Uniform-Cost Multi-Path Routing for Reconfigurable Data Center Networks

Jialong Li
Max Planck Institute for Informatics

Haotian Gong*
The University of British Columbia

Federico De Marchi
Max Planck Institute for Informatics

Aoyu Gong*
EPFL

Yiming Lei
Max Planck Institute for Informatics

Wei Bai
NVIDIA

Yiting Xia
Max Planck Institute for Informatics

Abstract

Reconfigurable data center networks (RDCNs) are arising as a promising data center network (DCN) design in the post-Moore's law era. However, the constantly reconfigured network topology in RDCNs invalidates the assumption of using *hop count* as the *cost metric* for routing, e.g., the *status quo* Equal-Cost Multi-Path routing (ECMP) in traditional DCNs. Unfortunately, existing routing solutions in RDCNs stick to the old assumption and deliver suboptimal performance either high in latency or low in bandwidth efficiency. In this paper, we redefine the cost metric for RDCN routing with *uniform cost* to unify the effects of topology disruption and hop count on latency and bandwidth efficiency. We propose Uniform-Cost Multi-Path routing (UCMP), an ECMP equivalent for RDCNs, where minimizing uniform cost leads flows of various sizes to the right balance between latency and bandwidth efficiency. Our simulation shows that UCMP achieves 53% to 98% lower flow completion time (FCT) and 1.55× bandwidth efficiency compared to the state-of-the-art RDCN routing strategy, and our testbed implementation demonstrates sustainable switch resource usage of UCMP as RDCNs scale.

CCS Concepts

• **Networks** → **Data center networks**; **Network protocols**.

Keywords

Data center networks, optical networks, routing, multi-path routing

ACM Reference Format:
Jialong Li, Haotian Gong, Federico De Marchi, Aoyu Gong, Yiming Lei, Wei Bai, and Yiting Xia. 2024. Uniform-Cost Multi-Path Routing for Reconfigurable Data Center Networks. In *ACM SIGCOMM 2024 Conference (ACM SIGCOMM '24), August 4–8, 2024, Sydney, NSW, Australia.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3651890.3672245

*Work done during the internship at Max Planck Institute for Informatics.

ACM SIGCOMM '24, August 4–8, 2024, Sydney, NSW, Australia

ACM ISBN 979-8-4007-0614-1/24/08
https://doi.org/10.1145/3651890.3672245

1 Introduction

The success of cloud data center networks (DCNs) largely benefited from multi-path routing. The Clos topology of most production DCNs offers numerous *equal-cost* paths to scale up the network capacity [23, 40], enabling Equal-Cost Multi-Path routing (ECMP) [7] and later extensions [5, 8, 21, 46] to utilize the rich bandwidth across parallel paths. However, this classic approach has lost ground in the emerging reconfigurable data center networks (RDCNs), as the slowdown of Moore's law shifts the realm of DCNs towards circuit switching technologies [12–15, 18–20, 24, 27–30, 32, 39, 44].

RDCNs function in a fundamentally different manner than traditional DCNs. As illustrated in Fig. 1a, an RDCN creates reconfigurable circuits between Top-of-Rack switch (ToR) pairs through circuit switches, e.g., optical circuit switches [12, 19, 30], crosspoint switches [18, 27, 39], and free-space optics [20, 24], forming a *direct-connect* topology on the ToR level. The network topology changes throughout time as the circuit switches reconfigure the circuits.

Over the years, RDCNs have evolved from *traffic-aware* to *traffic-oblivious*, driven by the low-latency and low-cost requirements of DCNs. Intuitively, traffic-aware RDCNs construct circuits on-demand to fit the DCN traffic. Yet, the global traffic collection and topology optimization incur millisecond- to second-scale delays, which either prolong traffic flows [13, 14, 20, 28, 32], or require a redundant static DCN for always-on connectivity during circuit reconfiguration [19, 44]. Thus, recent proposals feature traffic-oblivious RDCNs, which rotate among a pre-determined set of topologies in microseconds or sub-microseconds, agnostic to traffic [12, 29, 30]. They deliberately choose a sequence of well-connected graphs, e.g., expander graphs [29], as the topologies to ensure good network connectivity throughout time.

With these time-varying topologies, RDCNs challenge the common beliefs for multi-path routing from the ECMP era. Here, we revisit the design decisions for multi-path routing in traditional DCNs to shed light on new principles needed for multi-path routing in RDCNs.

First, we consider *what is the right cost metric for multi-path routing in RDCNs.* The *cost* in ECMP, and many other routing algorithms, is typically the path *hop count*, because it serves as a reliable proxy metric for routing latency in static networks. In RDCNs, though, the discrete nature of circuits invalidates this assumption. When circuits are reconfigured, packets may wait at intermediate hops for the next circuit to appear. Consequently, paths with the same hop count in an RDCN may exhibit different overall latencies due to

varying circuit-waiting times. *Routing latency in RDCNs thus needs to be redefined, and RDCNs require a new cost metric for multi-path routing.*

Next, we ask *how many multi-paths are appropriate for RDCNs.* We claim that the days of "the more paths, the merrier" are long gone, which assumes the use of ECMP on Clos networks so all paths have the same length. In contrast, the RDCN topologies such as expander graphs lack such parallel paths. Blindly applying multi-path routing, e.g., k-shortest path routing (KSP), would produce long paths with many joint edges that cause load imbalance and congestion [41, 43]. A higher k indicates more but longer paths, and 5 KSP paths ($k = 5$) has been reported to underperform a single path in RDCNs [29]! On the other hand, RDCNs have the natural effect of diversifying paths as they get updated with topology changes, which can be leveraged in the multi-path design. So, we conclude *RDCNs need a small set of carefully selected paths with respect to the reconfigured topologies.*

Finally, we *rethink the norm of routing flows indiscriminately across the multi-paths in ECMP.* Unlike in static networks, short and long flows in RDCNs thrive with paths of different natures, also due to circuit waiting. Short flows, being latency-sensitive, favor paths with immediately available circuits to avoid high circuit-waiting delays, even if such paths are inherently long. Conversely, long flows significantly impact bandwidth efficiency, measured as the reciprocal of the average hop count over all the traffic, since they generate heavy loads for every extra hop encountered. Thus, they favor short paths, despite the longer latency waiting for the circuits to appear. *These distinct routing behaviors of short and long flows should be respected in the design of multi-paths.*

In this paper, we set the first step towards *a principled methodology for multi-path routing in RDCNs* regarding the above requirements. We present *Uniform-Cost Multi-Path routing (UCMP), an ECMP equivalent for RDCNs.* In the design, we *redefine routing latency* for RDCNs taking the circuit-waiting delay into consideration, and propose *uniform cost* as the cost metric to unify the effects of *latency* and *hop count* on short and long flows, optimizing both *FCT* and *bandwidth efficiency.* Minimizing uniform cost guides us to choose a small set of efficient UCMP paths that have high path diversity and coverage as the network reconfigures over time. Flows of various sizes are each assigned to the most appropriate UCMP path with the minimum uniform cost, using our flow bucketing scheme based on flow aging [11] without prior knowledge of the flow size information.

Our principled approach is a giant leap from previous makeshift solutions that simply adapt established routing methods in static networks to RDCNs. Specifically, valiant load balancing (VLB) was repurposed from handling arbitrary traffic in static networks to handling arbitrary topologies in RDCNs [12, 30]. Known as two-phase routing, it first sends packets to random intermediate ToRs and then forwards them to their final destinations. The random choice of intermediate hops causes unpredictable circuit-waiting delay and jeopardizes FCT for short flows. KSP, commonly used for multi-path routing on static expander graphs, has been shown above to have worse performance than single-path routing in RDCNs. To mimic static networks, Opera adopts a topology-routing co-design that reconfigures a subset of circuits at a time and routes traffic on the remaining stable circuits [29]. As detailed in §2.2, Opera's rigid constraint to fully preserve the properties of static networks leads to a considerable percentage of unusable circuits, e.g., 1/6 on average in the simulated setting of the Opera paper [29].

We have implemented UCMP using Intel Tofino2 switches and Mellanox ConnectX-5 NICs, and have conducted extensive evaluations of UCMP with both simulations and testbed experiments. In our setting of a 108-ToR RDCN, a small set of 3.2 UCMP paths on average achieves high coverage, exposing each ToR pair to an average of 47.9 paths over time, and 93.2% of the UCMP paths are edge-disjoint. Under production DCN traffic, UCMP achieves 53% to 98% reduction in FCT and 1.55× bandwidth efficiency compared to the state-of-the-art RDCN routing solution. The advantages of UCMP have been confirmed on the testbed, and the resource consumption of UCMP on switches is within the capacity limit of commercial switch ASICs.

[This work does not raise any ethical issues.]

2 Background

In this section, we provide background information on traffic-oblivious RDCNs (§2.1)—the primary focus of this paper—and discuss routing strategies in traffic-oblivious RDCNs (§2.2).

2.1 Traffic-Oblivious RDCNs

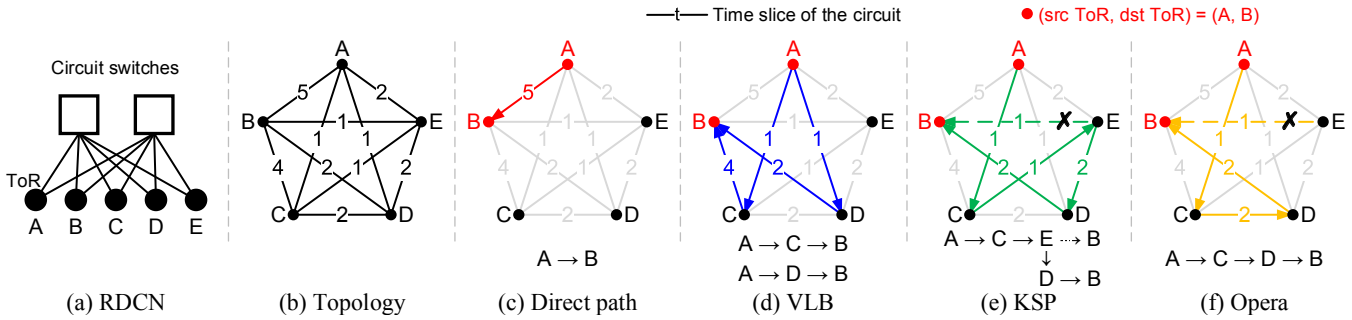
As mentioned in §1, traffic-oblivious RDCNs have gained popularity due to their latency and cost advantages compared to traffic-aware RDCNs. Throughout the rest of the paper, we use the term RDCNs to refer to traffic-oblivious RDCNs unless explicitly specified otherwise.

Fig. 1a illustrates an example (traffic-oblivious) RDCN, where the Top-of-Rack switches (ToRs) are interconnected through a number of circuit switches (2 in the figure). These circuit switches are controlled by an *RDCN controller* to reconfigure the circuits once a fixed duration of time, called a *time slice.* The sequence of ToR-wise circuit connections together with their time slices constitutes a *circuit schedule.* The circuit schedule is pre-determined irrespective of the traffic and repeats every *circuit cycle.* The latest RDCN designs feature microsecond-scale time slices, enabled by modern circuit switching technologies that reconfigure circuits in microseconds or even nanoseconds [12, 29, 30, 32]. A typical circuit cycle contains tens of time slices, resulting in a cycle duration ranging from microseconds to milliseconds.

To maximize the network connectivity, the circuits per time slice form a small-diameter graph, e.g., expander graph, and every ToR pair is guaranteed to have direct circuits at least once per cycle [29, 30]. Thus, RDCNs function as a sequence of cyclically repeating time-varying graphs. Fig. 1b shows the RDCN topology of Fig. 1a, where the time-varying graphs can be presented by a complete graph with the time slices of the circuits annotated on the edges. Hence, a path in an RDCN is specified by $p(src, dst, t_{start})$ from source ToR src to destination ToR dst with the starting time slice t_{start} for an instance of the time-varying graphs, as opposed to simply $p(src, dst)$ in traditional DCNs with static topologies.

$$latency(p) = (t_{end} - t_{start} + 1) \times u \tag{1}$$

The routing latency in RDCNs is also defined differently. Static networks use hop count as a reliable proxy metric for latency. In



(a) RDCN (b) Topology (c) Direct path (d) VLB (e) KSP (f) Opera

Figure 1: **Illustration of the RDCN structure and routing. (a) An example RDCN, (b) the time-varying topology of (a) with the circuit time slices annotated on the edges, (c)-(f) different routing strategies in the RDCN from A to B in time slice $t_{start} = 1$.**

RDCNs, however, paths with the same hop count may exhibit different latencies due to varying waiting times for upcoming circuits. We model latency for a routing path p in an RDCN by the number of elapsed time slices during routing, as shown in Eqn. 1, from the starting time slice t_{start} when the packet arrives at the source ToR to the ending time slice t_{end} when the packet is delivered to the destination ToR, multiplied by the time slice duration u.

The ending time slice, t_{end}, is essentially the time slice of the last-hop circuit on the path, because regardless of the trajectory taken, the delivery time is ultimately determined by the availability of the circuit at the last hop. For example, to route from ToR A to ToR B in time slice $t_{start} = 1$, the latency of path A→B in Fig. 1c is 5 time slices, with $t_{end} = 5$; and the latencies of A→D→B and A→C→B in Fig. 1d, both with 2 hops, are 2 and 4 time slices, respectively.

2.2 Routing in Traffic-Oblivious RDCNs

The most intuitive way of routing is through direct circuits, as shown in Fig. 1c, but the long latency waiting for the circuit, as discussed above, has motivated alternative routing approaches to use more readily available multi-hop paths.

VLB. Valiant Load Balancing (VLB), also known as two-phase routing, has been applied to RDCNs to maximize the throughput, especially for long flows [12, 30]. In *phase 1*, the source ToR distributes packets across randomly selected intermediate ToRs that are connected to the source ToR, e.g., C and D in Fig. 1d. This action achieves a similar effect as packet spraying. In *phase 2*, the intermediate ToRs forward the packets to the destination ToR when direct circuits are available. Despite the optimal throughput harnessing all available circuits, the long circuit-waiting time in phase 2 jeopardizes FCT of short flows. In Fig. 1d, for instance, the FCT is bottlenecked by the high-latency path A→C→B. In the worst case, the latency of a VLB path can be a full circuit cycle, which is as bad as using direct circuits.

KSP. K-shortest path routing (KSP) [6, 41, 43] is widely used for static topologies. When applied to the time-varying graphs in RDCNs, it computes the top k shortest paths for the graph instance in each time slice. In other words, a packet dispatched in time slice t_{start} follows the path determined by the graph corresponding to t_{start}. If the network is reconfigured while the packet is in-flight, it will be rerouted onto a path over the new graph. For example, in Fig. 1e where $k = 1$, the packet following A→C→E→B planned for $t_{start} = 1$ encounters reconfiguration of E→B by the end of time slice 1. It is rerouted at ToR E along E→D→B, which is available in time slice 2. KSP in RDCNs suffers from high hop counts when k is large. This is because small-diameter graphs, such as expanders, lack equal-cost paths, necessitating the use of longer paths when a large number of parallel paths are required. Rerouting caused by circuit reconfiguration exacerbates the problem.

Opera. Opera adopts a topology-routing co-design for RDCNs [29]. It reconfigures a subset of circuits at a time while routing traffic on the remaining stable circuits, treating them as a static topology. Opera ensures that no packets are in-flight during circuit reconfiguration by bounding the ToR buffer size and selecting paths with durations longer than the maximum one-way delay, assuming full ToR buffers in the worst case.

Opera handles short and long flows separately. It classifies flows under 15 MB as short flows and routes them using KSP on the stable circuits. Flows over 15 MB are considered long flows and are routed using VLB. Fig. 1f illustrates the case for short flows. Opera anticipates the upcoming reconfiguration of E→B and excludes the vanilla KSP ($k = 1$) path A→C→E→B due to the one-way delay exceeding the path duration. Instead, it selects the KSP path on the residual graph, excluding the infeasible paths.

Opera aims to fulfill different requirements of short and long flows. However, the hard cutoff of 15 MB is inadaptive to varying traffic workloads and network utilizations. Additionally, Opera avoids rerouting in-flight packets during circuit reconfigurations by imposing overly aggressive constraints that consider the worst-case buffer size. This approach results in unnecessary circuit waste. In the simulated network described in the Opera paper [29], on average, 1/6 of the circuits become unusable. This inefficiency further increases the hop count for KSP on the residual graphs.

3 UCMP Overview

We give an overview of UCMP in this section, where we define uniform cost (§3.1), present our design intuitions (§3.2), and show the path space of UCMP and prior solutions (§3.3).

3.1 Uniform Cost

Intuitively, the goal of routing is to deliver traffic at low latency. In RDCNs, however, the presence of circuit-waiting time means that solely minimizing latency (Eqn. 1) would lead traffic towards immediately available paths, often characterized by high hop counts, thereby compromising bandwidth efficiency. Short flows are latency-sensitive but incur minimal network load even across many hops, whereas long flows exhibit the opposite behavior. Therefore, we

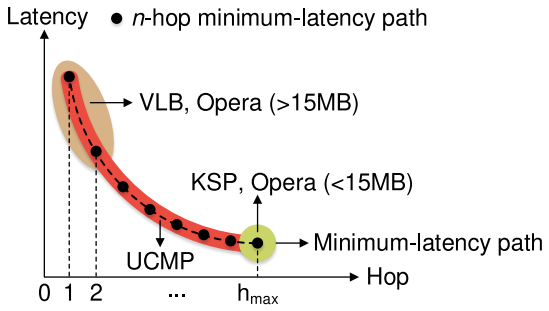


Figure 2: **UCMP overview.**

define uniform cost to unify the effects of *latency* and *hop count* on path selection, to achieve low FCT for short flows and high bandwidth efficiency for long flows.

$$C(p, f) = latency(p) + \alpha \times hop(p) \times size(f)/B \quad (2)$$

The above equation presents the uniform cost $C(p, f)$ for flow f on path p, where $latency(p)$ is as defined in Eqn. 1 and $hop(p)$ is the hop count of the path. As discussed above, we should penalize high hop counts for the reduction of bandwidth efficiency, and the penalty function should be linear with respect to the flow size $size(f)$, as x bytes sent over n hops consume nx bytes of network capacity. We divide $hop(p) \times size(f)$ by the link bandwidth B to convert it into a time quantity that is additive to $latency(p)$. We use a weight factor α to determine the relative importance between the latency and hop count terms. As we will explain in §5.2, α is tunable regarding the network utilization.

With this definition, minimizing uniform cost would guide short flows towards low-latency paths, often with more hops though, and guide long flows towards short paths, albeit with higher latency. Note that uniform cost is specific to both *paths* and *flows*, driven by different behaviors of short and long flows in RDCNs, in contrast to the traditional cost metric of hop count in ECMP that is only specific to paths.

3.2 Design Intuitions

In RDCNs, the microsecond-scale time slices require routing decisions to be done rapidly, and the fixed RDCN topologies enable pre-computation of routing paths. Thus, in UCMP, we opt for *offline path calculation*, i.e., generating UCMP groups with a collection of paths, and *online path assignment*, i.e., mapping flows to different paths in a UCMP group.

However, our uniform cost in Eqn. 2 cannot be used directly for offline path calculation due to the runtime factor of flow size. To address this challenge, we loosely define UCMP groups as sets of *candidate paths* that, without knowing the flow size information, potentially have the minimum uniform cost. Given all the candidate paths, online path assignment can then map each flow to its desired path with the minimum uniform cost for its specific flow size.

A candidate path must satisfy two conditions. (1) It must have the lowest latency among all the paths with the same hop count because the hop count term in Eqn. 2 is fixed for a prospective flow of a particular (unknown) size, making the uniform cost determined by the latency. (2) It must have lower latency than all other candidate paths with fewer hop counts; otherwise, there would exist a path with both lower hop count and lower latency for a prospective flow, resulting in a lower uniform cost.

Following these insights, as the UCMP path space shows in Fig. 2, we calculate the n-hop minimum-latency path for every hop count n, and among these paths, we select the ones with increasing hop count having lower latency. The calculation can stop at h_{max} hops, where the global minimum-latency path is located. There exists no candidate path beyond h_{max}, as any path to its right would have both a higher hop count and higher latency than the global minimum-latency path.

We assign each flow *a single path* in the UCMP group at runtime to avoid packet reordering. This path, as aforementioned, has the minimum uniform cost for its given flow size, to achieve the desirable tradeoff between latency and hop count. Recall from §3.1 that minimizing uniform cost directs short flows to long paths and long flows to short ones. To make UCMP general to cases where flow size information is unattainable, we devise a flow bucketing scheme based on flow aging [11] to enable path assignment without prior knowledge of flow sizes.

This method applies to arbitrary circuit schedules, which, as explained in §2.1, are pre-determined and usually generated in a round-robin manner. UCMP takes the circuit schedule as input and by this means functions as a generic routing solution for traffic-oblivious RDCNs.

3.3 Path Space

In Fig. 2, we compare UCMP's path space with that of prior routing schemes in §2.2. As discussed in §3.2, UCMP's path space includes the n-hop minimum-latency paths, n from 1 to h_{max}, and these paths have decreasing latency as n grows. The h_{max}-hop UCMP path with global minimum latency is essentially the continuous path without circuit waiting.

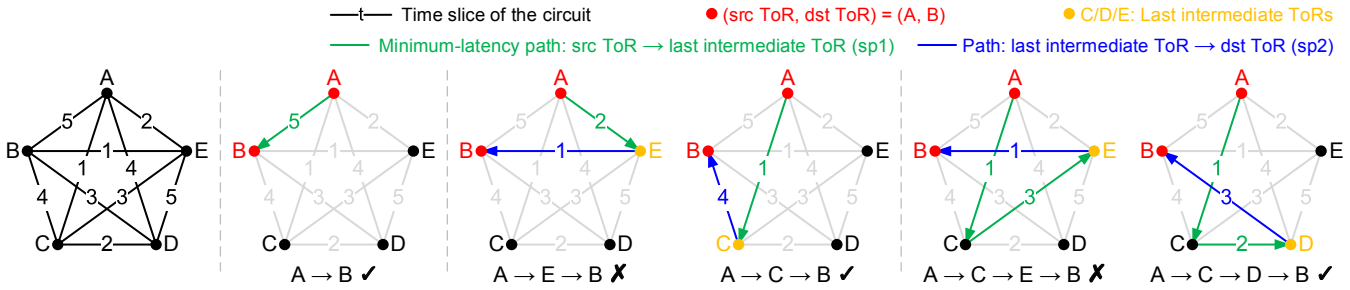
The 1-hop UCMP path is the naive direct path, as shown in Fig. 1c. VLB uses randomly-chosen 2-hop paths and includes the 1-hop paths if direct circuits exist. The VLB path space thus contains the 1- and 2-hop UCMP paths, though it has many other 2-hop paths with higher (random) latency (than the minimum-latency 2-hop UCMP path).

KSP computes k-shortest paths on the topology instance per time slice. It assumes continuous paths on the static topologies, so its space contains the h_{max}-hop UCMP path. It also includes other rerouted paths with longer latency and hop counts. Opera directs flows over 15 MB to VLB paths, and flows under 15 MB to KSP paths. Note that Opera's paths may present higher hop count than KSP's since it avoids using circuits being reconfigured.

In this figure, lower latency and lower hop count indicate better performance. UCMP exhibits the best tradeoff between latency and hop count among all the solutions.

4 Offline Path Calculation

Here, we introduce offline path calculation in detail following the intuitions in §3.2. We present our path computing algorithm (§4.1), show how to obtain h_{max} for the algorithm (§4.2), and give principles for forming UCMP groups (§4.3).



(a) Topology (b) 1-hop minimum-latency path (c) 2-hop minimum-latency path (d) 3-hop minimum-latency path

Figure 3: **Illustration of computing n-hop minimum-latency paths. (a) RDCN ToR-level topology, time slice of each circuit denoted on edges. (b) 1-hop, (c) 2-hop, (d) 3-hop minimum-latency paths from A to B starting at time slice $t_{start} = 1$.**

4.1 n-Hop Minimum-Latency Paths

For simplicity, we explain the offline UCMP path calculation algorithm with an example RDCN illustrated in Fig. 3. The specific algorithm Alg. 1 is left to Appx. A.

Recall from §2.1 that an RDCN path $p(src, dst, t_{start})$ is specific to the *src-dst* ToR pair and the time slice t_{start} when routing starts. According to Eqn. 1, for a specific t_{start}, the latency of the path is solely determined by t_{end} — the time slice of the last-hop circuit. Following this intuition, we divide an n-hop path $p^n(src, dst, t_{start})$ into two sub-paths: $sp_1 = p^{n-1}(src, last, t_{start})$ for the first $n-1$ hops from the *src* ToR to the *last* intermediate ToR and $sp_2 = p^1(last, dst, t_{start})$ for the last hop from the *last* intermediate ToR to the *dst* ToR. We have $latency(p^n) = latency(sp_2)$.

For example, as the right sub-figure of Fig. 3c shows, the 2-hop path p^2 = A→C→B from A to B in time slice $t_{start} = 1$ can be split into sp_1 = A→C and sp_2 = C→B. We have $latency(p^2) = latency(sp_2) = 4$ time slices because the latency of the entire path A→C→B is determined by the last hop C→B, as packets may wait at intermediate ToRs, i.e., C in this case. The left sub-figure, on the other hand, shows an infeasible split sp_1 = A→E and sp_2 = E→B for p^2 = A→E→B. Here, $latency(sp_1)$ (2 time slices) > $latency(sp_2)$ (1 time slice), meaning by the time packets reach E in time slice 2, the circuit E→B would have been gone.

Following the above idea, we can break down the search for the n-hop minimum-latency path $p^n_{min}(src, dst, t_{start})$ into three steps. (1) Find the sp_2 with the minimum latency, which determines the latency of p^n_{min}. (2) Get the corresponding sp_1, which can be computed as the minimum-latency path for $n-1$ hops $p^{n-1}_{min}(src, last, t_{start})$. (3) Ensure the split is feasible with $latency(sp_1) \leq latency(sp_2)$. We thus design a recursive algorithm where the n-hop path depends on the paths with $n-1$ hops, based on step (2).

As Fig. 3b depicts, the 1-hop minimum-latency path is simply the direct circuit A→B, i.e., the only 1-hop path. For 2 hops in Fig. 3c, we first try sp_2 = E→B as it has the minimum latency, but it leads to an infeasible split as explained above. Then we move on to sp_2 = D→B with the next lowest latency of 3 time slices, but the corresponding sp_1 = A→D is still infeasible with a latency of 4 time slices until we finally find sp_2 = C→B that makes the path A→C→B. Similarly, for 3 hops in Fig. 3d, we start from the minimum-latency sp_2 = E→B shown in the left sub-figure, and the sp_1 = A→C→E has been computed as the 2-hop minimum-latency path from A to E. Seeing this split infeasible, we continue the process until getting the path A→C→D→B in the right sub-figure.

4.2 Upper Bound of h_{max}

The algorithm stops computing at h_{max} hops where the global minimum-latency path is, as shown in Fig. 2. It is challenging to calculate the exact value of h_{max} given the diverse RDCN settings, so we develop a probabilistic approach to get its upper bound $\Omega(h_{max})$. We outline the general idea here and leave the detailed method to Appx. B.

We first compute the network diameter, i.e., the hop count of the longest path, for each topology instance of the RDCN. Among them, we choose the maximum value and label it h_{static}. Next, we calculate h_{slice}, the maximum number of hops a packet can traverse within a single time slice, using the propagation delay and transmission delay. The calculation of $\Omega(h_{max})$ can be divided into two cases.

Case I: $h_{slice} \geq h_{static}$. In this case, the global minimum-latency path can be completed within a single slice. Thus, $h_{max} \leq h_{static}$ and we take h_{static} for $\Omega(h_{max})$.

Case II: $h_{slice} < h_{static}$. In this case, the global minimum-latency path may cross different time slices. We define S as the maximum number of time slices the global minimum-latency path spans. Then, $h_{max} \leq h_{slice} \times S$, and S can be calculated by solving the Balls into Bins problem [36], where a packet traversing $h_{slice} \times S$ hops has high probability to reach the destination eventually. So, we set $\Omega(h_{max})$ to be $h_{slice} \times S$.

Our analysis in Appx. B shows that $\Omega(h_{max})$ is at most 15 hops under a wide range of RDCN settings up to 4320 ToRs. The computation of $\Omega(h_{max})$ has $O(N^3)$ time complexity, where N is the number of ToRs. As reasoned in Appx. A, with $\Omega(h_{max})$, the UCMP path calculation algorithm has $O(N^3)$ time complexity and $O(N^2)$ space complexity.

4.3 UCMP Groups

A UCMP group $P(src, dst, t_{start})$ is also defined per *src-dst* ToR pair per time slice t_{start} like the paths. As overviewed in §3.2, paths in a UCMP group have the properties below.

Property 1: $\forall\ p_1 \in P$ *and* $\forall\ p_2 \notin P$, *if* $hop(p_1) = hop(p_2)$, *then* $latency(p_1) < latency(p_2)$.

Property 2: $\forall\ p_1, p_2 \in P$, *if* $hop(p_1) = hop(p_2)$, *then* $latency(p_1) = latency(p_2)$.

Property 3: $\forall\ p_1, p_2 \in P$, *if* $hop(p_1) > hop(p_2)$, *then* $latency(p_1) < latency(p_2)$.

Table 1: **A case of calculating uniform cost. Underlined numbers denote the minimum uniform cost in each column.**

hop(p)	latency(p)	C(p,f_1=1 MB)	C(p,f_2=100 KB)	C(p,f_3=10 KB)
1-hop	60 µs	<u>140</u>	68	60.8
2-hop	15 µs	175	<u>31</u>	16.6
3-hop	10 µs	250	34	12.4
4-hop	5 µs	325	37	<u>8.2</u>

The path calculation algorithm in §4.1 ensures property 1, in that UCMP paths have the minimum latency for each hop count, and property 2 includes parallel solutions in the algorithm. Property 3 regulates UCMP paths with different hop counts: the latency decreases as the hop count grows. So, given the n-hop minimum path for each n, we filter out the ones that violate property 3 and obtain the UCMP group.

The paths thus selected exhibit the path space in Fig. 2. A UCMP group contains one minimum-latency path per hop count or a few if parallel solutions exist. In Fig. 5, we demonstrate high path diversity and coverage achieved by this modest set of paths, benefiting from the high percentage of edge-disjoint paths and the reconfiguring topologies that update UCMP groups per time slice.

A potential problem, though, is the limited path options for long flows to reach high throughput. We enable *latency relaxation* in this case. For long flows that initially resolve to 1- or 2-hop paths, we extend their choice to more 2-hop paths with relaxed latencies, as long as the hop count term in uniform cost (Eqn. 2) still dominates. These extended paths bear the same essence as VLB paths but have lower (though relaxed) uniform costs. For example, they allow waiting at the source for lower overall latency, contrary to VLB paths that enforce immediate forwarding in $phase1$.

5 Online Path Assignment

In this section, we introduce the online stage of UCMP, where we assign flows to paths without knowing flow sizes (§5.1), tune the weight factor α (§5.2), and address failures (§5.3).

5.1 Flow Size Buckets

For an upcoming flow, with its flow size, we can calculate straightforwardly the uniform cost of each path in the UCMP group and assign it the path with the minimum uniform cost.

Table 1 shows an example where $\alpha = 1$ in Eqn. 2 and each row presents a path in the UCMP group. The chosen paths with the minimum uniform cost for each flow size are underscored. The results align with our design objective: uniform cost directs long flows to short paths, optimizing bandwidth efficiency (e.g., the 1MB flow on the 1-hop path), routes short flows on long paths with low latency (e.g., the 10KB flow on the 4-hop path), and places middle-sized flows in between to strike a balance between latency and bandwidth efficiency (e.g., the 100KB flow on the 2-hop path).

We design a flow bucketing scheme for path assignment without prior knowledge of the flow size, as flow size information is oftentimes unattainable [11, 42]. Suppose p_1 and p_2 are two paths in a UCMP group with n and $n+1$ hops, respectively. To get the boundary flow size $size(f)$ of a flow f as the turning point from taking p_1 to p_2, we make their uniform costs identical $C(p_1, f) = C(p_2, f)$ and solve the following equation.

$$\begin{aligned} C(p_1, f) &= latency(p_1) + \alpha \times n \times size(f)/B = \\ C(p_2, f) &= latency(p_2) + \alpha \times (n+1) \times size(f)/B \\ size(f) &= B \times (latency(p_1) - latency(p_2))/\alpha \end{aligned} \tag{3}$$

Similarly, we calculate the boundary flow size for every pair of UCMP paths with adjacent hop counts and define flow size buckets with these boundary values. Then we apply flow aging [11] by setting these boundary sizes as the stepping thresholds. According to the trend in Fig. 2, each flow starts off being mapped to bucket 0 for the h_{max}-hop path with the globally minimum latency. It gradually moves leftwards on the figure to shorter paths with increased latency, as it sends more data and gets mapped to higher buckets. In normal cases, bucketing does not cause packet reordering, because flows shift from lower-latency paths to higher-latency paths.

The flow size buckets for a UCMP group are fixed when α is specified. If multiple paths have the same minimum uniform cost, meaning they are tied for minimum latency with the same hop count, these paths share the same flow size bucket. As stated in §3.2, each flow is assigned a single path in the UCMP group. In cases where there is a tie, we break the tie by randomly selecting a path for each flow using its 5-tuple hash value, similar to ECMP. The implementation details of this random flow assignment are described in §6.2.

Unlike ECMP, UCMP experiences hash collisions to a minor degree, thereby avoiding uneven load distribution. Thanks to high path coverage and frequent topology reconfigurations, flows on congested paths will switch to different paths in the next time slice, resulting in a load-balanced network. As shown in Appx. C, UCMP achieves load-balanced performance close to VLB, which is considered the upper bound with randomized packet spraying.

5.2 Tuning Weight Factor α

The weight factor α in Eqn. 2 affects the link utilization. A larger α increases the penalty for long paths and directs flows towards shorter paths to reduce link utilization. In UCMP, we support live tuning of α regarding a desirable link utilization.

$$\alpha \times size(f) = B \times (latency(p_1) - latency(p_2)) \tag{4}$$

Eqn. 3 can be converted easily to the above form. We simply change the function for mapping the flow size to the bucket (left-hand side of Eqn. 4), and the bucket boundaries (right-hand side of Eqn. 4) become fixed values irrelevant to α. In this way, the stepping thresholds for flow aging, namely the bucket boundaries, are pre-determined. New values of α can be broadcast to the hosts to update the mapping function, so flows are mapped to different buckets accordingly.

Network operators can tune α regarding a target link utilization and update it at runtime as traffic evolves. Measurement studies reveal that traffic in production DCNs changes gradually [34, 38], so daily or at most hourly tuning frequency is sufficient in practice. Fig. 10 verifies this method and guides us to set $\alpha = 0.5$ in the simulation for 70% ToR-to-ToR link utilization.

5.3 Failure Recovery

UCMP paths are robust to failures by nature. Fig. 5 demonstrates high path diversity and coverage of UCMP paths. In our simulated

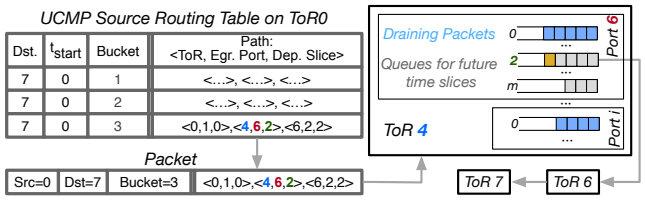


Figure 4: **Source routing for UCMP paths.**

network, UCMP provides multi-paths 94.4% of the time, and 93.2% of the UCMP paths are edge-disjoint. The multi-paths serve as backup paths in the event of failures, and ToRs can update the routing entries to redirect traffic to the backup paths, as detailed in the UCMP system (§6.2).

In the time slice when a pair of ToRs are connected by a direct circuit, the direct path has the minimum latency and minimum hop count, making it the sole path in the UCMP group as per the UCMP properties described in §4.3. For such instances, we prepare backup 2-hop paths for the single UCMP path. These time slices are rare, typically occurring once per circuit cycle. In our simulated network (§7), they only appear 5.6% of the time, and we provide backup paths for only 3.9% of the total UCMP paths.

As Fig. 12 shows, with this simple recovery strategy, UCMP can sustain 10% ToR failures, 5% link failures, and 16.6% circuit switch failures without loss of connectivity, and the performance degradation under normal levels of DCN failures is minimal. If higher availability is required, UCMP supports higher ratios of backup paths or recomputation for the affected paths, with compromised uniform costs.

6 UCMP Implementation

We implemented a UCMP prototype system with Intel Tofino2 switches and Mellanox NICs. In this section, we describe the key components including flow bucketing on the hosts (§6.1), source routing on source ToRs (§6.2), and rerouting on intermediate ToRs when packets violate the time schedule (§6.3).

6.1 Flow Aging and Bucketing

The boundary values of flow size buckets derived in §5.1 and §5.2 are specific to each UCMP group. To have globally recognizable buckets, we get the union of all the boundary values by Eqn. 4 across UCMP groups. These boundary values serve as the stepping thresholds for flow aging, and the intervals between the boundaries become the buckets. These universal bucket intervals are fine-grained, so for a particular UCMP group, several buckets may map to the same path.

We implemented flow aging in `libvma` [2], a high-performance user-space library supported by Mellanox NICs. Alternatively, it can be implemented as a Linux kernel module like done in PIAS [11]. It counts the number of bytes each flow has sent and tags outgoing packets of the flow with the corresponding bucket index. We follow PIAS to use the Differentiated Services Code Point (DSCP) field for bucket tagging, which has 6 bits and can support 64 buckets. This is sufficient for UCMP. As we will show in Table 2, UCMP requires 42 buckets for a 1024-ToR RDCN.

6.2 Source Routing

UCMP requires the entire path to be known before routing is performed, so we realized UCMP with source routing on source ToRs. We pre-load the routing paths onto each source ToR. Fig. 4 illustrates the UCMP lookup table entries, where the destination ToR and the starting time slice map to a UCMP group, and the bucket index identifies the specific path in the group. The action field contains the list of next-hop ToRs for the matched path, which are written into the packet using the Strict Source and Record Route (SSRR) option for source routing [33]. We overwrite the next-hop IP addresses in SSRR with the tuple of the *next-hop ToR*, the *egress port* on the ToR, and the *time slice* when the packet should be sent out from the port.

In the case of multiple tied paths in a bucket, we adopt ECMP's strategy of randomly distributing flows across different paths. This can be realized using the action selector feature on Tofino2 switches, which allows one match entry to have a group of action data (i.e., tied paths) and select one action based on a given index (i.e., the 5-tuple hash value of a flow).

We assume that the ToRs are synchronized with the RDCN controller and are aware of the circuit schedule, as suggested by previous RDCN work [29, 30, 32]. Packet waiting at intermediate ToRs can be achieved using queue pausing/unpausing in Tofino2. Each egress port of Tofino2 switches has 128 queues, and each queue is unpaused at a pre-set time slice to be active for draining packets. Once a time slice duration elapses, the currently active queue is paused, and the next priority queue is resumed. This process is periodic and aligns with the RDCN's circuit schedule. Note that the queues can be circular without one-to-one mapping to time slices. Table 2 shows a 1024-ToR RDCN needs 32 queues per port at most, under Tofino2's limit of 128 queues per port.

As Fig. 4 shows, a packet from ToR_0 to ToR_7 arrives at ToR_0 in $t_{start} = 0$. It belongs to bucket 3, so it matches to the third entry in ToR_0's UCMP source routing table. The corresponding UCMP path is written to the packet header's SSRR field and will be enforced by the following ToRs. First, the source ToR_0 forwards the packet to the next hop ToR_4 at the current time slice 0. When the packet arrives at ToR_4, ToR_4 reads out egress $port_6$ and send time slice $t = 2$ from the path list. It then enqueues the packet to $port_6$'s priority queue 2, waiting for the send time slice $t = 2$, which is 2 time slices away from the current time slice 0. Once the queue is unpaused at $t = 2$, the packet will be sent out to the next hop ToR_6 and eventually arrive at the destination ToR_7.

Per-flow path allocation in UCMP does not require extra states on ToRs other than the buckets, which makes UCMP lookup tables scalable to large DCNs irrespective of the number of flows. As shown in Table 2, the SRAM usage of lookup table entries per ToR for a 1024-ToR RDCN setting is only 3.06% of the Tofino2 capacity, demonstrating the sustainability of our UCMP design.

6.3 Rerouting

Each priority queue lasts for one time slice duration, indicating it can only carry a fixed number of bytes, equivalent to the product of the time slice duration and the link bandwidth. Consequently, it is possible that a packet cannot be buffered at the target priority queue and miss the planned time slice. In such scenarios, we adopt

a simple rerouting strategy: recirculating the packet and using the intermediate ToR as the new source ToR. Source routing in §6.2 is repeated at the intermediate ToR to select a new UCMP path and forward the packet onward. Packets that have been recirculated more than 5 times on a ToR are dropped, which is extremely rare in our simulation (§7). We observe at most 3.03% rerouted packets in all our simulation experiments, even under heavy traffic loads saturating the network core. These results indicate the low overhead of the rerouting mechanism.

7 Evaluation

In this section, we evaluate the performance of UCMP with analytics of the UCMP paths and simulations over DCN traffic traces. We introduce our experimental setup (§7.1), followed by experiments to evaluate different aspects of the UCMP design (§7.2 to §7.4).

7.1 Experimental Setup

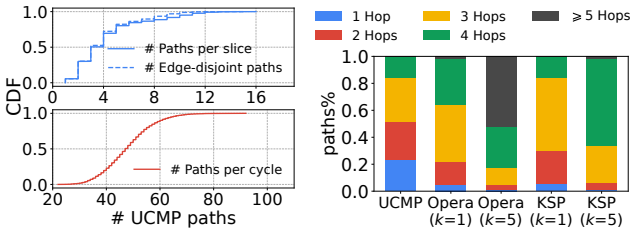
Simulated network. We realize UCMP in the *htsim* packet-level simulator, which is widely used for evaluating routing and transport designs in DCNs and RDCNs [25, 29, 37, 41, 43]. The simulated RDCN comprises 108 ToRs, each connected to 6 hosts via 6 downlinks and to 6 circuit switches via 6 uplinks. The network contains 648 hosts and 6 circuit switches in total. All links are 100 Gbps. The propagation delay between every ToR pair is set to 500 ns (100 meters of fiber length), and the distance between hosts and ToRs is ignored.

Baselines. We compare UCMP with VLB, KSP, and Opera as explained in §2.2. For KSP and Opera, we set k to 1 and 5 to evaluate the single-path and multi-path variants.

Transport protocols. We run UCMP with DCTCP and NDP to evaluate its performance under different DCN transport protocols. We pair each routing scheme with its default transport protocol: RotorLB for VLB [30], DCTCP for KSP [43], and NDP for short flows in Opera under 15 MB. In Opera, long flows over 15 MB are routed with VLB thus RotorLB is used as well. For DCTCP, we configure the switch queue size to 300 MTU-sized packets and the ECN threshold to 65 MTU-sized packets. For NDP, the switch queue size is set to 80 MTU-sized packets to satisfy the Opera constraint.

Circuit settings. To align with different circuit switching technologies [12, 29, 30], we set the circuit reconfiguration delay to 10 ns, 1 µs, and 10 µs, and set the time slice duration to 1 µs, 10 µs, 50 µs, and 300 µs. For Opera, we adopt its native circuit schedule that offsets the reconfiguration times across circuit switches to achieve partial topology reconfiguration per time slice. We modify this schedule by removing the offsets to obtain a fully reconfigurable schedule for VLB, KSP, and UCMP. We consider an alternative round-robin circuit schedule for UCMP to test it under different schedules.

Workloads. We run the *web search* and *data mining* traces from Microsoft's production DCNs [9, 23]. We scale the traces to reach 40% utilization on the host-to-ToR links. This is an extremely high load saturating the core bandwidth. To put things in perspective, our simulated network uses fewer switches than a 3:1 oversubscribed Clos topology. The web search trace features short flows, with the majority under 15 MB, whereas the data mining trace involves long flows whose sizes can reach up to 1 GB, and the majority of packets are from flows exceeding 15 MB.



(a) (b)

Figure 5: **(a) UCMP path numbers. (b) Hop count comparison.**

7.2 Path Characteristics

We first analyze the UCMP paths to show their properties, which will help understand the later simulation results.

Number of paths. As explained in §4.3, a UCMP group $P(src, dst, t_{start})$ is specific to a source ToR src, a destination ToR dst, and a starting time slice t_{start} for routing. The top figure of Fig. 5a presents the UCMP group size distribution. Over 20% UCMP groups contain more than 5 paths, and the majority have more than 3 paths. These UCMP paths, though modest in numbers, are deliberately chosen for minimum uniform cost. UCMP achieves path diversity across time slices, as shown in the bottom figure for the number of unique paths per cycle. For example, the case for a single path in the UCMP group is when there happens to be a direct circuit between the ToR pair, which has both minimum latency and hop count by the uniform cost definition (Eqn. 2). In the next time slice, this ToR pair will have a new UCMP group having more paths but still ensuring minimum uniform cost. In this way, UCMP provides a wide range of high-quality path options throughout time.

Edge-disjoint paths. In the top figure of Fig. 5a, 93.2% of the UCMP paths are edge-disjoint. Similarly, the percentage is 89.8% for the alternative setting with a round-robin circuit schedule in Fig. 16 of Appx. C. Recall from the UCMP path space in Fig. 2 that UCMP paths have decreasing latency as the hop count grows, and the path latency defined in Eqn. 1 is determined solely by the last hop, so UCMP paths have different last-hop ToRs, in different time slices. With the average path length as low as 2.32 hops in Fig. 5b, which will be explained below, UCMP paths have a high probability of being edge-disjoint. This property contributes to high path coverage. Consequently, as we detail in Appx. C, the loads across ToR ports in our simulated network are well balanced, with Jain's fairness index [17] of 0.9.

Path hop counts. Fig. 5b displays the hop count distribution of all the paths over all the ToR pairs across time slices. UCMP exhibits lower hop counts than Opera and KSP, with an average of 2.32 hops and all the paths under 4 hops. KSP and Opera have high hop counts because they stick to continuous paths, and Opera even more so due to the extra constraint preventing rerouting of in-flight packets during circuit reconfiguration. We find this constraint an overshoot, because UCMP only has up to 3.03% rerouted packets in all our simulations, under the high traffic load saturating the network core. Naively expanding from single-path ($k = 1$) to multi-paths ($k = 5$) increases the hop count significantly: Opera's average hop count rises from 3.11 to 4.45, while KSP's from 2.80 to 3.61. This

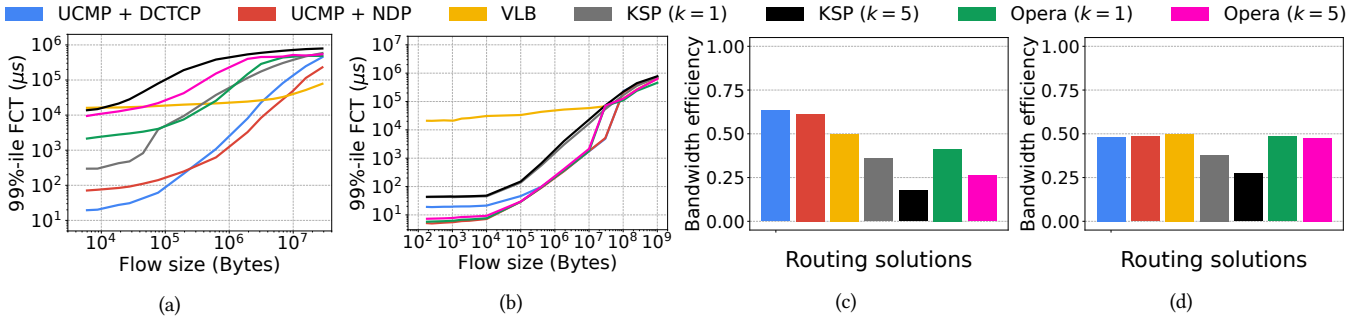


Figure 6: **FCTs under (a) web search and (b) data mining. Bandwidth efficiency under (c) web search and (d) data mining.**

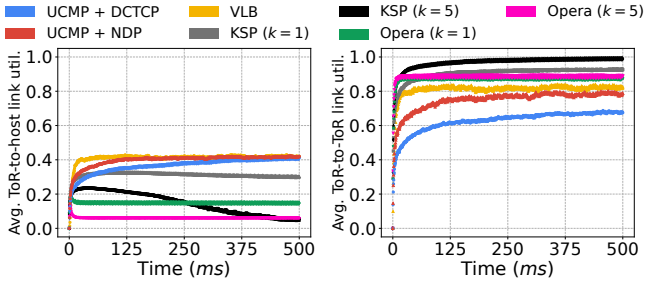


Figure 7: **Average (a) ToR-to-host (b) ToR-to-ToR link utilization under the web search trace.**

demonstrates the necessity of choosing multi-paths in a principled way like UCMP.

7.3 Performance Comparison

Next, we compare UCMP performance to the baselines in simulations, with the default setting of 50 µs time slice duration, 10 ns reconfiguration delay, and weight factor $\alpha = 0.5$.

FCT. In Fig. 6a and Fig. 6b, UCMP almost always outperforms the baselines, and we discuss the impact of transport protocols in §7.4. For the web search trace in Fig. 6a, UCMP achieves orders of magnitude lower FCTs for short flows. This is because minimizing uniform cost directs short flows to longer yet lower-latency paths, while long flows tend to wait at intermediate ToRs for shorter paths. In this regard, UCMP places a higher priority on sending short flows. VLB experiences high FCTs for short flows due to the excessive waiting time during its *phase 2* of transmission, while its high throughput benefits long flows. KSP and Opera exhibit elevated FCTs for the high hop counts shown in Fig. 5b. Notably, UCMP with NDP has 53% to 98% lower FCTs compared to the state-of-the-art single-path Opera routing ($k = 1$) also using NDP.

For the data mining trace in Fig. 6b, where long flows dominate, Opera resorts long flows over 15 MB to VLB, and we enable latency relaxation for UCMP. We find DCTCP and NDP are unable to achieve high throughput for a large number of latency-relaxed 2-hop paths. To overcome this limitation, we adopt the RotorLB transport protocol from Opera, whose implementation in the htsim simulator requires the full set of VLB paths as latency-relaxed paths. We continue to use DCTCP and NDP for the remaining short flows over regular UCMP paths. UCMP offloads long flows less aggressively than Opera but exhibits lower FCTs due to more efficient UCMP paths minimizing uniform costs. We expect even better performance of UCMP should transport protocols be improved for discontinuous latency-relaxed 2-hop paths, which, by design according §4.3, include 2-hop paths having lower minimum costs than VLB, e.g., allowing waiting at the source for lower overall latency.

Bandwidth efficiency. In Fig. 6c and Fig. 6d, UCMP also demonstrates high bandwidth efficiency. A value of 0.5, as in VLB, implies that packets, on average, traverse 2 hops. In Fig. 6c, UCMP achieves the highest bandwidth efficiency, e.g., 1.26× that of VLB, 1.76× that of KSP ($k = 1$), and 1.55× that of Opera ($k = 1$). These results are attributed to UCMP guiding long flows towards 1- and 2-hop paths in the web search trace. For the data mining trace in Fig. 6d, UCMP achieves similar bandwidth efficiency to VLB due to comparable effects of latency relaxation, and so does Opera which resolves to VLB in this case. KSP, and Opera under web search, display worse bandwidth efficiency, especially when $k = 5$, due to the high hop counts as Fig. 5b reveals.

Link utilization. To deepen the understanding of bandwidth efficiency, we look at the link utilization of the web search trace in Fig. 7a and Fig. 7b. The results for the data mining trace are shown in Fig. 17 of Appx. D. UCMP achieves 40% ToR-to-host link utilization, as high as the traffic load, suggesting UCMP facilitates high throughput of the hosts. At the same time, UCMP keeps the ToR-to-ToR link utilization at the lowest level. It shows UCMP maintains minimal network congestion, consistent with the low hop counts in Fig. 5b and high bandwidth efficiency in Fig. 6c. VLB doubles the load in the network, i.e., 40% ToR-to-host vs. 80% ToR-to-ToR, as a result of the 2-hop routing. Other solutions collapse with falling host throughput due to the long paths.

7.4 Performance under Various Settings

We move on to examine different design aspects of UCMP under diverse settings. Without loss of generality, these experiments emphasize on the web search trace.

Impact of transport protocols. UCMP is not reliant on a strict queue size upper bound for packet delivery as Opera. It shows obvious performance advantages over the baselines in Fig. 6 with both DCTCP and NDP.

In the web search trace, most traffic is made of shorter flows that route through standard UCMP paths, so the network core gets congested. In this case, DCTCP is better able to rate limit longer flows and prevent queues from excessively building up. This results in lower FCT for short flows compared to NDP, which struggles more to control traffic, and has to resort to active recovery as queues fill up.

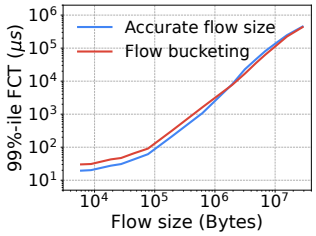


Figure 8: **FCTs with accurate flow size and flow bucketing.**

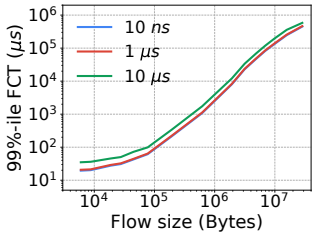
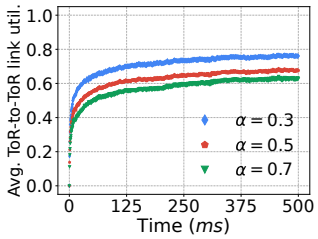
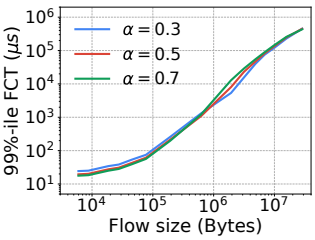


Figure 9: **FCTs under different reconfiguration delays.**

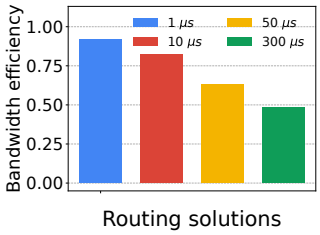


(a)

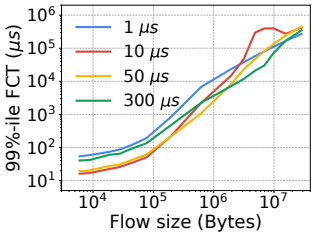


(b)

Figure 10: **(a) Avg. ToR-to-ToR link util. and (b) FCTs under α.**



(a)



(b)

Figure 11: **(a) Bandwidth efficiency and (b) FCTs under different time slice durations.**

In the data mining trace, most traffic consists of longer flows routed through latency-relaxed paths, resulting in a more uniformly distributed traffic load in the core. In this scenario, NDP performs better than DCTCP for the remaining short flows over the regular UCMP paths, thanks to NDP's lower latency overhead and more aggressive initial rate. Note that RotorLB is used for long flows over latency-relaxed paths. Improving DCTCP and NDP to deliver high throughput for a large number of 2-hop paths remains a topic for future study.

Degree of packet rerouting. We observe 3.03% of packets are rerouted in the web search trace and 0.9% in the data mining trace. Rerouting is initiated when a predetermined path is no longer available; the network then treats the current intermediate ToR as the starting point for selecting an alternative path. The new path chosen still follows the minimum uniform cost principle. This ensures that short flows are immediately directed along low-latency paths, whereas long flows are routed along short paths that optimize for bandwidth efficiency.

Accuracy of flow bucketing. As Fig. 8 shows, the tail FCTs achieved with our flow bucketing mechanism (§5.1) are very close to those obtained with accurate flow size information. More precisely, flow bucketing introduces slight FCT increases for shorter flows and decreases for longer flows. This is because flow bucketing, built atop flow aging, initially treats new flows as short flows, directing a small portion of packets from long flows to low-latency paths at the start. It modestly speeds up longer flows while marginally increasing congestion on ToR-to-ToR links, thus slightly delaying shorter flows. Overall, this finding confirms that UCMP is effective without accurate flow size information.

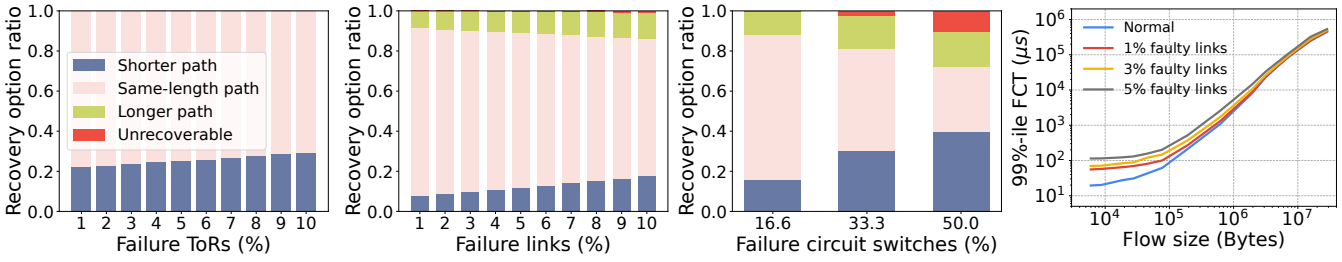
Impact of weight factor α. In §5.2, we have explained how to tune α in Eqn. 2 according to a targeted link utilization. As discussed, α balances latency and hop count. Increasing α directs flows to shorter yet higher-latency paths, resulting in improved bandwidth efficiency, i.e., a lower ToR-to-ToR link utilization. Fig. 10a validates our method, showing decreased link utilization with the increase of α. It also shows tuning interval can effectively regulate the link utilization, without causing high sensitivity. We set α to 0.5 for 70% ToR-to-ToR link utilization. As Fig. 10b reveals, FCT is insensitive to variations in α as long as it stays in a reasonable range for the targeted link utilization.

Impact of time slice duration. Fig. 11 demonstrates UCMP's adaptability to diverse time slice durations, contrary to the rigid constraint in Opera. In Eqn. 1, we define routing latency as the product of the number of time slices a path spans and the time slice duration. Hence, reducing the time slice duration leads to lower latency, which would cause uniform cost in Eqn. 2 to choose shorter paths to lower the penalty of the hop count term. Fig. 11a reflects this trend, where shorter time slices enjoy higher bandwidth efficiency, thanks to the lower hop counts. FCTs in Fig. 11b are less straightforward. For shorter time slices, on one hand, the improved bandwidth efficiency in Fig. 11a suggests less load in the network; on the other hand, packets are more likely to be rerouted due to reduced slice durations. The joint effects result in the FCT variations in Fig. 11b, but they follow similar trends of growth and consistently outperform the baselines.

Impact of reconfiguration delay. The reconfiguration delay primarily influences the RDCN duty cycle, defined as the ratio of the circuit holding time to the total cycle duration. As Fig. 9 depicts, a longer reconfiguration delay results in packets remaining in queues for an extended time before transitioning to the next time slice, leading to increased FCTs. With our based time slice duration of 50 µs, changing the reconfiguration delay from 1 µs to 10 µs reduces the duty cycle from 98% to 83%. Most RDCNs suggest a duty cycle larger than 90% [12, 30].

Robustness to failures. Fig. 12a-c illustrate the breakdown of UCMP recovery options available to the affected paths under ToR, link, and circuit switch failures. Only backing up 3.9% of the paths as described in §5.3, UCMP can sustain 10% ToR failures, 5% link failures, and 16.6% circuit switch failures without loss of connectivity, and the large majority of failed paths can transition to other UCMP paths with the same length, which preserves the minimum uniform cost. From §5.3, the high path diversity and the high percentage of edge-disjoint paths in Fig. 5a ensures failure resilience.

In Fig. 12d, UCMP exhibits low FCT degradation under up to 5% link failures over the web search trace. Putting it into context with Fig. 6a, even the elevated FCTs under failures are orders of magnitude lower than those of the baselines. This observation indicates no need for additional failure recovery involving backup paths or path recomputation at compromised uniform costs, though UCMP supports that.



(a) (b) (c) (d)

Figure 12: **UCMP's recovery options for (a) ToR, (b) link, (c) circuit switch failures, and (d) FCTs under 1%, 3%, and 5% faulty links.**

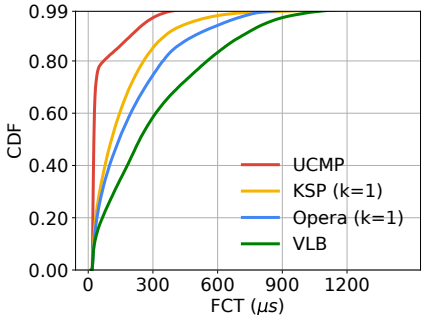


Figure 13: **Memcached FCTs on testbed.**

8 Prototype Testbed

In this section, we implement UCMP on a small-scale testbed to demonstrate the feasibility of our UCMP system in §6. We also implement VLB, KSP, and Opera on the testbed for cross-validation of the simulation results, with real applications on an end-to-end system. Besides performance comparisons, we realize the UCMP paths for various RDCN scales up to 1024 ToRs to evaluate the system overheads.

Testbed setup. Our RDCN testbed consists of 3 Intel Tofino2 programmable switches and 4 servers each equipped with a Mellanox ConnectX-5 dual-port NIC. We virtualize 2 physical switches each into 4 logical ToRs, and make each NIC port act as a logical host. We emulate a circuit switch with the third switch. Our emulated network thus contains 8 logical ToRs each connected to a logical host with a 100 Gbps downlink and to the circuit switch via 4 10 Gbps uplinks to mirror oversubscription in real DCNs. We set the time slice duration to 50 µs and the reconfiguration delay to 1 µs, as one of the settings in simulation. We set $k = 1$ for KSP and Opera, and set $\alpha = 0.5$ for UCMP. Everything is run with TCP as the transport protocol.

Application performance. We use the *Memcached* [3] key-value storage to generate short flows. We run 1 Memcached server and 7 Memslap [4] benchmarking clients each on a host, and a client requests 4KB data to a server in each PULL operation. We run *iPerf* [1] to generate long flows, where each host sends long-lasting traffic in the background to the host under its neighboring ToR.

Fig. 13 shows the FCT distribution of the Memcached flows. Although the testbed and simulation results are not directly comparable, the relative performance of the routing methods follows the same trends as in Fig. 6a. UCMP still outperforms all baselines by large margins, reaffirming the effectiveness of UCMP paths. KSP has lower FCTs than Opera because rerouting of in-flight packets is unlikely on the small-scale testbed, and KSP benefits from shorter paths. This is similar to the case for short flows in Fig. 6a. The impact of transport protocols is minimal for the Memcached short flows with fixed throughput background traffic. We thus verify the correctness of our simulation experiments.

Table 2: **Hardware resource usage for various RDCN scales.**

(N, d)	#Q/port	#Buckets	#Entries/ToR	SRAM
(108, 6)	18	27	9.5K	0.38%
(324, 12)	27	34	50K	0.94%
(768, 24)	32	36	140K	2.31%
(1024, 32)	32	42	187K	3.06%

Number of queues. Table 2 presents UCMP's switch resource consumption, where N is the number of ToRs in the RDCN, and d is the number of uplinks per ToR. The first column shows the maximum number of priority queues needed per ToR egress port, which by our design in §6.2 is the number of time slices per cycle, essentially N/d. This number does not change much as N and d scale simultaneously in practice. A 1024-ToR RDCN needs 32 queues per port, well under the capacity limit of commercial switches [22, 35], e.g., 128 queues per port in Tofino2 switches.

Number of flow buckets. The second column shows the total number of buckets across topologies in the RDCN. A 1024-ToR RDCN requires 42 buckets. The number of buckets is mainly determined by h_{max}, the maximum hop count of UCMP paths, and N/d, the number of time slices per cycle. N/d stays mostly constant as discussed above, and h_{max} is the diameter of the topologies, so the number of buckets grows slowly with the network scale. As described in §6.1, we use the 6-bit DSCP field to encode buckets, which provides 64 buckets in total, adequate for large-scale RDCNs.

Number of routing entries. The third column lists the number of routing entries per source ToR used for source routing in §6.2. This number is proportional to the number of destination ToRs $N - 1$ and the number of time slices per cycle N/d. UCMP requires 187K entries for 1024 ToRs, and the entries are stored in the switch SRAM. The low SRAM usage in the last column suggests our design is sustainable.

9 Discussion

Buffer management. Any RDCN routing scheme that involves waiting at intermediate hops are subject to buffering issues, e.g., VLB and UCMP. Nevertheless, with the performance gains shown in the paper, we should not step back to solutions like Opera and KSP that stick to continuous paths.

The maximum queue occupancy is influenced by various factors, including the transport protocol, traffic load, and time slice duration.

Generally, shorter time slices drastically reduce waiting times and consequently buffer usage. We have not observed severe buffer problems with UCMP in our experiments, partly because UCMP benefits from the short time slices of modern RDCNs, and partly because UCMP further reduces the waiting times by considering both latency and bandwidth efficiency when optimizing the uniform cost. However, buffering issues may still occur, especially for long flows. A potential hybrid solution could involve buffering short flows at ToRs and long flows at hosts, offering a viable buffering-bandwidth trade-off.

Packet reordering. Packet reordering is a common issue in RDCNs, as the topology changes and paths get updated accordingly. It is especially serious with randomized packet spraying across multiple time slices such as in VLB, but is also present in the single-path routes of Opera. UCMP also exhibits packet reordering when flows cross different time slices and later-dispatched packets end up on faster paths than the previously sent packets. Nevertheless, our solution aims to avoid heavy reordering by mapping flows to a single UCMP path. Fortunately, there are recent transport protocols specifically designed to handle reordering, e.g., NDP [25] and TDTCP [16]. We show good performance with NDP in the paper, and we leave deeper study of the topic to future work.

UCMP extension. UCMP serves as an equivalent to ECMP for RDCNs. Inspired by existing approaches that allocate ECMP paths based on congestion sensing [8, 21, 26], our method could also incorporate congestion states into the online path assignment. We can change the mapping function from flows to the buckets to impose penalties on congested paths, like live tuning of α in §5.2. In this way, we make congested paths less favorable and help mitigate network hot spots. We leave this optimization for future work.

Improvement of transport protocols. Transport protocols designed for static networks work well under the assumption of relatively stable end-to-end latency. However, latency of UCMP paths, especially shorter (e.g., 2-hop) ones, can be subject to large variations of waiting time at intermediate ToRs. As observed in our simulation experiments (§7), DCTCP and NDP perform reasonably well for longer paths but suffer from low throughput for latency-relaxed 2-hop paths. Since routing solutions for RDCNs are usually coupled with customized transport protocols for optimized performance [12, 30], we consider it necessary to develop dedicated transport protocols for UCMP in the future.

Switch-centric design. UCMP is a switch-centric design, with most functionalities implemented on the ToRs. The host system simply handles flow aging and determines the bucket for each flow. While an ECMP-like design transparent to host machines might seem appealing, moving the flow bucketing functionality to the ToRs would require per-flow states to track the accumulated traffic amount, potentially causing state explosion on switches.

10 Related Work

RDCNs. Traffic-aware RDCNs were first proposed to reconfigure the topology based on real-time traffic demands [15, 20, 24, 32, 34]. As circuit duration started to shrink, traffic-oblivious RDCNs gained popularity as a scalable alternative [12, 29, 30]. The focus of this work is on routing for traffic-oblivious RDCNs, and we have shown performance benefits over VLB [12, 30], KSP [41, 43], and Opera [29]. Mars [6] and VBS [10, 45] have been proposed recently to optimize the circuit schedules for traffic-oblivious RDCNs, and their routing strategies are similar to VLB. In that sense, we expect UCMP to improve their performance as UCMP is general to different traffic-oblivious circuit schedules. reTCP [31] and TDTCP [16] are TCP variants for RDCNs, but originally optimized for hybrid electrical-optical RDCNs. Their suitability for UCMP specific to pure RDCNs is unclear, and we leave this investigation to future work.

Multi-path routing and load balancing in DCNs. ECMP is the *status quo* routing scheme for Clos topologies such as Fat-tree [7, 40] and VL2 [23]. It leverages the large number of equal-cost paths to uniformly distribute traffic. WCMP [46] is an extension to ECMP for use in settings where the available paths are asymmetric. For instance, Jupiter [34], Google's traffic-aware RDCN architecture, uses WCMP for better load balancing instead of always mapping flows to the shortest paths like in ECMP. In this regard, UCMP is a further extension that takes advantage of the unique setting of RDCNs to provide paths based on latency-efficiency trade-offs. CONGA [8], DRILL [21], Vertigo [5], and Hula [26] assign flows to ECMP paths based on real-time congestion signals, which leads to better load balancing. Our UCMP implementation assigns paths statically based on the flow size, but we do not exclude future extensions to consider congestion states. On the other hand, MPTCP [37] is a TCP extension that splits a single flow in multiple subflows traveling across different paths in parallel, and is often coupled with KSP routing in expander graph topologies. In this work we showcase UCMP using single-path transport, but we believe an adoption of MPTCP-like transport could benefit performance.

11 Conclusion

Benefiting from the progress of high-speed circuit switches, traffic-oblivious RDCNs emerge as a potential solution for the next-generation cloud infrastructure. However, routing on such network topologies remains relatively unexplored. This paper presents UCMP as the first principled method for multi-path routing in RDCNs. We have demonstrated how our definition of uniform cost effectively balances latency and hop count, and how our design of offline path calculation and online path assignment successfully delivers the expected performance in FCT and bandwidth efficiency. Being first of its kind, we hope UCMP to inspire alternative principled designs for multi-path routing in RDCNs. Also as an ECMP equivalent, we anticipate followup work on runtime adjustments of the load distribution over the UCMP paths, as this line of work developed on top of ECMP [5, 8, 21, 46].

Acknowledgments

We thank our shepherd, Amin Vahdat, and the anonymous reviewers for their insightful comments. We also extend our gratitude to our collaborators, Balakrishnan Chandrasekaran and Raj Joshi, for their involvement in the design of an early version of this work.

References

[1] 2024. iPerf. https://iperf.fr/.
[2] 2024. Mellanox Messaging Accelerator. https://github.com/Mellanox/libvma/blob/master/README.
[3] 2024. Memchached. https://memcached.org/.
[4] 2024. Memslap. http://docs.libmemcached.org/bin/memslap.html.
[5] Sepehr Abdous, Erfan Sharafzadeh, and Soudeh Ghorbani. 2021. Burst-tolerant datacenter networks with vertigo. In *Proceedings of the 17th International Conference on emerging Networking Experiments and Technologies*. 1–15.
[6] Vamsi Addanki, Chen Avin, and Stefan Schmid. 2023. Mars: Near-optimal throughput with shallow buffers in reconfigurable datacenter networks. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 7, 1 (2023), 1–43.
[7] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. 2008. A scalable, commodity data center network architecture. *ACM SIGCOMM computer communication review* 38, 4 (2008), 63–74.
[8] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, et al. 2014. CONGA: Distributed congestion-aware load balancing for datacenters. In *Proceedings of the 2014 ACM conference on SIGCOMM*. 503–514.
[9] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 Conference*. 63–74.
[10] Daniel Amir, Tegan Wilson, Vishal Shrivastav, Hakim Weatherspoon, Robert Kleinberg, and Rachit Agarwal. 2022. Optimal oblivious reconfigurable networks. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*. 1339–1352.
[11] Wei Bai, Li Chen, Kai Chen, Dongsu Han, Chen Tian, and Hao Wang. 2017. PIAS: Practical information-agnostic flow scheduling for commodity data centers. *IEEE/ACM Transactions on Networking* 25, 4 (2017), 1954–1967.
[12] Hitesh Ballani, Paolo Costa, Raphael Behrendt, Daniel Cletheroe, Istvan Haller, Krzysztof Jozwik, Fotini Karinou, Sophie Lange, Kai Shi, Benn Thomsen, et al. 2020. 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*. 782–797.
[13] Kai Chen, Ankit Singla, Atul Singh, Kishore Ramachandran, Lei Xu, Yueping Zhang, Xitao Wen, and Yan Chen. 2013. OSA: An optical switching architecture for data center networks with unprecedented flexibility. *IEEE/ACM Transactions on Networking* 22, 2 (2013), 498–511.
[14] Kai Chen, Xitao Wen, Xingyu Ma, Yan Chen, Yong Xia, Chengchen Hu, and Qunfeng Dong. 2015. WaveCube: A scalable, fault-tolerant, high-performance optical data center architecture. In *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 1903–1911.
[15] Li Chen, Kai Chen, Zhonghua Zhu, Minlan Yu, George Porter, Chunming Qiao, and Shan Zhong. 2017. Enabling Wide-Spread Communications on Optical Fabric with MegaSwitch. In *NSDI*, Vol. 17. 577–593.
[16] Shawn Shuoshuo Chen, Weiyang Wang, Christopher Canel, Srinivasan Seshan, Alex C Snoeren, and Peter Steenkiste. 2022. Time-division TCP for reconfigurable data center networks. In *Proceedings of the ACM SIGCOMM 2022 Conference*. 19–35.
[17] Dah-Ming Chiu and Raj Jain. 1989. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN systems* 17, 1 (1989), 1–14.
[18] Paolo Costa, Hitesh Ballani, Kaveh Razavi, and Ian Kash. 2015. R2C2: A network stack for rack-scale computers. *ACM SIGCOMM Computer Communication Review* 45, 4 (2015), 551–564.
[19] Nathan Farrington, George Porter, Sivasankar Radhakrishnan, Hamid Hajabdolali Bazzaz, Vikram Subramanya, Yeshaiahu Fainman, George Papen, and Amin Vahdat. 2010. Helios: a hybrid electrical/optical switch architecture for modular data centers. In *Proceedings of the ACM SIGCOMM 2010 Conference*. 339–350.
[20] Monia Ghobadi, Ratul Mahajan, Amar Phanishayee, Nikhil Devanur, Janardhan Kulkarni, Gireeja Ranade, Pierre-Alexandre Blanche, Houman Rastegarfar, Madeleine Glick, and Daniel Kilper. 2016. Projector: Agile reconfigurable data center interconnect. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 216–229.
[21] Soudeh Ghorbani, Zibin Yang, P Brighten Godfrey, Yashar Ganjali, and Amin Firoozshahian. 2017. Drill: Micro load balancing for low-latency data center networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 225–238.
[22] Prateesh Goyal, Preey Shah, Kevin Zhao, Georgios Nikolaidis, Mohammad Alizadeh, and Thomas E. Anderson. 2022. Backpressure flow control. In *Proceedings of NSDI*.
[23] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. 2009. VL2: A scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*. 51–62.
[24] Navid Hamedazimi, Zafar Qazi, Himanshu Gupta, Vyas Sekar, Samir R Das, Jon P Longtin, Himanshu Shah, and Ashish Tanwer. 2014. Firefly: A reconfigurable wireless data center fabric using free-space optics. In *Proceedings of the 2014 ACM conference on SIGCOMM*. 319–330.
[25] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W Moore, Gianni Antichi, and Marcin Wójcik. 2017. 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*. 29–42.
[26] Naga Katta, Mukesh Hira, Changhoon Kim, Anirudh Sivaraman, and Jennifer Rexford. 2016. Hula: Scalable load balancing using programmable data planes. In *Proceedings of the Symposium on SDN Research*. 1–12.
[27] Sergey Legtchenko, Nicholas Chen, Daniel Cletheroe, Antony Rowstron, Hugh Williams, and Xiaohan Zhao. 2016. {XFabric}: A Reconfigurable {In-Rack} Network for {Rack-Scale} Computers. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. 15–29.
[28] Yunpeng James Liu, Peter Xiang Gao, Bernard Wong, and Srinivasan Keshav. 2014. Quartz: a new design element for low-latency DCNs. *ACM SIGCOMM Computer Communication Review* 44, 4 (2014), 283–294.
[29] William M Mellette, Rajdeep Das, Yibo Guo, Rob McGuinness, Alex C Snoeren, and George Porter. 2020. Expanding across time to deliver bandwidth efficiency and low latency. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 1–18.
[30] William M Mellette, Rob McGuinness, Arjun Roy, Alex Forencich, George Papen, Alex C Snoeren, and George Porter. 2017. Rotornet: A scalable, low-complexity, optical datacenter network. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 267–280.
[31] Matthew K Mukerjee, Christopher Canel, Weiyang Wang, Daehyeok Kim, Srinivasan Seshan, and Alex C Snoeren. 2020. Adapting {TCP} for reconfigurable datacenter networks. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 651–666.
[32] George Porter, Richard Strong, Nathan Farrington, Alex Forencich, Pang Chen-Sun, Tajana Rosing, Yeshaiahu Fainman, George Papen, and Amin Vahdat. 2013. Integrating microsecond circuit switching into the data center. *ACM SIGCOMM Computer Communication Review* 43, 4 (2013), 447–458.
[33] Jon Postel. 1981. RFC0791: Internet protocol.
[34] Leon Poutievski, Omid Mashayekhi, Joon Ong, Arjun Singh, Mukarram Tariq, Rui Wang, Jianan Zhang, Virginia Beauregard, Patrick Conner, Steve Gribble, et al. 2022. Jupiter evolving: transforming google's datacenter network via optical circuit switches and software-defined networking. In *Proceedings of the ACM SIGCOMM 2022 Conference*. 66–85.
[35] Ting Qu, Raj Joshi, Mun Choon Chan, Ben Leong, Deke Guo, and Zhong Liu. 2019. SQR: In-network packet loss recovery from link failures for highly reliable datacenter networks. In *Proceedings of ICNP*.
[36] Martin Raab and Angelika Steger. 1998. "Balls into bins"—A simple and tight analysis. In *International Workshop on Randomization and Approximation Techniques in Computer Science*. Springer, 159–170.
[37] Costin Raiciu, Sebastien Barre, Christopher Pluntke, Adam Greenhalgh, Damon Wischik, and Mark Handley. 2011. Improving datacenter performance and robustness with multipath TCP. *ACM SIGCOMM Computer Communication Review* 41, 4 (2011), 266–277.
[38] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C Snoeren. 2015. Inside the social network's (datacenter) network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. 123–137.
[39] Vishal Shrivastav, Asaf Valadarsky, Hitesh Ballani, Paolo Costa, Ki Suh Lee, Han Wang, Rachit Agarwal, and Hakim Weatherspoon. 2019. Shoal: A network architecture for disaggregated racks. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 255–270.
[40] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, et al. 2015. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. *ACM SIGCOMM computer communication review* 45, 4 (2015), 183–197.
[41] Ankit Singla, Chi-Yao Hong, Lucian Popa, and P Brighten Godfrey. 2012. Jellyfish: Networking data centers randomly. In *Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*. 225–238.
[42] Vojislav Đukić, Sangeetha Abdu Jyothi, Bojan Karlaš, Muhsen Owaida, Ce Zhang, and Ankit Singla. 2019. Is advance knowledge of flow sizes a plausible assumption?. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 565–580.
[43] Asaf Valadarsky, Gal Shahaf, Michael Dinitz, and Michael Schapira. 2016. Xpander: Towards optimal-performance datacenters. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*. 205–219.
[44] Guohui Wang, David G Andersen, Michael Kaminsky, Konstantina Papagiannaki, TS Eugene Ng, Michael Kozuch, and Michael Ryan. 2010. c-Through: Part-time optics in data centers. In *Proceedings of the ACM SIGCOMM 2010 Conference*. 327–338.
[45] Tegan Wilson, Daniel Amir, Vishal Shrivastav, Hakim Weatherspoon, and Robert Kleinberg. 2023. Extending Optimal Oblivious Reconfigurable Networks to all

N. In *2023 Symposium on Algorithmic Principles of Computer Systems (APOCS)*. SIAM, 1–16.

[46] Junlan Zhou, Malveeka Tewari, Min Zhu, Abdul Kabbani, Leon Poutievski, Arjun Singh, and Amin Vahdat. 2014. WCMP: Weighted cost multipathing for improved fairness in data centers. In *Proceedings of the Ninth European Conference on Computer Systems*. 1–14.

Appendix

Appendices are supporting material that has not been peer-reviewed.

A Path Computing Algorithm

Algorithm 1 n-Hop Minimum-Latency Path Algorithm

Require:
$\mathcal{R} \leftarrow$ set of ToRs
$\mathcal{S} \leftarrow$ set of time slices in one cycle
$h_{max} \leftarrow$ hop count of the globally minimum-latency path
Ensure:
$\mathcal{P} \leftarrow$ set of n-hop minimum-latency paths over all src-dst ToR pairs in $\mathcal{R} \times \mathcal{R}$ and all start time slices t_{start} in $\mathcal{S}$, $n \in [1, h_{max}]$

for t_{start} in $\mathcal{S}$ **do**
  for n in $[1, h_{max}]$ **do**
    if n == 1 **then** ▷ Direct paths
      for $\langle src, dst \rangle$ in $\mathcal{R} \times \mathcal{R}$ **do**
        $p(src, dst, t_{start}, 1) = [src, dst]$
        $\mathcal{P}$.add(p)
    else ▷ Multi-hop paths
      for $\langle src, dst \rangle$ in $\mathcal{R} \times \mathcal{R}$ **do**
        $SP \leftarrow \emptyset$
        $sp_1 = p(src, dst, t_{start}, n-1)$
        for intermediate ToR $last$ in $\mathcal{R} - \{src, dst\}$ **do**
          $sp_2 = p(last, dst, t_{start}, 1)$
          $SP.add(sp_2)$
        sort SP by $latency(sp_2)$ in ascending order
        for sp_2 in SP **do**
          if $latency(sp_1) \leq latency(sp_2)$ **then**
            $p = sp_1 + sp_2$
            $\mathcal{P}$.add(p)
            break
return $\mathcal{P}$

The time complexity of calculating $\Omega(h_{max})$ is $O(N^3)$, where N represents the number of ToRs. As detailed in Alg. 1, incorporating $\Omega(h_{max})$ results in a time complexity of $O(N^3 \times \Omega(h_{max}) \times N/d)$ and a space complexity of $O(N^2 \times \Omega(h_{max}) \times N/d)$. Here, $\Omega(h_{max})$ and N/d (the number of time slices per cycle) remain relatively small and constant even as the network size increases. Consequently, the overall time and space complexity simplify to $O(N^3)$ and $O(N^2)$, respectively.

B Details on Upper Bound of h_{max}

We provide more details on determining the upper bound of h_{max}, which is the hop count of the globally fastest path. Recall that h_{static} indicates the largest network diameter of expander graphs that constitute RDCNs, and h_{slice} denotes the maximum number of hops a packet can traverse within a single time slice. For example, if the link bandwidth is 100 Gbps, the network transmission delay for an MTU-sized packet is 120 ns. Additionally, assuming a 100-meter distance between a ToR pair, the propagation delay is 500 ns. In the case of a 1 µs time slice, the packet can traverse at most $h_{slice} = \lfloor 1000/(500+120) \rfloor = 1$ hop. With a time slice of 10 µs, $h_{slice} = \lfloor 10000/(500+120) \rfloor = 16$ hops. The computation of h_{max} is conducted under two cases.

Case I: $h_{slice} \geq h_{static}$. Then $h_{max} \leq h_{static}$.

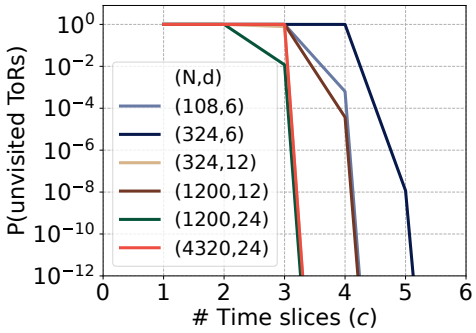


Figure 14: $P(unvisited\ ToRs)$ **across diverse topology scales.**

This suggests that any packet, irrespective of the src/dst ToR, can reach the dst ToR within a single time slice. This further implies that any n-hop fastest path will complete within a time slice. So we have $h_{max} \leq h_{static}$.

Case II: $h_{slice} < h_{static}$. Then $h_{max} \leq h_{slice} \times S$.

Here S is the maximum number of time slices the globally fastest path can cover. In cases where the time slice is short, a packet might have to span several time slices before ultimately arriving at the dst ToR. In such cases, a packet might never reach the dst ToR if a path is selected poorly. At the end of each time slice, a packet could halt at an intermediate ToR, which always requires more than h_{slice} hops to reach the dst ToR. However, when next hop is chosen randomly, we could estimate the probability of not reaching the dst ToR after c time slices. Subsequently, S is set to c when the probability is low enough. Regarding h_{slice} hops at each time slice, the total hop count is $h_{slice} \times S$. This implies that a packet can reach any ToR after $h_{slice} \times S$ hops even the next hop is chosen randomly.

Estimating the probability of not reaching every dst ToR can be modeled as the Balls into Bins problem [36]: throw m balls into n bins randomly, determining the likelihood of having empty bins. Relating this concept to our scenario, consider a RDCN consisting of N ToRs, each equipped with d uplinks. Aligned with the RDCN's attributes, at each new time slice a ToR can connect to d new ToRs. After c time slices, this accumulates to $M = d^c$ combinations. If the packet chooses a ToR randomly at every time slice, after c time slices, what is the probability of the existence of unexplored ToRs? We follow the solution of Balls into Bins problem and calculate the probability, denoted as $P(unvisited\ ToRs)$.

We start by calculating the probability denoted as $P(one\ \ ToR\ is\ \ unvisited)$, which indicates the likelihood of a ToR remaining unexplored.

$$P(one\ ToR\ is\ unvisited) = (1 - 1/N)^M \tag{5}$$

$$\begin{aligned} P(unvisited\ ToRs) &= 1 - P(all\ ToRs\ are\ visited) \\ &= 1 - [P(one\ ToR\ is\ visited)]^N \\ &= 1 - [1 - P(one\ ToR\ is\ unvisited)]^N \\ &= 1 - [1 - (1 - 1/N)^M]^N \end{aligned} \tag{6}$$

We compute $P(unvisited\ ToRs)$ across diverse topology scales (Fig. 14). As the number of time slices (c) increases, the probability drops significantly. We adopt a probability threshold of 10^{-12} and use the current c value as the maximum time slices (S) a packet

Table 3: **Upper bounds of h_{max} for various topology scales.**

Time slice	(N,d)	h_{slice}	h_{static}	Case	S	$\Omega(h_{max})$
1 µs	(108,6)	1	5	II	5	**5**
1 µs	(324,6)	1	8	II	6	**6**
2 µs	(108,6)	3	5	II	5	**15**
2 µs	(4320,24)	3	4	II	4	**12**
5 µs	(1200,12)	8	5	I	5	**5**
10 µs	(4320,24)	16	4	I	4	**4**

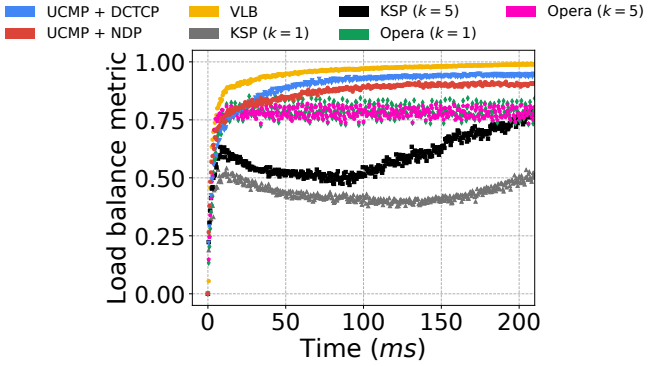


Figure 15: **Load balance metric with web search trace.**

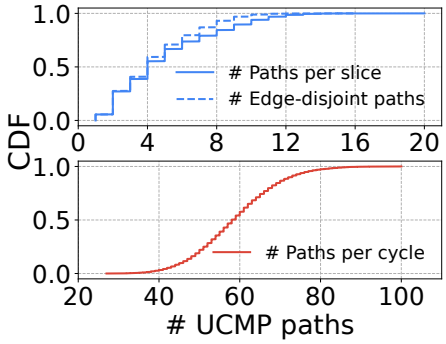


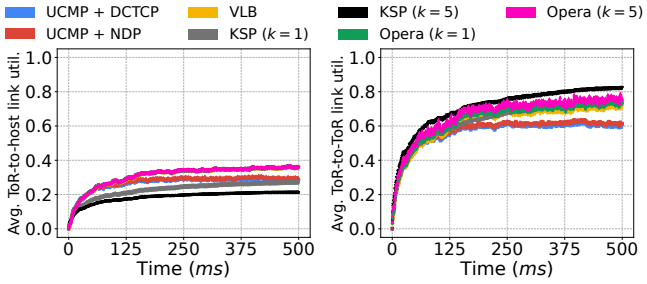
Figure 16: **Number of paths in UCMP groups under a randomly generated schedule.**

needs to traverse. For instance, in a RDCN with 108 ToRs and 6 uplinks, S is set to 5 to ensure that $P(unvisited\ ToRs)$ is below the probability threshold. Similarly, in a scenario with 324 ToRs and 6 uplinks, S is set to 6.

We further provide upper bounds of h_{max} for various topology scales and time slice durations in Table 3. The upper bound is denoted as $\Omega(h_{max})$. It's apparent that $\Omega(h_{max})$ in Case II surpass those in Case I. This is due to the fact that $\Omega(h_{max})$ in Case II are relatively loose upper bounds. Yet, considering that the path computations are conducted offline, these values remain within an acceptable range.

C UCMP Path Analysis

High path coverage. UCMP achieves high path coverage with a fairly load-balanced network. We use the Jain fairness [17] of the queue sizes across the ToR egress ports to assess the level of load balancing, as Eqn. 7 shows.



(a) (b)

Figure 17: **Average (a) ToR-to-host (b) ToR-to-ToR link utilization under the data mining trace.**

$$Load\ \ balance\ \ metric = \frac{(\sum_{i=1}^{m} q_i)^2}{m\sum_{i=1}^{m} q_i^2} \tag{7}$$

The range of the load balance metric is $[1/m, 1]$, where m is the number of total egress ports. A larger value means the network is more load-balancing, and a metric of 1 indicates all queue sizes at each port are precisely the same.

In our simulation, queue sizes are sampled every 20 µs, and we only display one sampling point every 1 ms. As Fig. 15 shows, VLB has the best path coverage, with the metric approaching its upper bound of 1, meaning all ports are almost identically used. Having the metric around 0.9, UCMP achieves fairly high path coverage, but not at the cost of per-packet randomization like in VLB, which indicates the effectiveness of path selection in UCMP.

Consistent on different schedules. Fig. 16 presents the UCMP path numbers with another randomly generated schedule. On top is the UCMP group size distribution, and the bottom is the number of unique paths per cycle for a ToR pair. Under this schedule, 89.8% of the UCMP paths are edge-disjoint. We observe consistent results compared to the path numbers of the default schedule shown in Fig. 5a.

D Link Utilization for Data Mining Trace

Fig. 17 shows the ToR-to-ToR and ToR-to-host link utilization for the data mining trace.