30.8**	**75.6**	**41.4**	**147.6**	**70.6**
Proactive - Demand	**40.3**	**30.7**	**73.8**	**30.9**	**115.9**	**41.7**	**216.9**	**71.2**
Reactive	**61.7**	**33.7**	**120**	**34.6**	**239.1**	**46.4**	**612.6**	**77.4**

Table 2: Control loop latency breakdown (ms) for traffic localization (TL) and uplink load-balancing (ULB).

only steps 3 & 4 will be executed; for proactive RDC with direct configuration command, only the last step is required.

To obtain these results, we ran a set of experiments using different numbers of racks, with 32 servers per rack, using the traffic patterns from the Facebook traces. The ToR switches are connected to a single *agg.* switch. Since our testbed only has four ToR switches, we emulated more ToR switches using servers and ensured that each server has the same latency for collecting counters and installing routing rules as a physical ToR switch. The number of forwarding rules to be installed is bounded by 32 for the ToR switches and 32 $\times$ #racks for the *agg.* switch. And the number varies depending on the traffic patterns and may be different across switches. The overall rule installation delay is determined by the slowest switch, which has the most number of changes.

Table 2 breaks down the control loop latency for traffic localization (TL) and uplink load-balancing (ULB) use cases. Overall, reactive RDC's non-disruptive control loop latency before reconfiguration is 612.6ms for TL and 77.4ms for ULB, which are on similar timescales with state-of-the-art traffic engineering techniques [38]. Whereas, proactive RDC can reduce this control loop delay to 147.6 ms and 70.6 ms respectively. Since RDC aims to reconfigure the network at large timescales (e.g., seconds or longer), this control loop is efficient enough to be practical. Note that all the above numbers are obtained with our own testbed. With the cutting-edge high-performance switch hardware [9, 16, 18], the latency can be further reduced to support more frequent reconfiguration.

Reconfiguration transient state. It is important to observe that a circuit reconfiguration in RDC happens only when needed, and for the vast majority of the time, circuits are continuously active. When a reconfiguration happens to a circuit, a transient disruption to that circuit does occur. For example, AWGR and star-coupler-based OCSes are becoming popular as tunable lasers with sub-nanosecond wavelength switching are being fabricated [33, 35, 48, 49, 58, 77]. Considering 400 Gbps link speed and 1 ns of switching delay, only 50 bytes of traffic will be buffered or dropped during the transient phase. Also, 2D-MEMS based OCSes are available, having a reconfiguration delay of few microseconds [96]. Even with a relatively slow OCS in our testbed, our experiments show that RDC provides large performance benefits.

6 Related Work

Various DCN proposals recognize the need for serving dynamic workloads and provision bandwidth on demand with reconfigurable topologies. It can be achieved by adding extra bandwidth to the network by creating ad hoc links at runtime [53, 74, 80, 110, 118], but they mostly focus on providing reconfigurable topology at the rack level, assuming skewed inter-rack traffic. RDC, however, alleviates the reliance on such an assumption and achieves higher performance without adding extra bandwidth. Another line of work constructs an all-connected flexible network core with a high capacity [32, 44, 84–86, 96], but they mostly focus on rack-level rather than edge reconfigurability. Flat-tree [115] is an architecture proposal with partial edge-level reconfigurability, which enables DC-wide reconfigurability by dynamically changing the topology between Clos [26] and random graph [106]. However, the topology modes are limited and only suitable for generally expected workload patterns, e.g., rack-, pod-, or DC-local. Our workshop paper [111] does not contain a detailed design, implementation, or evaluation.

Besides architectural solutions, there are also numerous works that improve flow performance by optimizing task placements. For instance, Sinbad [45] selectively chooses data transfer destinations to avoid network congestion; ShuffleWatcher [25] attempts to localize the shuffle phase of MapReduce jobs to one or a few racks; Corral [72] jointly places input data and compute to reduce inter-rack traffic for recurring jobs. However, these works all have important drawbacks as they only optimize data transfer for one or two stages of job executions. As we noted before, the traffic pattern may change in different stages of a job's lifetime. Also, there is a set of research projects that improve network performance at the upper layers in the stack. Optimized transport protocols (e.g., DCTCP [28], MPTCP [97]) and traffic engineering techniques (e.g., Hedera [27], MicroTE [38], Varys [47]) can improve flow performance for many applications.

7 Conclusion

The rackless data center (RDC) is a novel network architecture that logically removes the static rack boundaries, using circuit switching to achieve topological reconfigurability at the edge. In this architecture, servers in different physical racks can be grouped into the same locality group at runtime based on traffic patterns. By co-designing the network architecture and the control systems, RDC can benefit a wide range of realistic data center workloads. Our evaluations with testbed and simulation setups show that RDC leads to substantial performance benefits for real-world applications.

Acknowledgment

We thank our shepherd George Porter and the anonymous reviewers for their valuable feedback. This research is partly sponsored by the NSF under CNS-1718980, CNS-1801884, and CNS-1815525.

References

[1] Top of rack vs end of row data center designs. http://bradhedlund.com/2009/04/05/top-of-rack-vs-end-of-row-data-center-designs/, 2009.

[2] S320 photonic switch hardware user manual. http://www.calient.net/wp-content/uploads/downloads/2013/04/CALIENT-S-Series-Photonic-Switch-Hardware-User-Manual-Rev-A-460xxx-00-v10.pdf, 2012.

[3] Introducing data center fabric, the next-generation facebook data center network. https://code.fb.com/production-engineering/introducing-data-center-fabric-the-next-\generation-facebook-data-center-network, 2014.

[4] Facebook network analytics data sharing. https://www.facebook.com/groups/1144031739005495/, 2016.

[5] Apache hadoop: Fair scheduler. https://hadoop.apache.org/docs/r2.7.4/hadoop-yarn/hadoop-yarn-site/FairScheduler.html, 2017.

[6] Sailing through the data deluge. https: //rockleyphotonics.com/wp-content/uploads/2019/02/Rockley-Photonics-Sailing-through-the-Data-Deluge.pdf., 2019.

[7] 25.6 tb/s strataxgs broadcom tomahawk 4 ethernet switch series. https://www.broadcom.com/products/ethernet-connectivity/switching/ strataxgs/bcm56990-series, 2020.

[8] 320x320 3D MEMS optical circuit switch. https://www.calient.net/products/edge640-optical-circuit-switch/, 2020.

[9] 32*100Gbps Ethernet Switch. https://www.fs.com/products/107081.html, 2020.

[10] Apache hadoop. http://hadoop.apache.org, 2020.

[11] Apache hadoop: Capacity scheduler. https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html, 2020.

[12] Apache hadoop yarn project. http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html, 2020.

[13] Hdfs architecture. https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html, 2020.

[14] Specifying data center it pod architectures. https://www.apc.com/salestools/WTOL-AHAPRN/WTOL-AHAPRN_R0_EN.pdf, 2020.

[15] 100G PAM4 850nm 100m optical transceiver module. https://www.fs.com/products/93264.html, 2021.

[16] 32*400Gbps Ethernet Switch. https://www.fs.com/products/96982.html, 2021.

[17] 400G QSFP-DD Passive Direct Attach Copper Twinax Cable (3m). https://www.fs.com/products/82454.html, 2021.

[18] Barefoot tofino. https://www.barefootnetworks.com/products/brief-tofino, 2021.

[19] Core and pod data center design. http://go.bigswitch.com/rs/974-WXR-561/images/Core-and-Pod%20Overview.pdf, 2021.

[20] Duplex single mode optical fiber cable (10m). https://www.fs.com/products/40203.html, 2021.

[21] Duplex single mode optical fiber cable (3m). https://www.fs.com/products/40193.html, 2021.

[22] Ibm prefabricated modular data center. https://www.ibm.com/us-en/marketplace/prefabricated-modular-data-center, 2021.

[23] Memcached. https://memcached.org, 2021.

[24] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 265–283, 2016.

[25] Faraz Ahmad, Srimat T Chakradhar, Anand Raghunathan, and TN Vijaykumar. Shufflewatcher: Shuffle-aware scheduling in multi-tenant mapreduce clusters. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 1–13, 2014.

[26] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *ACM SIGCOMM Computer Communication Review*, volume 38, pages 63–74. ACM, 2008.

[27] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: dynamic flow scheduling for data center networks. In *NSDI*, volume 10, pages 89–92, 2010.

[28] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 Conference*, pages 63–74, 2010.

[29] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 conference*, pages 63–74, 2010.

[30] Mohammad Alizadeh, Abdul Kabbani, Tom Edsall, Balaji Prabhakar, Amin Vahdat, and Masato Yasuda. Less is more: trading a little bandwidth for ultra-low latency in the data center. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation NSDI 12)*, pages 253–266, 2012.

[31] Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Song Jiang, and Mike Paleczny. Workload analysis of a large-scale key-value store. In *ACM SIGMETRICS Performance Evaluation Review*, volume 40, pages 53–64. ACM, 2012.

[32] Paraskevas Bakopoulos, Konstantinos Christodoulopoulos, Giada Landi, Muzzamil Aziz, Eitan Zahavi, Domenico Gallico, Richard Pitwon, Konstantinos Tokas, Ioannis Patronas, Marco Capitani, et al. Nephele: An end-to-end scalable and dynamically reconfigurable optical architecture for application-aware sdn cloud data centers. *IEEE Communications Magazine*, 56(2):178–188, 2018.

[33] 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*, pages 782–797, 2020.

[34] Hitesh Ballani, Paolo Costa, Istvan Haller, Krzysztof Jozwik, Kai Shi, Benn Thomsen, and Hugh Williams. Bridging the last mile for optical switching in data centers. In *Optical Fiber Communication Conference*, pages W1C–3. Optical Society of America, 2018.

[35] Joshua L Benjamin, Thomas Gerard, Domaniç Lavery, Polina Bayvel, and Georgios Zervas. Pulse: optical circuit switched data center architecture operating at nanosecond timescales. *Journal of Lightwave Technology*, 38(18):4906–4921, 2020.

[36] Theophilus Benson, Aditya Akella, and David A Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 267–280, 2010.

[37] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Understanding data center traffic characteristics. In *Proceedings of the 1st ACM workshop on Research on enterprise networking*, pages 65–72. ACM, 2009.

[38] 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*, page 8. ACM, 2011.

[39] Sergey Blagodurov, Alexandra Fedorova, Evgeny Vinnik, Tyler Dwyer, and Fabien Hermenier. Multi-objective job placement in clusters. In *SC'15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12. IEEE, 2015.

[40] Peter Bodík, Ishai Menache, Mosharaf Chowdhury, Pradeepkumar Mani, David A Maltz, and Ion Stoica. Surviving failures in bandwidth-constrained datacenters. *ACM SIGCOMM Computer Communication Review*, 42(4):431–442, 2012.

[41] Nathan Bronson, Zach Amsden, George Cabrera, Prasad Chakka, Peter Dimov, Hui Ding, Jack Ferris, Anthony Giardullo, Sachin Kulkarni, Harry Li, et al. {TAO}: Facebook's distributed data store for the social graph. In *2013 USENIX Annual Technical Conference (USENIX ATC 13)*, pages 49–60, 2013.

[42] Marco Canini, Petr Kuznetsov, Dan Levin, and Stefan Schmid. A distributed and robust sdn control plane for transactional network updates. In *2015 IEEE conference on computer communications (INFOCOM)*, pages 190–198. IEEE, 2015.

[43] Andromachi Chatzieleftheriou, Sergey Legtchenko, Hugh Williams, and Antony Rowstron. Larry: Practical network reconfigurability in the data center. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 141–156, 2018.

[44] 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, 2014.

[45] Mosharaf Chowdhury, Srikanth Kandula, and Ion Stoica. Leveraging endpoint flexibility in data-intensive clusters. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 231–242. ACM, 2013.

[46] Mosharaf Chowdhury, Matei Zaharia, Justin Ma, Michael I Jordan, and Ion Stoica. Managing data transfers in computer clusters with orchestra. *ACM SIGCOMM Computer Communication Review*, 41(4):98–109, 2011.

[47] Mosharaf Chowdhury, Yuan Zhong, and Ion Stoica. Efficient coflow scheduling with varys. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 443–454, 2014.

[48] Kari Clark, Hitesh Ballani, Polina Bayvel, Daniel Cletheroe, Thomas Gerard, Istvan Haller, Krzysztof Jozwik, Kai Shi, Benn Thomsen, Philip Watts, et al. Sub-nanosecond clock and data recovery in an optically-switched data centre network. In *2018 European Conference on Optical Communication (ECOC)*, pages 1–3. IEEE, 2018.

[49] Kari A Clark, Daniel Cletheroe, Thomas Gerard, Istvan Haller, Krzysztof Jozwik, Kai Shi, Benn Thomsen, Hugh Williams, Georgios Zervas, Hitesh Ballani, et al. Synchronous subnanosecond clock and data recovery for optically switched data centres using clock phase caching. *Nature Electronics*, 3(7):426–433, 2020.

[50] Sushovan Das, Weitao Wang, and TS Ng. Towards all-optical circuit-switched datacenter network cores: The case for mitigating traffic skewness at the edge. In *ACM SIGCOMM 2021 Workshop on Optical Systems (OptSys' 21)*, 2021.

[51] Mauro Dell'Amico and Silvano Martello. Bounds for the cardinality constrained p cmax problem. *Journal of Scheduling*, 4(3):123–138, 2001.

[52] Vojislav Dukic, Ginni Khanna, Christos Gkantsidis, Thomas Karagiannis, Francesca Parmigiani, Ankit Singla, Mark Filer, Jeffrey L Cox, Anna Ptasznik, Nick Harland, et al. Beyond the mega-data center: networking multi-data center regions. 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*, pages 765–781, 2020.

[53] 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. *ACM SIGCOMM Computer Communication Review*, 40(4):339–350, 2010.

[54] G.C Fox, S.W Otto, and A.J.G Hey. Matrix algorithms on a hypercube i: Matrix multiplication. *Parallel Computing*, 4(1):17 – 31, 1987.

[55] Rohan Gandhi, Hongqiang Harry Liu, Y Charlie Hu, Guohan Lu, Jitendra Padhye, Lihua Yuan, and Ming Zhang. Duet: Cloud scale load balancing with hardware and software. *ACM SIGCOMM Computer Communication Review*, 44(4):27–38, 2014.

[56] Peter X Gao, Akshay Narayan, Sagar Karandikar, Joao Carreira, Sangjin Han, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. Network requirements for resource disaggregation. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 249–264, 2016.

[57] Thomas Gerard, Kari Clark, Adam Funnell, Kai Shi, Benn Thomsen, Philip Watts, Krzysztof Jozwik, Istvan Haller, Hugh Williams, Paolo Costa, et al. Fast and uniform optically-switched data centre networks enabled by amplitude caching. In *2021 Optical Fiber Communications Conference and Exhibition (OFC)*, pages 1–3. IEEE, 2021.

[58] Thomas Gerard, Christopher Parsonson, Zacharaya Shabka, Polina Bayvel, Domaniç Lavery, and Georgios Zervas. Swift: Scalable ultra-wideband subnanosecond wavelength switching for data centre networks. *arXiv preprint arXiv:2003.05489*, 2020.

[59] 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 *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 216–229. ACM, 2016.

[60] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *Nsdi*, volume 11, pages 24–24, 2011.

[61] Robert Grandl, Ganesh Ananthanarayanan, Srikanth Kandula, Sriram Rao, and Aditya Akella. Multi-resource packing for cluster schedulers. *ACM SIGCOMM Computer Communication Review*, 44(4):455–466, 2014.

[62] 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 *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, pages 51–62, 2009.

[63] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G Shin. Efficient memory disaggregation with infiniswap. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 649–667, 2017.

[64] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. Bcube: a high performance, server-centric network architecture for modular data centers. *ACM SIGCOMM Computer Communication Review*, 39(4):63–74, 2009.

[65] Chuanxiong Guo, Haitao Wu, Kun Tan, Lei Shi, Yongguang Zhang, and Songwu Lu. Dcell: a scalable and fault-tolerant network structure for data centers. In *ACM SIGCOMM Computer Communication Review*, volume 38, pages 75–86. ACM, 2008.

[66] 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 Computer Communication Review*, volume 44, pages 319–330. ACM, 2014.

[67] 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 *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 29–42, New York, NY, USA, 2017. ACM.

[68] Keqiang He, Junaid Khalid, Aaron Gember-Jacobson, Sourav Das, Chaithan Prakash, Aditya Akella, Li Erran Li, and Marina Thottan. Measuring control plane latency in sdn-enabled switches. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, page 25. ACM, 2015.

[69] Christian E Hopps. Analysis of an equal-cost multi-path algorithm. 2000.

[70] Chang-Hong Hsu, Qingyuan Deng, Jason Mars, and Lingjia Tang. Smoothoperator: Reducing power fragmentation and improving power utilization in large-scale datacenters. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '18, pages 535–548, New York, NY, USA, 2018. ACM.

[71] Michael Isard, Vijayan Prabhakaran, Jon Currey, Udi Wieder, Kunal Talwar, and Andrew Goldberg. Quincy: fair scheduling for distributed computing clusters. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 261–276, 2009.

[72] Virajith Jalaparti, Peter Bodik, Ishai Menache, Sriram Rao, Konstantin Makarychev, and Matthew Caesar. Network-aware scheduling for data-parallel jobs: Plan when you can. *ACM SIGCOMM Computer Communication Review*, 45(4):407–420, 2015.

[73] Xin Jin, Hongqiang Harry Liu, Rohan Gandhi, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Jennifer Rexford, and Roger Wattenhofer. Dynamic scheduling of network updates. *ACM SIGCOMM Computer Communication Review*, 44(4):539–550, 2014.

[74] Srikanth Kandula, Jitendra Padhye, and Paramvir Bahl. Flyways to de-congest data center networks. 2009.

[75] Robert Krauthgamer, Joseph Naor, and Roy Schwartz. Partitioning graphs into balanced components. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pages 942–949. SIAM, 2009.

[76] Maciej Kuźniar, Peter Perešíni, and Dejan Kostić. What you need to know about sdn flow tables. In *International Conference on Passive and Active Network Measurement*, pages 347–359. Springer, 2015.

[77] Sophie Lange, Arslan S Raja, Kai Shi, Maxim Karpov, Raphael Behrendt, Daniel Cletheroe, Istvan Haller, Fotini Karinou, Xin Fu, Junqiu Liu, et al. Sub-nanosecond optical switching using chip-based soliton microcombs. In *Optical Fiber Communication Conference*, pages W2A–4. Optical Society of America, 2020.

[78] Dominique LaSalle and George Karypis. Multi-threaded graph partitioning. In *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 225–236. IEEE, 2013.

[79] T Li, B Cole, P Morton, and D Li. Rfc2281: Cisco hot standby router protocol (hsrp), 1998.

[80] 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 *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 1–15, Seattle, WA, 2014. USENIX Association.

[81] Hongqiang Harry Liu, Xin Wu, Ming Zhang, Lihua Yuan, Roger Wattenhofer, and David Maltz. zupdate: Updating data center networks with zero loss. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 411–422. ACM, 2013.

[82] Vincent Liu, Daniel Halperin, Arvind Krishnamurthy, and Thomas Anderson. F10: A fault-tolerant engineered network. In *Presented as part of the 10th*

USENIX Symposium on Networked Systems Design and Implementation (NSDI 13), pages 399–412, 2013.

[83] Vincent Liu, Danyang Zhuo, Simon Peter, Arvind Krishnamurthy, and Thomas Anderson. Subways: A case for redundant, inexpensive data center edge links. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, page 27. ACM, 2015.

[84] Yunpeng James Liu, Peter Xiang Gao, Bernard Wong, and Srinivasan Keshav. Quartz: a new design element for low-latency dcns. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 283–294. ACM, 2014.

[85] 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. *arXiv e-prints*, page arXiv:1903.12307, Mar 2019.

[86] 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*, pages 267–280. ACM, 2017.

[87] Xiaoqiao Meng, Vasileios Pappas, and Li Zhang. Improving the scalability of data center networks with traffic-aware virtual machine placement. In *2010 Proceedings IEEE INFOCOM*, pages 1–9. IEEE, 2010.

[88] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 15–28. ACM, 2017.

[89] Wil Michiels, Jan Korst, Emile Aarts, and Jan Van Leeuwen. Performance ratios for the differencing method applied to the balanced number partitioning problem. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 583–595. Springer, 2003.

[90] Tal Mizrahi and Yoram Moses. Time4: Time for sdn. *IEEE Transactions on Network and Service Management*, 13(3):433–446, 2016.

[91] Samuel K Moore. Another step toward the end of moore's law: Samsung and tsmc move to 5-nanometer manufacturing-[news]. *IEEE Spectrum*, 56(6):9–10, 2019.

[92] Mihir Nanavati, Jake Wires, and Andrew Warfield. Decibel: Isolation and sharing in disaggregated {Rack-Scale} storage. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 17–33, 2017.

[93] Rajesh Nishtala, Hans Fugal, Steven Grimm, Marc Kwiatkowski, Herman Lee, Harry C Li, Ryan McElroy, Mike Paleczny, Daniel Peek, Paul Saab, et al. Scaling memcache at facebook. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 385–398, 2013.

[94] Vlad Nitu, Boris Teabe, Alain Tchana, Canturk Isci, and Daniel Hagimont. Welcome to zombieland: practical and energy-efficient memory disaggregation in a datacenter. In *Proceedings of the Thirteenth EuroSys Conference*, pages 1–12, 2018.

[95] Rina Panigrahy, Kunal Talwar, Lincoln Uyeda, and Udi Wieder. Heuristics for vector bin packing. *research. microsoft. com*, 2011.

[96] 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, pages 447–458, New York, NY, USA, 2013. ACM.

[97] Costin Raiciu, Sebastien Barre, Christopher Pluntke, Adam Greenhalgh, Damon Wischik, and Mark Handley. Improving datacenter performance and robustness with multipath tcp. *ACM SIGCOMM Computer Communication Review*, 41(4):266–277, 2011.

[98] Mark Reitblatt, Nate Foster, Jennifer Rexford, Cole Schlesinger, and David Walker. Abstractions for network update. *ACM SIGCOMM Computer Communication Review*, 42(4):323–334, 2012.

[99] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. Inside the social network's (datacenter) network. *SIGCOMM Comput. Commun. Rev.*, 45(4):123–137, August 2015.

[100] Liron Schiff, Stefan Schmid, and Petr Kuznetsov. In-band synchronization for distributed sdn control planes. *ACM SIGCOMM Computer Communication Review*, 46(1):37–43, 2016.

[101] Tae Joon Seok, Niels Quack, Sangyoon Han, Wencong Zhang, Richard S Muller, and Ming C Wu. Reliability study of digital silicon photonic mems switches. In *2015 IEEE 12th International Conference on Group IV Photonics (GFP)*, pages 205–206. IEEE, 2015.

[102] Tae Joon Seok, Niels Quack, Sangyoon Han, Wencong Zhang, Richard S Muller, and Ming C Wu. Reliability study of digital silicon photonic mems switches. In *Group IV Photonics (GFP), 2015 IEEE 12th International Conference on*, pages 205–206. IEEE, 2015.

[103] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiying Zhang. {LegoOS}: A disseminated, distributed {OS} for hardware resource disaggregation. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 69–87, 2018.

[104] Vishal Shrivastav, Asaf Valadarsky, Hitesh Ballani, Paolo Costa, Ki Suh Lee, Han Wang, Rachit Agarwal, and Hakim Weatherspoon. Shoal: A network architecture for disaggregated racks. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 255–270, Boston, MA, 2019. USENIX Association.

[105] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, et al. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. *ACM SIGCOMM computer communication review*, 45(4):183–197, 2015.

[106] Ankit Singla, Chi-Yao Hong, Lucian Popa, and Philip Brighten Godfrey. Jellyfish: Networking data centers, randomly. In *NSDI*, volume 12, pages 1–6, 2012.

[107] Rob Stone, Ruby Chen, Jeff Rahn, Srinivas Venkataraman, Xu Wang, Katharine Schmidtke, and James Stewart. Co-packaged optics for data center switching. In *2020 European Conference on Optical Communications (ECOC)*, pages 1–3. IEEE, 2020.

[108] Xiongchao Tang, Haojie Wang, Xiaosong Ma, Nosayba El-Sayed, Jidong Zhai, Wenguang Chen, and Ashraf Aboulnaga. Spread-n-share: improving application performance and cluster throughput with resource-aware job placement. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15, 2019.

[109] Meg Walraed-Sullivan, Amin Vahdat, and Keith Marzullo. Aspen trees: balancing data center fault tolerance, scalability and cost. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, pages 85–96, 2013.

[110] Guohui Wang, David G Andersen, Michael Kaminsky, Konstantina Papagiannaki, TS Ng, Michael Kozuch, and Michael Ryan. c-through: Part-time optics in data centers. In *ACM SIGCOMM Computer Communication Review*, volume 40, pages 327–338. ACM, 2010.

[111] Dingming Wu, Weitao Wang, Ang Chen, and TS Ng. Say no to rack boundaries: Towards a reconfigurable pod-centric dcn architecture. In *Proceedings of the 2019 ACM Symposium on SDN Research*, pages 112–118. ACM, 2019.

[112] Dingming Wu, Yiting Xia, Xiaoye Steven Sun, Xin Sunny Huang, Simbarashe Dzinamarira, and TS Eugene Ng. Masking failures from application performance in data center networks with shareable backup. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 176–190, 2018.

[113] Xin Wu, Daniel Turner, Chao-Chih Chen, David A Maltz, Xiaowei Yang, Lihua Yuan, and Ming Zhang. Netpilot: automating datacenter network failure mitigation. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 419–430. ACM, 2012.

[114] Yiting Xia, Xin Sunny Huang, and T. S. Eugene Ng. Stop rerouting!: Enabling sharebackup for failure recovery in data center networks. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, HotNets-XVI, pages 171–177, New York, NY, USA, 2017. ACM.

[115] Yiting Xia, Xiaoye Steven Sun, Simbarashe Dzinamarira, Dingming Wu, Xin Sunny Huang, and TS Eugene Ng. A tale of two topologies: Exploring convertible data center network architectures with flat-tree. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 295–308, 2017.

[116] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *Proceedings of the 5th European conference on Computer systems*, pages 265–278, 2010.

[117] David Zats, Tathagata Das, Prashanth Mohan, Dhruba Borthakur, and Randy Katz. Detail: reducing the flow completion time tail in datacenter networks. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 139–150. ACM, 2012.

[118] 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. *ACM SIGCOMM CCR*, 42(4):443–454, 2012.

[119] Danyang Zhuo, Monia Ghobadi, Ratul Mahajan, Klaus-Tycho Förster, Arvind Krishnamurthy, and Thomas Anderson. Understanding and mitigating packet corruption in data center networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 362–375, 2017.

[120] Christopher Zimmer, Saurabh Gupta, Scott Atchley, Sudharshan S Vazhkudai, and Carl Albing. A multi-faceted approach to job placement for improved performance on extreme-scale systems. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1015–1025. IEEE, 2016.

A Appendix

This appendix includes more discussions and results.

A.1 The RDC 0/1 updates

Instead of changing packet version, in RDC a switch performs binary changes from VLAN tagging packets to not, and from not VLAN tagging packets to tagging. Assume packets are in VLAN tagging mode before the change and there is a single VLAN tagging rule at the ingress switch for all packets. We first install the new set of rules with lower priority that matches only on destination IPs, note that the more general matching rules always have lower priority. Then, we remove the VLAN tagging rule. The untagged packets in the transient state can immediately match against the new set of rules. Similarly, if packets are not in VLAN tagging mode before the update, we first install the new set of rules matches on both VLAN tag and destination IPs and then install a single VLAN tagging rule for all packets.

Fig. 16 illustrates the update mechanism in RDC, which we call 0/1 update. It uses an example of forwarding state updates on an OpenFlow ToR switch, which has 4 ports. Ports 1 and 2 are connected to servers, ports 3 and 4 are connected to the *agg.* switches. Packet versions are encoded in the VLAN tag. Before the update, packets are first matched against a VLAN table that tags packets with a VLAN ID. Those tagged packets are then matched against the old rules in the forwarding table. During the transient state of rule updating, packets become untagged and can thus immediately match against the new rules without being dropped. The instructions of the forwarding table direct packets to the group table where packets are either directly sent out via an output port or get load-balanced over multiple output ports using the *select* group type. Similarly, an update from the not-tagging mode to the tagging mode also causes no packet loss.

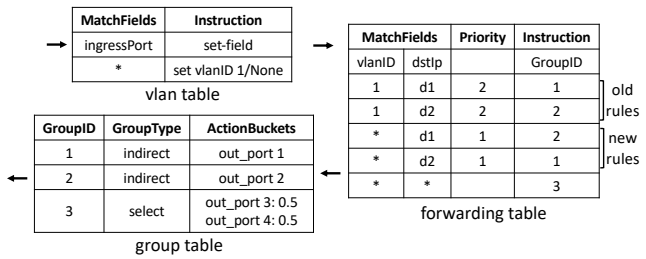

Figure 16: An example of RDC's 0/1 rule update on an OpenFlow-enabled ToR switch.

A.2 Use case: Uplink load-balancing

In §4, we have discussed four use cases for RDC. Among these, uplink load balancing is another reactive RDC algorithm. We discuss this use case in more detail, including the data collection, bias mitigation, and control algorithm. The proactive mode (Use case 3) is simply driven by applications, and the mixed optimization (Use case 4) is also case-specific, so we focus on the reactive algorithm for Use case 2.

Traffic data collection. RDC maintains flow counters on ToRs to monitor the amount of traffic that each server has sent outside the pod. We assume each RDC pod has a unique ID, e.g., an IP address prefix shared by all servers in the pod. Counters are only installed and updated for inter-pod traffic. This can be implemented in the switch using two separate flow tables. The first flow table matches on the destination IP prefix and has only one rule matching the switch's own pod ID. If the first table misses, the second table then matches the 5-tuple and updates the associated counters. Otherwise, the packet skips the second table and goes to the forwarding table. By default, a miss on the second table will not result in packet loss, but a *go-to* action to the rest of the switch pipeline, which avoids traffic disruption when the counter rules change.

Demand estimation. We use a similar technique to estimate the true demand of servers in bottlenecked racks assuming they fair-share the uplink bandwidth. The estimates are obtained by first aggregating the flow counters for each server and then scaling up the per-server demand to reach an aggregate uplink throughput as if the rack is not oversubscribed. We only apply this technique to racks that have been bottlenecked in the collection period to prevent idle racks from being mistakenly treated as hot. This technique keeps the relative order of server traffic load but brings larger quantitative differences among servers, guiding our algorithm to compute better topologies.

Algorithm. For 1-CS RDC, we view the uplink load-balancing problem as a balanced graph partition problem; for multi-CS RDC, we use a heuristic algorithm to obtain the reconfiguration plan. The details of the above two algorithms are included in §A.4.

A.3 Hedera demand estimation algorithm

The pseudocode is shown in Algorithm 1. M is the demand matrix, H is the set of hosts in the network. e_S is the equal share rate of the flows, d_T is the total demand for the destination, and d_S is the demand limited by the sender, $f.rl$ is a flag for a receiver-limited flow, and $< src \rightarrow anydst >$ represents all the flows from the specific source host *src* to any destination host.

A general explanation for this algorithm is expanding the flow demand at the source host with the fair share, and then reducing the demand of some flows according to the capacity of the destination hosts. In each iteration, one or more flows will converge. Eventually, all the flows will converge after multiple iterations [27].

A.4 Topology optimization algorithm details

1. Problem formulation. Assume that the number of racks in a pod is m, each rack has n servers, and each pod has k CS switches to reallocate the server. To keep a record of which server is connected to which ToR switch, we use another

Algorithm 1: Hedera demand estimation [27]

Input: M: traffic matrix, H: the set of all hosts
Output: M: estimated demand matrix

while *some* $M_{i,j}$ *demand changed* **do**
  for *host* $src \in H$ **do**
    $es \leftarrow \frac{1-\sum \text{converged flow demand}}{\text{unconverged flow number}}$, $flow \in < src \rightarrow anydst >$
    for *flow* $f \in < src \rightarrow anydst >$ **do**
      if f *not converged* **then**
        $M_{f.src,f.dst}.demand \leftarrow e_S$
  for *host* $dst \in H$ **do**
    for $f \in < anysrc \rightarrow dst >$ **do**
      $f.rl \leftarrow true$
      $d_T \leftarrow d_T + f.demand$
      $n_R \leftarrow n_R + 1$
    if $d_T > 1$ **then**
      $e_S \leftarrow \frac{1}{n_R}$
      while *some* $f.rl$ *was set to false* **do**
        $n_R \leftarrow 0$
        for $f \in < anysrc \rightarrow dst > \& f.rl$ **do**
          if $f.demand < e_S$ **then**
            $d_S \leftarrow d_S + f.demand$
            $f.rl \leftarrow false$
          else
            $n_R \leftarrow n_R + 1$
        $es \leftarrow \frac{1-d_S}{n_R}$
      for $f \in < src \rightarrow dst > \& f.rl$ **do**
        $M_{f.src,f.dst}.demand \leftarrow es$
        $M_{f.src,f.dst}.converged \leftarrow true$

matrix $C[mn][m]$, if $C[i][j]$ is 1, then server i is connected to ToR j; the server and TOR are not connected if the value is 0. For a valid allocation of the servers, the first constraint is that one server should only be connected to one ToR:

$$\sum_{j=0}^{m-1} C[i][j] = 1, \forall i \in [0, mn) \tag{1}$$

The second constraint is because only a limited number of ports from each ToR are connected to every CS, which is $\frac{n}{k}$. Thus, among all the $\frac{mn}{k}$ servers connected to one CS, only $\frac{n}{k}$ of them can be connected to the same ToR switch:

$$\sum_{x=0}^{m-1} \sum_{y=0}^{\frac{n}{k}-1} C[x \cdot n + i * \frac{n}{k} + y][j] = \frac{n}{k}, \forall i \in [0,k), \forall j \in [0,m) \tag{2}$$

The goal for traffic localization is to localize the inter-rack traffic within a pod as much as possible. Hence, the objective function is to maximize the total amount of localized traffic. Assume that the traffic demand matrix is $D[mn][mn]$, which covers all the server pairs in a pod. Only when two servers are connected to the same ToR, $C[x][j] \cdot C[y][j] = 1$, so that the following equation shows the amount of localized traffic demand:

$$\text{Maximize: } \sum_{x=0}^{mn-1} \sum_{y=0}^{mn-1} \sum_{j=0}^{m-1} C[x][j] \cdot C[y][j] \cdot D[x][y] \tag{3}$$

The goal for uplink load-balancing is to balance the load across all uplinks. Hence, we choose to minimize the maximum load of any uplink for the out-of-pod traffic. Assume that the out-of-pod traffic demand matrix is $U[mn]$. The objective function is:

$$\text{Minimize: } MAX \left\{ \sum_{i=0}^{mn-1} C[i][j] \cdot U[i] \right\}, j \in [0,m) \tag{4}$$

2. Heuristic traffic localization algorithm for 1-CS RDC. For RDC with only 1 circuit switch, the topology optimization problem will just become a graph partition problem. And the new objective function is that we want to partition the vertices (servers) in the graph into groups equally and let the edges (traffic demand) within the groups to be maximum. Assume the traffic demand is a graph $G = (E,V)$, where V is the vertex set (i.e., servers) and E is the edge set. The weight of an edge $e, w(e)$, is the traffic demand between the vertices. To simplify the computation, we do not distinguish the directions of traffic between a server pair, i.e., graph G is non-directional. Our goal is to partition the graph into subgraphs of equal numbers of vertices such that the weighted sum of cross-subgraph edges is minimized. We require partitions of the same size because each ToR must connect to the same fixed number of servers. The balanced graph partitioning problem is NP-hard, but high-quality, efficient heuristics have been proposed in a library *parmetis* [78]. Thus, for the traffic localization problem, we can set the objective to maximize the edge weights insides each group and use the BGP method to solve it.

3. Heuristic uplink load-balancing algorithm for 1-CS RDC. For RDC with only 1 circuit switch, the uplink load-balancing problem will also become a graph partition problem. Our formulation partitions mn number of servers $1, 2, ..., mn$ into m subsets $S_1, S_2, ...S_m$ such that each subset S_j has exactly n servers and the maximum cost of a subset, defined as $\max(\{c(S_j)\})$ is minimized, where $c(S_j) = \sum U[i](i \in S_j)$. Again, we require a balanced partition of the servers because each ToR must host the same number of servers. The problem is also NP-hard when $k > 2$ [51, 89]. We use the same high-quality and efficient heuristics, *parmetis*, to solve this problem by simply changing the objective function to balance the out-of-pod throughput for each group.

4. Heuristic traffic localization algorithm for multi-CS RDC. For RDC with multiple circuit switches, our heuristic firstly groups the servers under the same CS into m bundles equally and maximizes the traffic within each bundle, since all the servers within a bundle should be connected to the same ToR. After obtaining the bundles, for one CS, we only need to assign each of them to a different ToR switch, and the goal is to maximize the traffic demand among bundles under the same ToR switch. In total we have mk bundles, each bundle will be connected to one ToR switch, recorded as $BC[mk][m]$. Moreover, the bundles can be used to calculate an aggregated traffic demand matrix $BD[mk][mk]$. Thus, the simplified traffic localization algorithm can be presented as:

$$\sum_{j=0}^{m-1} BC[i][j] = 1, \forall i \in [0, mk) \tag{5}$$

$$\sum_{x=0}^{m-1} BC[x \cdot m + i][j] = 1, \forall i \in [0, m), \forall j \in [0, m) \tag{6}$$

$$\text{Maximize: } \sum_{x=0}^{mk-1} \sum_{y=0}^{mk-1} \sum_{j=0}^{m-1} BC[x][j] \cdot BC[y][j] \cdot BD[x][y] \tag{7}$$

5. Heuristic uplink load-balancing algorithm for multi-CS RDC. For the heuristic ULB algorithm, again the servers are grouped under the same CS into m bundles equally. And the objective function is to minimize the maximum out-of-pod traffic of each bundle. The idea behind this heuristic is that if each OCS gives balanced out-of-pod traffic to each ToR, then the total out-of-pod traffic from each ToR should also be balanced. The constraints remain the same. Thus, we divide the problem into many sub-problems, and each sub-problem focuses on the servers connected to the same OCS:

$$\sum_{j=0}^{m-1} C[i][j] = 1, \forall i \in [0, mn) \tag{8}$$

$$\sum_{x=0}^{m-1} \sum_{y=0}^{\frac{n}{k}-1} C[x \cdot n + i * \frac{n}{k} + y][j] = \frac{n}{k}, \forall i \in [0, k), \forall j \in [0, m) \tag{9}$$

$$\text{Minimize: } MAX \left\{ \sum_{r=0}^{m-1} \sum_{i=0}^{\frac{n}{k}-1} C[r * n + \frac{n}{k} * s + i][j] \cdot U[i] \right\}, \forall j \in [0, m) \tag{10}$$