Luopan: Sampling based Load Balancing in Data Center Networks

Peng Wang, *Member, IEEE,* George Trimponias, Hong Xu, *Member, IEEE,* Yanhui Geng

Abstract—Data center networks demand high-performance, robust, and practical data plane load balancing protocols. Despite progress, existing work falls short of meeting these requirements. We design, analyze, and evaluate Luopan, a novel sampling based load balancing protocol that overcomes these challenges. Luopan operates at flowcell granularity similar to Presto. It periodically samples a few paths for each destination switch and directs flowcells to the least congested one. By being congestion-aware, Luopan improves flow completion time (FCT), and is more robust to topological asymmetries compared to Presto. The sampling approach simplifies the protocol and makes it much more scalable for implementation in large-scale networks compared to existing congestion-aware schemes. We provide analysis to show that Luopan's periodic sampling has the same asymptotic behavior as instantaneous sampling: taking 2 random samples provides exponential improvements over 1 sample. We conduct comprehensive packet-level simulations with production workloads. The results show that Luopan consistently outperforms state-of-the-art schemes in large-scale topologies. Compared to Presto, Luopan with 2 samples improves the 99.9%ile FCT of mice flows by up to 35%, and average FCT of medium and elephant flows by up to 30%. Luopan also performs significantly better than Local Sampling with large asymmetry.

Index Terms—Data center networks, load balancing, network congestion, distributed

1 Introduction

Data center networks use multi-rooted Clos topologies to provide many equal-cost paths between hosts [4], [18]. To load balance traffic, switches run ECMP—Equal Cost Multi-Path—that forwards packets among equal-cost egress ports using static hashing. Though simple to implement, ECMP's drawbacks are widely recognized in the community. Hash collisions cause flow collisions and congestion, degrading throughput for elephant flows [5], [12], [14] and tail latency for mice flows [7], [8], [25], [37].

Recent work such as Presto [20] proposes to break flows into small flowcells and load balance flowcells across available paths in a round-robin fashion. By transforming the heavy-tailed flows into many smaller flowcells, Presto can better balance the load and improve flow completion time (FCT) for medium and large flows (§2.1). However, in practice most flows are small and only have a few flowcells. We find that in one production network 90% of the flows have less than 6 flowcells (§2.2). This implies that a flow can only utilize a few random paths out of the hundreds available in typical large scale production networks [9], [33]. Further, Presto's round-robin only balances the number of flowcells. It does not work well with link failures and network asymmetry, which are rather common in practice [17]. Even in a symmetric network with uniform flowcells, Presto's round-robin still causes transient congestion in the lower tier of a multi-tier Clos network, because it sequentially uses the ports of a switch first before moving to the next (§2.2). Transient load imbalance still exists with Presto, which degrades the tail FCT for mice flows.

A more robust approach is congestion-aware load balancing advocated by CONGA [6] and HULA [24]. Switches monitor congestion levels for each path and direct a flow or flowlet to the least congested path. This is responsive to changing network conditions, and robust to failures and network asymmetry [6], [24]. To make the best load balancing decisions, prior work strives to collect congestion feedback for *each* path between the source and destination ToR switches. These omniscient schemes perform well in small-scale enterprise networks with simple 2-tier leaf-spine topologies [6]. The challenge is that they have serious scalability and overhead issues that impede the deployment potential in large-scale networks (§2.3). Production networks such as Google's [33], Facebook's [9], and Amazon's [3] use 3-tier or even more complex Clos topologies. For a typical 3-tier Clos network, hundreds of paths exist between any two ToR switches, and a ToR switch can communicate with hundreds of other ToR switches [9]. Thus, omniscient per-path feedback requires storing and tracking a daunting number of paths at *each* ToR in the time scale of RTT (tens of microseconds). Further, acquiring omniscient information involves many switches in the process and makes the control loop slower.

We explore a different direction: what if we use congestion information of just a few random paths for load balancing? To answer this question we design and evaluate Luopan[1], a sampling based load balancing protocol for large-scale data center networks. Inspired by the power of

- *This work was supported in part by Huawei Technologies (contract research no. 9231208) and the Research Grants Council, University Grants Committee of Hong Kong (GRF-11202315). The corresponding author is Hong Xu.*
- *P. Wang and H. Xu are with the Department of Computer Science, City University of Hong Kong, Hong Kong. Email: pewang4-c@my.cityu.edu.hk; henry.xu@cityu.edu.hk.*
- *G. Trimponias is with Huawei Noah's Ark Lab, Hong Kong. Email: g.trimponias@huawei.com.*
- *Y. Geng is with Huawei Montreal Research Centre, Canada. Email: geng.yanhui@huawei.com.*

1. Magnetic compass in Chinese.

two choices in randomized load balancing [26], in Luopan each ToR switch sends probe packets to sample a few random paths towards each active destination ToR switch. It stores the information in a local table, directs flowcells to the least congested path, and *periodically* re-samples the network to refresh the table (§3). Luopan greatly reduces the overhead of maintaining global congestion information, and is more scalable for practical deployment. It also offers a much simpler switch implementation compared to existing work [6], [24].

We conduct comprehensive packet-level simulations in ns-3 using production workloads to assess Luopan's performance (§4). Luopan provides salient performance benefits: with two samples it improves the 99.9%ile FCT for mice flows by up to 35%, and average FCT of elephant flows by up to 30% compared to the congestion-agnostic Presto. Using more samples only marginally improves performance. Luopan achieves salient performance with negligible bandwidth overhead: assuming a 10G network with a probe size of 38B, sampling period of 40μs, and 100 active destination ToR switches, Luopan's probing overhead on each link is 0.38%.

Our evaluation also shows that Luopan outperforms Local Sampling, where a switch locally and independently samples some random egress ports and selects the least congested one for each flowcell [16]. We find that with symmetric topologies and uniform traffic patterns, Local Sampling is only marginally worse than Luopan with global path-wise information. However, when there are link failures or non-uniform traffic, Luopan reduces the 99%ile FCT of mice flows by 4$\times$ over Local Sampling. In these scenarios, congestion happens in the downstream segment of a path, and knowing local information is insufficient to direct traffic away from congested links in the first few hops.

In summary, our contributions are the following:

- We design and implement Luopan, which uses a few samples to achieve global congestion-awareness and effectively balance load for large-scale data center networks.
- We extensively evaluate Luopan with packet-level simulations and further explore parameter settings (including number of samples and expiry time). We demonstrate that two samples are enough to deliver good performance. Our results also show that Luopan can effectively handle topological asymmetry caused by failures without involving the controller.
- We also analyze and establish the effectiveness of Luopan's *periodic* sampling design compared to instantaneous sampling required by power of two choices [26] in the Appendix.

2 Design Decisions

We start by motivating and justifying the key design decisions behind Luopan.

2.1 Load Balancing Granularity: Flowcell

The first key design decision is to adopt flowcells as the load balancing granularity. Per-flow load balancing such as ECMP is ineffective in data centers because traffic is heavy-tailed with many mice flows and a few elephant flows that carry most bytes [5]–[7], [20], [29]. Simply balancing the number of flows does not balance the load on each path. Sub-flow level load balancing achieves better FCT by transforming the heavy-tailed flows into many data units of similar sizes. Presto is the state-of-the-art fine-grained load balancing protocol that breaks a flow into flowcells of at most 64KB [20]. This threshold value is chosen to be the maximum TCP Segment Offload (TSO) size to utilize high speed optimizations provided by the NIC and OS. Most mice flows are also smaller than 64KB and do not suffer from packet reordering. Regardless, existing mechanisms such as modifying the Generic Receive Offload (GRO) handler in the host network stack [20] can be used to minimize the adverse effect of reordering. It has been shown [20] that Presto with flowcells outperforms per-flow ECMP and other sub-flow units such as flowlets [6].

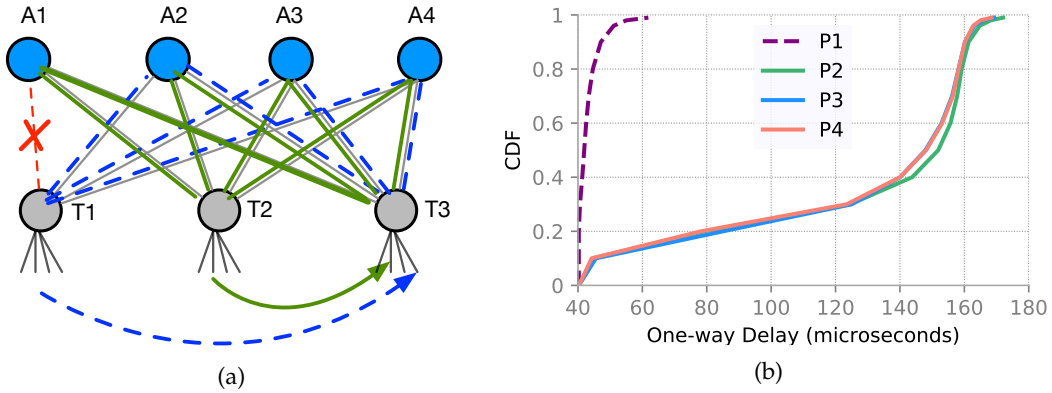
2.2 Congestion-aware vs. Congestion-agnostic

Presto [20] uses round-robin to route flowcells. This congestion-agnostic approach balances the link load well in symmetric topologies and delivers good FCT performance for medium and large flows.

We argue that a congestion-aware approach has advantages over Presto's round-robin design. First, a congestion-aware approach is fundamentally more robust to link failures and topological asymmetries [6]. A congestion-agnostic protocol can handle failures by pruning the spanning tree affected by failure and adjusting the weights of paths over the spanning tree. However it involves slow and expensive controller reconfiguration [20]. Besides, weighted round-robin in a load-oblivious manner could not completely solve load imbalance.

We consider the asymmetric topology in Fig. 1a where the link between switch $A1$ and $T1$ is down. End-hosts connected to $T1$ and $T2$ send TCP traffic to $T3$. Due to the failure, $P1$: $T1{\rightarrow}A1{\rightarrow}T3$ is not reachable from T1. Thus, traffic from T1 can only take $P2$: $T1{\rightarrow}A2{\rightarrow}T3$, $P3$: $T1{\rightarrow}A3{\rightarrow}T3$ and $P4$: $T1{\rightarrow}A4{\rightarrow}T3$ to T3. But $T2$ can take all four paths. Presto equally spreads flowcells from both $T1$ and $T2$ among these available paths. To quantize the impact of asymmetry, we measure one-way delay among all paths from $T2$ to $T3$. Fig. 1b shows the CDF of path delay. Obviously, $P1$ is under-utilized while other paths are heavily loaded. $P1$'s 99%ile delay is much less than the other three paths, by a factor of 2.7x. This shows severe load imbalance. To better utilize the network, more flowcells should be sent to $P1$. By dynamically reacting to congestion and shifting traffic away from hotspots caused by failures, a congestion-aware approach can re-balance the load among paths [6], [16], [24].

Second, even in symmetric topologies, Presto's round-robin can also cause transient congestion and performance problems. To show this we conduct some packet-level simulations on ns-3 for a 12-pod fat-tree fabric. We compare Presto to an "ideal" congestion-aware approach where each flowcell is routed along the least congested path. Background traffic is generated based on the web search workload widely used in the literature [6], [20], [24]. We



(a) (b)

Fig. 1: Sceario of topological asymmetry: (a) Link failure between $A1$ and $T1$ causes topological asymmetry; End-hosts from switch $T1$ and $T2$ send traffic to $T3$; (b) CDF of one-way delay measured for paths from switch $T2$ to $T3$

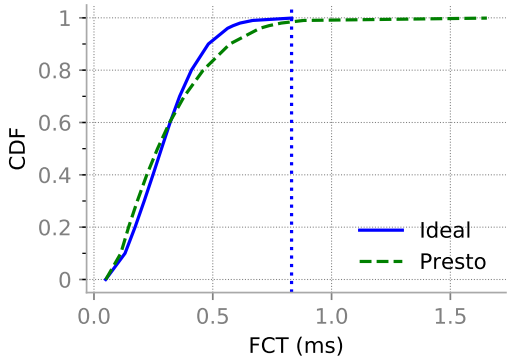


Fig. 2: Distribution of FCT for 2KB mice flows.

then issue 20K flows of 2KB between random pairs of hosts within 250ms and collect their completion time. Fig. 2 shows the CDF. We observe that while they do provide similar average-case performance, Presto's 99.9%ile FCT is 2× larger than that of the "ideal" congestion-aware scheme.

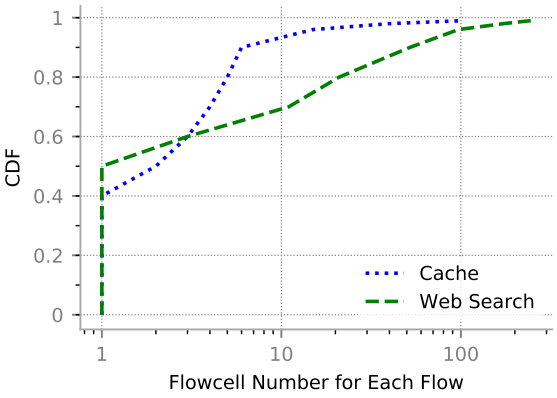


Fig. 3: Distribution of number of flowcells in a flow.

The reason for round-robin's inefficiency is twofold. First, the number of flowcells in most flows is much smaller than the number of paths available. Fig. 3 depicts that 90% of flows have less than 50 flowcells for the web search workload [7] and 6 flowcells for the Facebook workload [29], another commonly used flow size distribution. Typically hundreds of paths exist between any pair of ToR switches in large scale production data centers [9], [33]. Thus a flow only leverages a small portion of all available paths during its lifetime.

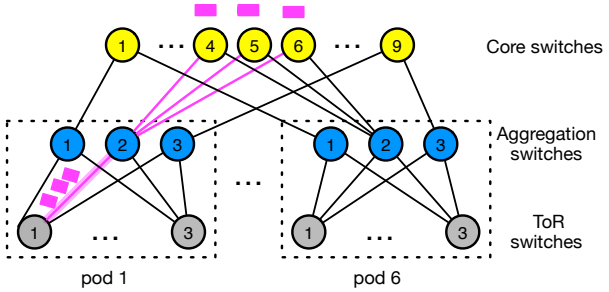


Fig. 4: Example of the clustering effect of round-robin load balancing. Even with flowcells, transient congestion may still happen.

Second, the limited number of paths selected by round-robin cluster around a few switches in the lower layers of the topology. Take Fig. 4 with a 6-pod fat-tree as an example. A flow from ToR switch 1 has 3 flowcells to send to a host in pod 6. The first flowcell is routed by Presto to a random path, say aggregation switch 2 then core switch 4. The next two flowcells then take the successive egress ports at aggregation switch 2 to go to core switches 5 and 6. The three paths share the same ToR-aggregation link as highlighted, while other aggregation switches are not used at all. In some extreme cases, a few links may be repeatedly exploited by multiple flows over a short period, which can lead to queue buildup. The clustering effect of round-robin becomes more serious with more layers in the topology, which are very common in large-scale data centers [29], [33]. Selecting paths randomly can alleviate clustering but they still cause long tail latency due to the randomness.

2.3 Sampling vs. Omniscient

Existing congestion-aware load balancing schemes such as CONGA [6] and HULA [24] maintain congestion information for all paths connecting a ToR switch pair. This is done by either frequently (every tens of microseconds) flooding

probes to all paths as in HULA [24], or opportunistically piggybacking information on data packets as in CONGA [6].

This omniscient approach works well in small-scale enterprise networks with 2-tier leaf-spine topologies [6]. It however suffers from serious scalability challenges that make it difficult to deploy in practice. Production networks such as Google's [33], Facebook's [9], and Amazon's [3] use 3-tier or even more complex Clos topologies. For a typical 3-tier Clos network, hundreds of paths exist between any two ToR switches, and a ToR switch can communicate with hundreds of other ToR switches [9], [33]. For example a ToR switch in Facebook's Altoona data center network has ∼220K paths to the fabric [9]. Google's Jupiter network has even more paths due to its 7-layer structure [33]. Thus, omniscient per-path feedback requires tracking a huge number of paths at *each* ToR switch at the time scale of a few RTTs. Moreover, flooding probes every tens of microseconds imposes significant bandwidth overhead to the network and processing overhead to the switches. Even opportunistic piggybacking on data packets is very difficult, since there would not be enough concurrent flows that happen to traverse all paths at the same time.

Thus, Luopan uses randomized sampling to practically exploit congestion feedback in a large-scale topology. Sampling has much less overhead and scales better than the omniscient approach.

3 Design

We present the design of Luopan in this section. Luopan is a data plane protocol. ToR switches periodically send probe packets to sample the congestion metrics of two random end-to-end paths. The information is maintained in a table at the ToR switch for each active destination ToR (§3.1). Load balancing decisions are made at flowcell granularity: a flowcell is routed to the least congested sampled path (§3.2).

3.1 Path Sampling

We explain Luopan's sampling mechanism, including the congestion metric used, probe packet header, the sampling process, and its bandwidth overhead.

Congestion Metric. Queue length [16] and link utilization [6] [24] are two commonly used congestion metrics in the literature. Luopan builds upon flowcells [20] that are at most 64KB. Flowcells are more sensitive to queuing delay rather than link utilization, since each of them finishes quickly. Besides, transient congestion caused by collisions among flowcells happens in a short time period, which is hard to be captured by link utilization. We therefore use queue length as the congestion metric to make path selection decisions. We experiment with both metrics and find that queue length has much better performance.

Probe packet header. Sampling is performed using probe packets. As illustrated in Fig. 5, each probe packet is 38 bytes with a 14B Ethernet header, a 4B probe header, and a 20B IP header. Probe header contains the following information:

- **Path ID, PID (24 bits):** This field specifies the complete path to be explored by the packet from the source ToR to the destination ToR switch. 24 bits are sufficient since the number of paths between a pair of ToR switches rarely reaches 16 million.
- **Type (1 bit):** This field is used to identify probe and ACK-probe packet. It is set to 0 for probe packet and 1 for ACK-probe packet.
- **Quantized Congestion Metric, QCM (7 bits):** This field contains the sum of queue lengths over each hop of an end-to-end path, quantized within 7 bits.

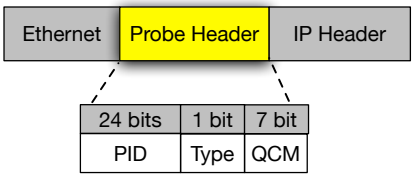


Fig. 5: Header format of a probe packet.

Sampling Process. Fig. 6 illustrates the sampling process in Luopan. The source ToR switch can be configured to send d probes to sample d paths per destination ToR switch. As a probe packet traverses each switch on the path, the instantaneous queue length of the corresponding egress port is added to the current value in the QCM field of the probe header. The destination ToR immediately generates an ACK-probe packet in response to each probe packet. The IP and probe headers are copied from the probe packet to the ACK-probe packet, with the source and destination IP addresses swapped for the reverse direction, and probe type changed to 1. Upon receiving the ACK-probe, the source ToR retrieves the destination ToR switch address, path ID, and congestion metric, and updates the corresponding entry in a congestion table. CONGA [6] shows that switches can actually maintain a large number of entries (i.e., 64K) for the congestion table. Besides, entries without hits in a certain time period are removed.

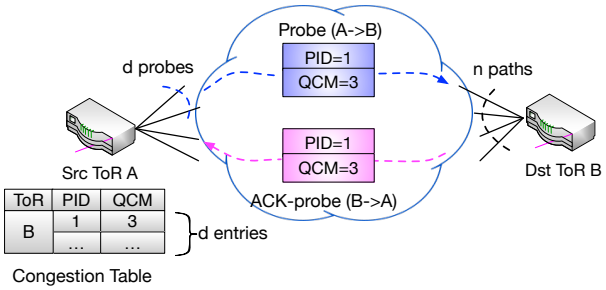


Fig. 6: Luopan's sampling process. The source ToR samples congestion of d paths over a total of n paths available. It updates its congestion table upon receiving the ACK-probe packet from destination ToR switches.

To maintain up-to-date information, entries in a congestion table expire after a short time, and new probes are sent out immediately to another d random paths. We empirically find that Luopan with expiry time of $40\,\mu s$ and 2 samples achieves a good balance between performance and sampling overhead in a large scale 10Gbps network with various settings (§4.3, §4.4). We thus choose them as the default parameters for Luopan. Although parameters for Luopan in this paper are static, in principle it is also possible to dynamically adapt them based on network congestion.

Bandwidth overhead. We now discuss the bandwidth overhead of our sampling approach. Overhead is determined by number of probes and expiry time of the congestion table entry. Specifically, the average probe overhead on each network link is:

$$\frac{2 * numActiveDstToRs * probeSize}{expiryTime * numLinks * linkBw}.$$

$numActiveDstToRs$ is the number of active destination ToR switches that current flows talk to. A source ToR switch needs to sample at least two paths from itself to each active destination ToR switch. $numLinks$ is the number of links for each ToR switch to send traffic to its upper layer switches. $numLinks * linkBw$ represents the total capacity to carry traffic from each ToR switch to upper layer switches. For a network with 10G links where $numActiveToRs$ is 100, $probeSize$ is 38 bytes, $expiryTime$ is 40us, and $numLinks$ is 40, the bandwidth overhead is only 0.38% of the 10G link capacity, compared to HULA's 7.6% overhead [24] with the same settings.

3.2 Path Selection

Similar to Presto, a sending host in Luopan adds a sequentially increasing flowcell ID into each packet. Flowcell ID increases by 1 for every 64KB data sent out. The source MAC address field is used to hold the flowcell ID [20]. Luopan recognizes the flowcell ID for each packet by matching the source MAC address field.

Luopan makes path selection decisions on the first packet of each flowcell, and records the selected path ID in a flow table. Flow table is the second table maintained at a ToR switch; the first table being the congestion table discussed in §3.1. Each flow table entry records flow ID (i.e., flow's 5-tuple), flowcell ID, path ID, and time last seen. When a new packet arrives and matches both the flow ID and flowcell ID, it is a subsequent packet from a recorded flowcell and thus routed to the selected path recorded. If it only matches the flow ID, it belongs to the next flowcell and the new flowcell ID is recorded. If it does not match any entry in the flow table, it is the first packet of a flowcell from a new flow and a new entry is added to the flow table. For the first packet of each flowcell, a new path selection decision should be made by looking up the congestion table, and recorded in the flow table. If it does not match any dest ToR in the congestion table, Luopan creates a new entry in the congestion table, and chooses a random path as the path selection result. Luopan then sends probe packets to the corresponding destination ToR switch to start the sampling process. After probe packets come back and update the congestion table entry, subsequent flowcells can use the information to choose better paths. Flow table entries that have not been updated for a long time (e.g., 1ms) will be removed.

3.3 Source Routing

In Luopan, we implement source routing to forward probe packets and data packets along designated paths. Source routing is not a new concept. There are several implementations using ECMP spoofing [28], MPLS, VXLAN tag [6], or shadow MAC [20]. Luopan uses a source routing implementation proposed in Sourcey [21], which is simple and scalable. The source ToR translates the selected path ID into output port numbers at each switch on this path, and pushes them as a stack of labels into the packet headers. Each switch simply looks up the label as the output port for each packet and then pops it off. The overhead is negligible since the diameter of data center networks is usually small.

4 Evaluation

We evaluate Luopan's performance using packet-level ns-3 simulation. We answer three key questions here:

- How does Luopan perform in typical data center network topologies of various scales running realistic workloads (§4.2, §4.4)?
- How does Luopan perform with different parameter settings (§4.3)?
- Is Luopan effective in handling Incast (§4.5), non-uniform workloads (§4.6) and topological asymmetry (§4.7)?

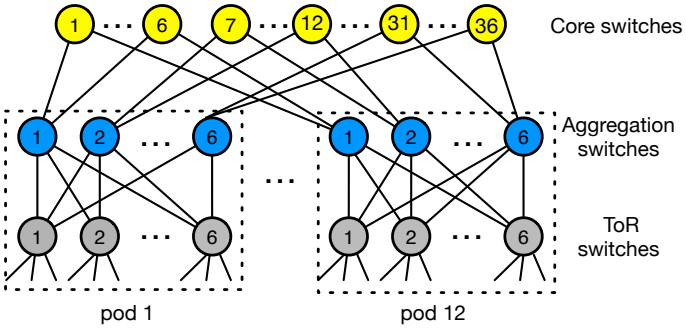


Fig. 7: 12-pod fat-tree topology used for evaluation experiments.

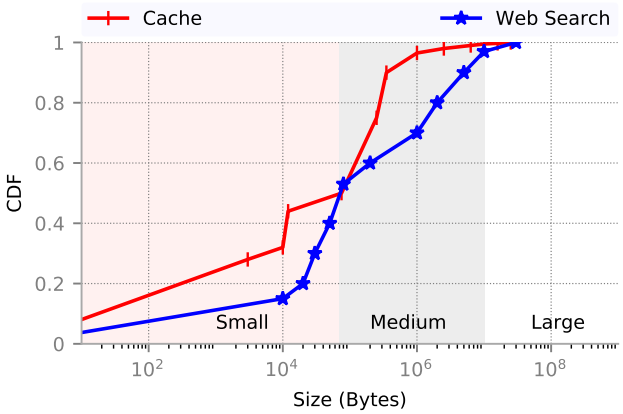
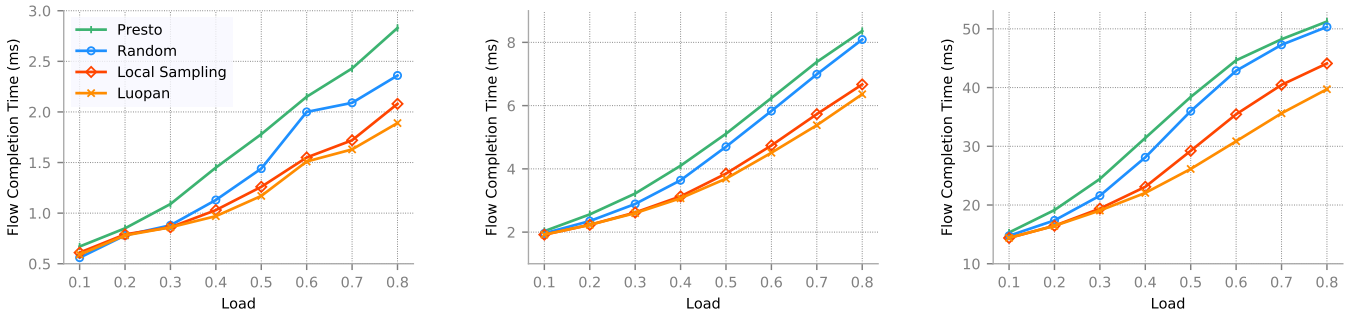


Fig. 8: The flow size distributions used in the evaluation.

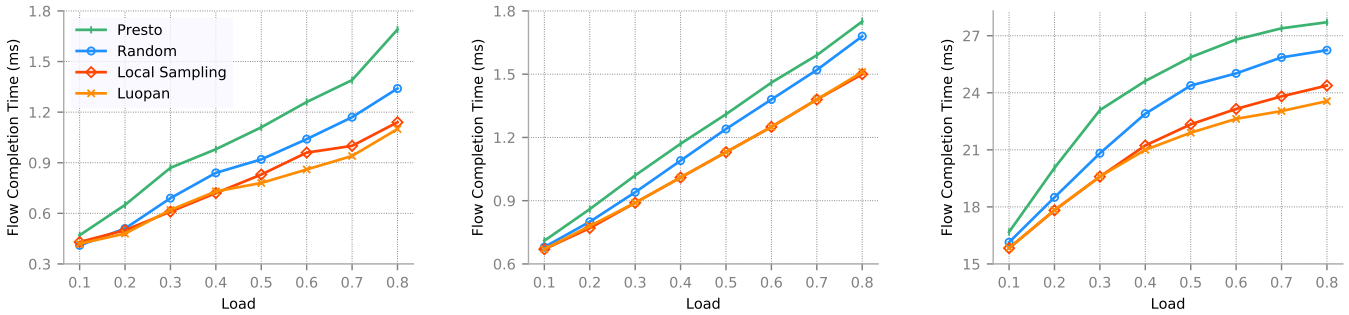
4.1 Methodology

Fabric topology. As shown in Fig. 7, we use a 12-pod fat-tree [4] as the baseline topology. The fabric consists of 432 hosts and 36 core switches. There are 36 equal-cost paths between any pair of ToR switches across pods. Link capacity



(a) (0,64KB]: 99.9%ile tail (b) (64KB,10MB]: Avg (c) (10MB, ∞): Avg

Fig. 9: Flow completion time for web search workload in 12-pod fat-tree with oversubscription of 4:1.



(a) (0,64KB]: 99.9%ile tail (b) (64KB,10MB]: Avg (c) (10MB, ∞): Avg

Fig. 10: Flow completion time for cache workload in 12-pod fat-tree with oversubscription of 4:1.

is 10Gbps, and the fabric RTT is ∼40μs. Each switch port has a buffer of 300KB. In the non-oversubscribed topology, each ToR switch connects to 6 aggregation switches and 6 end-hosts. We vary the number of end-hosts to achieve different over-subscription ratios. Our baseline topology uses 4:1 over-subscription ratio.

Realistic workloads. We use two realistic workloads from production data centers running web search [7] and cache jobs [29]. Both workloads are heavy-tailed as shown in Fig. 8: most flows are mice but elephant flows carry a large fraction of traffic. Each host continuously samples the flow size distribution, and sends a flow to a random receiver in a different pod. Flow arrivals follow a Poisson process, and we vary the mean inter-arrival time to simulate different network loads ranging from 0.1 to 0.8.

Performance metric. We use flow completion time (FCT) as the primary performance metric. We consider 99.9%ile FCT for mice flows (0, 64KB], and average FCT for medium flows (64KB, 10MB] and elephant flows (10MB, $+\infty$). These settings are in accordance with prior work [6], [8], [15], [20].

Schemes compared. We use the following load balancing protocols.

- *Luopan*: Our sampling based scheme. The expiry time is 40 µs. We select the better one of *two* randomly sampled paths. These parameters are fixed in all experiments except the sensitivity analysis (§4.3).
- *Local Sampling*: Each switch locally samples all egress ports and sends a flowcell to the least congested one. A flowcell is agnostic to congestion of subsequent links along its path. Thus, Local Sampling leads to worse performance in asymmetric topologies (§4.7).
- *Random*: Each flowcell selects a random path.
- *Presto*: This is the state-of-the-art load balancing protocol at flowcell granularity [20]. A flow sends its first flowcell to a random path. Consecutive flowcells in the same flow are then sent in a round-robin fashion to other paths.

4.2 Overall Performance

In this section, we show the overall performance of Luopan in a large scale network. As explained in §4.1, Luopan uses expiry time of 40 µs with 2 samples. Fig. 9 and Fig. 10 show the FCT results for both workloads, with network load varying from 0.1 to 0.8. Compared to Presto, Luopan improves the 99.9%ile FCT for mice flows by 8% to 35% for the web search workload, and by 10% to 35% for the cache workload. Compared to Random, Luopan improves the 99.9%ile FCT for mice flows by up to 24% for the web search workload, and up to 20% for the cache workload. In terms of the average FCT for medium flows, Luopan achieves up to around 25% improvement for the web search workload and 13% for the cache workload. Luopan also improves FCT for elephant flows by as much as ∼30%. Presto's bad FCT for mice flows can be explained by the clustering effect as discussed in §2.2. The same low layer links are repeatedly used by several successive flowcells of a flow. This causes transient imbalance among the links and temporary increase in queue occupancies. Flowcells experience longer queuing delay in the network which causes large tail latency. More-

over, elephant flows also suffer from the imbalance. Due to the impact of flowcell-based load balancing, packets within the same flow traverse different paths and do not always arrive in order. To prevent reordering, out-of-order packets need to be re-sorted below TCP layer. Earlier out-of-order packets need to wait a long time for the right order packets before being submitted to the TCP layer. Significant load imbalance exacerbates the adverse effect of reordering. This undoubtedly harms TCP throughput.

Another interesting observation is that Local Sampling closely tracks Luopan and performs only slightly worse in all metrics. Since the topology is symmetric and each pair of sender and receiver is randomly selected for each generated flow (i.e., the number of senders and receivers is roughly equal), the upstream path segments observe similar congestion as the downstream path segments. Local Sampling can balance traffic on upstream path segments well. Thus, it performs close to Luopan in these scenarios.

4.3 Sensitivity Analysis

In this section, we conduct sensitivity analysis for two important parameters in Luopan: expiry time of the congestion information, and number of samples per destination switch.

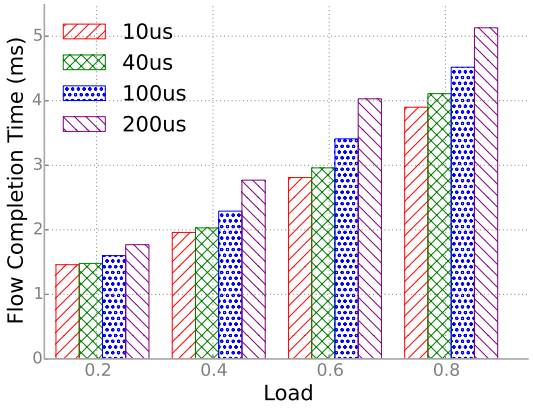


Fig. 11: Sensitivity to expiry time for Luopan with two samples.

Impact of expiry time: Entries in the congestion table expire periodically (every T seconds) to avoid stale information. More frequent sampling accelerates the congestion information update process at the cost of more probing overhead. We aim to find a good setting of expiry time to strike a balance between performance and overhead. Fig. 11 shows the overall average FCT of all flows with various expiry time settings. Intuitively, Luopan's performance suffers when expiry time increases. Interestingly, when increasing T from $10\,\mu s$ to $40\,\mu s$, Luopan experiences marginal performance degradation for moderate network load (0.2, 0.4 and 0.6). Even for high load of 0.8, performance degradation of $40\,\mu s$ compared to $10\,\mu s$ is only 5%. As T increases to $100\,\mu s$ and $200\,\mu s$, performance degradation is more significant. An expiry time of $40\,\mu s$ achieves a good balance between performance and overhead. As network load and network speed increase, the arrival rate of flowcells increases, and one may need to reduce the expiry time in order to avoid the likelihood that many flowcells are routed to the same path. Luopan can also empirically determine the best value for the expiry time.

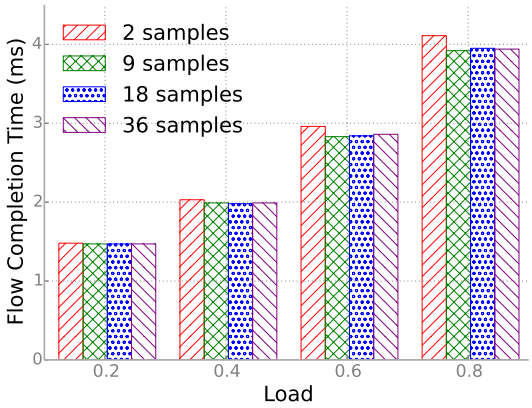


Fig. 12: Sensitivity to number of samples with 40us expiry time.

Impact of number of samples: We vary the number of samples from 2 to 36 in Luopan and investigate its impact on the overall average FCT for all flows. Note that the number of paths in the 12-pod fabric is 36 between a pair of ToR switches. Luopan with 36 samples effectively has omniscient information. We observe that performance with different numbers of samples is similar for low network loads. Fig. 12 shows the effect of number of samples with $40\,\mu s$ expiry time. As the number of samples increases, the overall average FCT slightly decreases. Luopan with 2 samples achieves similar performance with more samples. Even in high network load (0.6 and 0.8), the performance degradation is within 5%. We also find that increasing the number of samples does not improve performance much after the number exceeds 9 even for high network load. One reason is that paths chosen from 9 sampled paths are good enough. Another reason is that increasing the number of samples exacerbates the synchronization effect of choosing the best path. Multiple flowcells from different ToR switches may synchronously choose paths that share the same under-utilized links in the higher layers of the topology. The side effect of synchronization negates to certain extent the benefits of more samples.

4.4 Network Fabric Settings

In this section, we present Luopan's performance gains under fabrics with different settings, particularly, different network scales, network over-subscriptions, different transport protocols, and buffer sizes. We evaluate performance of Luopan, Random, Local Sampling and Presto running the web search workload.

Impact of network scale. Fig. 13 shows the average FCT for all flows in 6-pod, 12-pod, and 24-pod fat-tree networks at 60% traffic load. We make the following observations. First the performance gap between Presto and other schemes becomes larger as the network scale increases. Presto outperforms Random in the 6-pod fat-tree, but its performance drops drastically for larger scale fat-tree. In the 12-pod fat-tree, the improvement of Luopan over Presto becomes 40%. As explained in §2.2, the clustering effect of Presto becomes more serious in a larger network, since consecutive flowcells are more likely to be placed onto the same lower layer

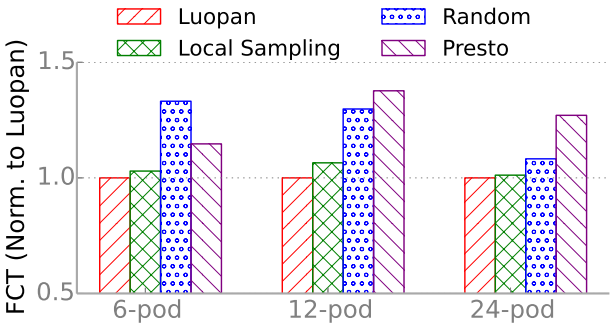


Fig. 13: Comparison of average FCT for all flows among Luopan (2 samples with expiry time of 40 µs), Random, Local Sampling, and Presto in a fat-tree with different scales.

links in a short period. Besides, we also observe consistent performance improvement of Luopan over Random. The improvement in overall average FCT of Luopan over Random ranges from 10% to 35% for all network scales.

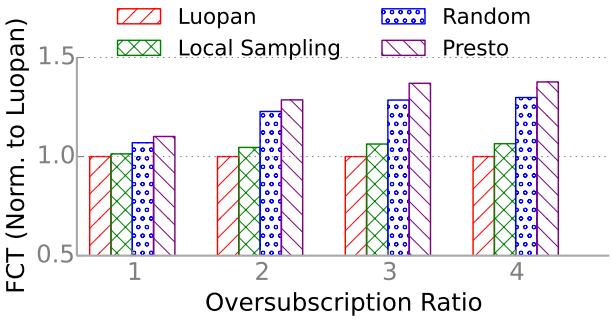
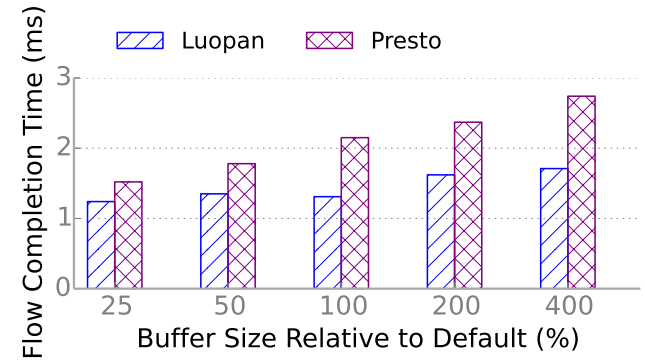


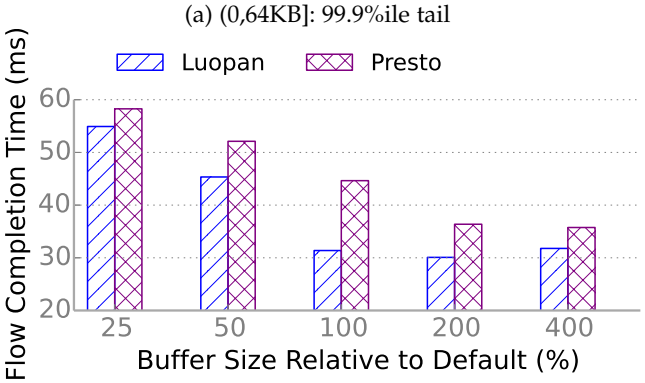
Fig. 14: Comparison of average FCT for all flows among Luopan (2 samples with expiry time of 40 µs), Random, Local Sampling, and Presto in a 12-pod fat-tree with different over-subscription ratios.

Impact of network over-subscription. We also vary the over-subscription ratios to simulate different extents of in-network contention, and see its impact on Luopan's benefits. We adjust the number of hosts to over-subscribe a 12-pod network with ratios from 2 to 4. Fig. 14 shows the comparison result using again the overall average FCT for all flows at 60% traffic load. In the non-oversubscribed network, congestion mostly occurs at the network edge. The improvement of Luopan over Random is marginal. This makes sense since Luopan aims at alleviating in-network congestion with better load balancing. With the increase of over-subscription ratio, in-network congestion becomes more severe. Luopan then outperforms Random by around 23%, and Presto by 22% to 38%. In practice a data center network is usually over-subscribed at 4:1 or higher [29] for cost reasons. Therefore Luopan is able to provide salient performance benefits in realistic network settings.

Impact of switch buffer size. To show Luopan's robustness to different buffer sizes, we vary buffer size from 25% to 400% of the default buffer size (300KB). Fig. 15 shows the 99.9%ile FCT of mice flows and average FCT of elephant flows. With the increase of buffer size, tail FCT of mice flows for both Luopan and Presto becomes larger. Beyond 100% of the default buffer size, FCT of Luopan becomes stable, while that of Presto keeps increasing. This is because over-buffering exacerbates congestion of the worst paths, which prolongs the tail latency of mice flows. However, Luopan is globally congestion-aware, which effectively avoids paths with larger queue length, and more robust to the increase of buffer size. Luopan consistently outperforms Presto by 19% to 37% on 99.9%ile FCT of mice flows. For elephant flows, we observe that performance of both Luopan and Presto becomes better. This is because larger buffers lead to higher utilization of links, which effectively increases the throughput of elephant flows. Luopan still achieves 6% to 30% performance gains over Presto.



(a) (0,64KB]: 99.9%ile tail



(b) (10MB, ∞): Avg

Fig. 15: Effect of buffer size on overall average FCT of all flows. Luopan uses 2 samples with expiry time of 40 µs. Default buffer size is 300KB.

Impact of transport protocols. Besides TCP, DCTCP is a commonly used congestion control protocol in data center networks [7]. DCTCP uses a fixed marking threshold at the switch queue, and marks based on the instantaneous queue length. It estimates the extent of network congestion based on the fraction of marked packets of every window of data, and then reduces window size in proportion to it. We repeat the previous experiment in §4.2 for DCTCP with network load varying from 0.3 to 0.8. We set the ECN marking threshold at the switch queue to be 100KB. Fig. 16 compares the overall average FCT. All schemes achieve better performance with DCTCP than TCP. Luopan still achieves the best performance with DCTCP, outperforming Presto by around 15% on the overall average FCT. Under DCTCP, the performance gap between Luopan and Presto becomes smaller. This is not surprising, since DCTCP effectively

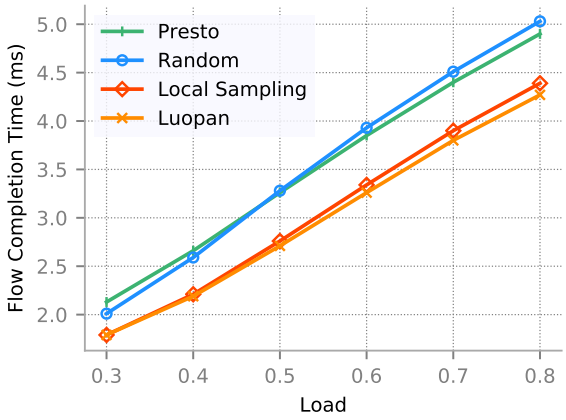


Fig. 16: Overall average FCT for all flows among Luopan (2 samples with expiry time of 40 µs), Random, Local Sampling, and Presto in a 12-pod fat-tree using DCTCP.

keeps queue sizes small and thus alleviates congestion of the worst path.

4.5 Incast

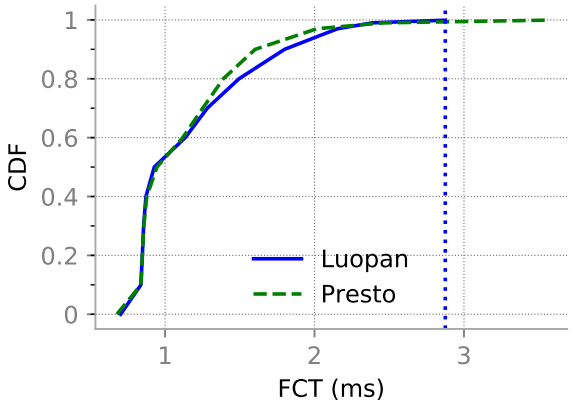
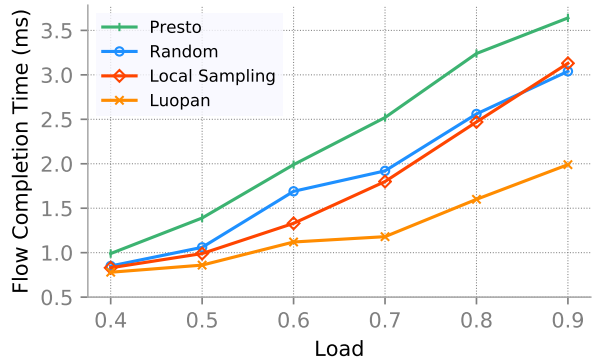
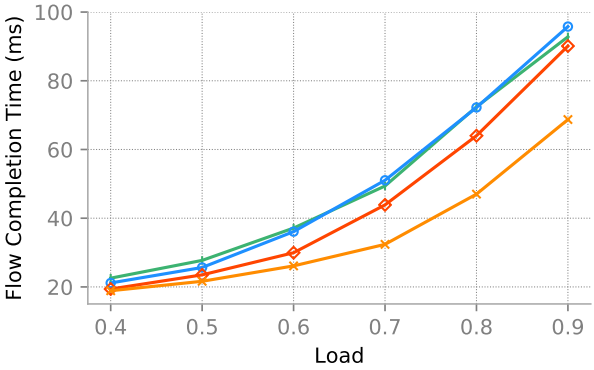


Fig. 17: Incast: CDF of Incast job completion time. Background traffic is web search workload at 80% load.

Incast is a common traffic pattern in datacenters which arises in many applications such as web search and MapReduce. We evaluate Luopan's performance for Incast traffic pattern. In our experiment, a client requests for a 1MB file stripped across 20 other different servers. Upon receiving the request, these servers respond with 1MB/20 of data in a synchronized pattern. The last response determines the completion time of an Incast job. We launch 7.5K Incast jobs within 250ms and measure the completion time of each Incast job. We also generate background traffic using the web search workload at 80% network load. The minimum retransmission timeout is 200 µs and transport protocol used is DCTCP. In Fig. 17, Luopan outperforms Presto by 20% on the 99.9%ile tail completion time. In Incast traffic pattern, congestion mostly occurs on the receiver side. In-network load balancing does not help to alleviate congestion on the last hop. Luopan still provides benefits because it reduces in-network queuing delay for all flows.



(a) (0,64KB]: 99.9%ile tail FCT



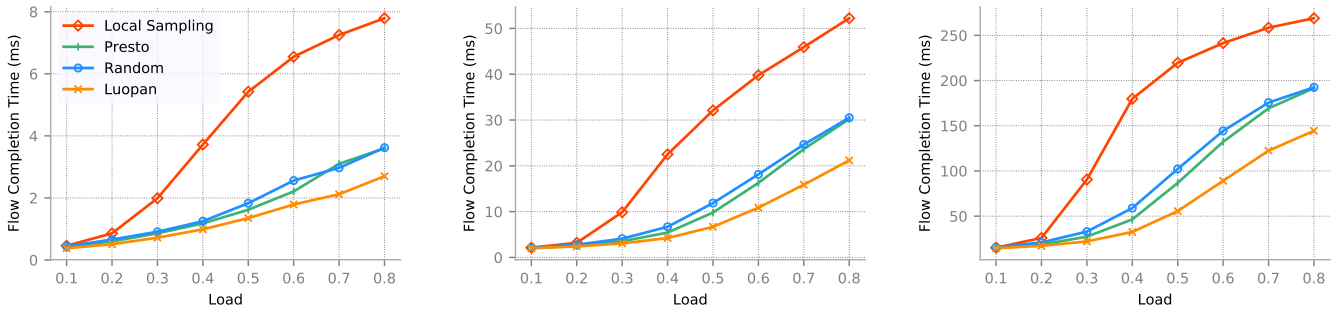
(b) (10MB, ∞): Avg FCT

Fig. 18: Comparison in 99.9%ile FCT for mice flows and average FCT for elephant flows among all schemes under non-uniform workloads. Luopan uses 2 samples with expiry time of 40 µs.

4.6 Non-uniform Workload

As discussed in §4.2 and §4.4, Local Sampling performs similar to Luopan in symmetric topologies with uniform workloads. In this section, we consider the non-uniform workload scenario, and highlight the robustness of Luopan with global information compared to Local Sampling. Except for the traffic pattern, the other experimental settings are same as §4.2 and §4.4. In this experiment, we compare all four schemes with non-uniform workloads.

To create non-uniform workloads, clients from one pod repeatedly retrieve data from servers in the other two pods. The data size follows the web search workload. We set the network load between 40% to 90% with respect to the aggregate link capacity to a single pod. Clearly, now the downstream paths are more likely to be the bottleneck. As a result, most flowcells suffer congestion in the downstream direction to the clients. To better balance traffic, downstream congestion information is essentially required. However, neither Local Sampling nor congestion-agnostic schemes, including Presto and Random, can utilize downstream congestion information. The results in Fig. 18 confirm that Luopan is significantly better than other schemes in terms of the 99.9%ile FCT of mice flows and average FCT of elephant flows over all loads. Luopan outperforms Presto by 21% to 53%, Random by 8% to 39%, and Local Sampling by 6% to 36% on the 99.9%tile FCT of mice flows. For elephant flows, Luopan reduces the average FCT by up to 35% over



(a) (0,64KB]: 99%ile tail (b) (64KB,10MB]: Avg (c) (10MB, ∞): Avg

Fig. 19: FCT for web search workload in a 6-pod fat-tree with 10 randomly selected aggregation-core link failures.

Presto and Random, and 26% over Local Sampling. Local Sampling performs roughly on par with Random. This is because making path selection decisions based on per-hop congestion cannot avoid the hotspots in the downstream direction, and Local Sampling essentially degrades to Random.

4.7 Topological Asymmetry

All experiments we conducted so far are for symmetric topologies. We have validated that Luopan works well in different settings. In data center networks, asymmetry is bound to happen: link failure events are common, which lead to topological asymmetry. Balancing traffic becomes challenging in an asymmetric topology due to unequal available capacity of paths between one or more source/destination pairs. In this section, we evaluate the performance of Luopan in a 6-pod network with 10 random aggregation-core link failures. Network settings are the same as in §4.2. We run web search workload with network load from 0.1 to 0.8. We do not consider the reaction time of the routing protocol to handle failures.

Fig. 19 shows the performance comparison among all schemes in the asymmetric fat-tree topology. Luopan is particularly effective in handling asymmetry, achieving up to 31% lower 99%ile FCT for mice flows and ∼35% lower average FCT for elephant flows than Presto and Random. Interestingly, we observe that Local Sampling performs the worst. Compared to Luopan, it is up to 4× worse in 99%tile FCT for mice flows, 5.3× worse in average FCT for medium flows and 5.5× worse in average FCT for elephant flows.

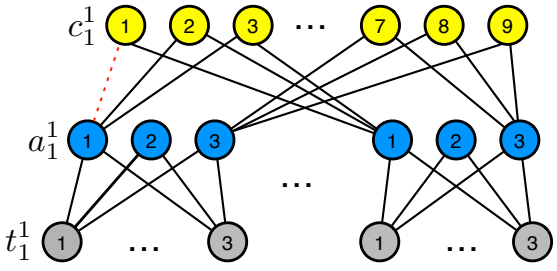


Fig. 20: The link between a_1^1 and c_1^1 fails. There are only two links available from a_1^1 to core switches after link fails.

We now use a simple example in Fig. 20 to explain why Local Sampling does not handle topological asymmetry well and performs the worst. After the link between a_1^1 and c_1^1 fails, the aggregate bandwidth from a_1^1 to core switches suffers a $\frac{1}{3}$ loss. Uplinks from ToR switches to a_1^1 then appear less utilized than other uplinks, since most traffic is congested at a_1^1. Therefore, Local Sampling is more likely to choose a_1^1 at each ToR switch, which actually exacerbates congestion at a_1^1 and leads to inferior FCT.

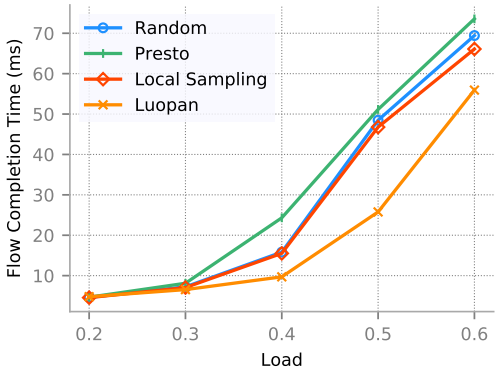


Fig. 21: Overall average FCT for web search workload in an asymmetric leaf-spine topology.

We further conduct experiments in an asymmetric 2-tier leaf-spine topology when a fabric link fails (Fig. 1a in §2.2). We use the web search workload. Hosts under switch $T1$ and $T2$ send traffic to those under $T3$. Fig. 21 shows the overall average FCT for the web search workload. Since the aggregate bandwidth from $T1$ to spine switches faces a 25% loss, we only consider network loads up to 60%. Luopan still performs the best among all schemes. Compared to Presto, Luopan achieves up to 55% lower overall average FCT. As opposed to the results for the 3-tier fat-tree topology, Local Sampling now performs better than Presto and similar to Random. This is because Local Sampling can now recognize the uplink from leaf switch with less congestion and shift traffic to it. Yet Local Sampling is not able to handle congestion and asymmetry at downstream links.

We now explain why Luopan performs the best with global congestion information. For congestion-agnostic schemes like Presto and Random, they try to evenly distribute flowcells among available paths. Flows from $T1$ and $T2$ equally share the bottleneck downstream links $A2 \rightarrow T3$, $A3 \rightarrow T3$ and $A4 \rightarrow T3$. Assume that TCP has been in the

steady state, flows from $T2$ send at 5Gbps (half of full link capacity) at $P2$, $P3$, and $P4$. Since loads on each path are roughly equal, $T1$ also keeps the rate of flows on $P1$ to be 5Gbps. This leads to 50% of $P1$'s capacity being wasted. $T3$ should send twice more traffic on $P1$ than other paths. We measure the number of flowcells sent to each path from $T2$ to $T3$ to validate the effectiveness of Luopan. Table 1 reports the results and shows that Luopan roughly sends twice more flowcells on $P1$ than other paths. Presto, Random, and Local Sampling are not able to react to congestion at downstream paths, still evenly sending traffic to each path.

	P1	P2	P3	P4
Luopan	42.8%	18.7%	19.1%	19.4%
Presto	22.5%	27.0%	24.9%	25.6%
Random	23.0%	25.9%	25.4%	25.7%
Local Sampling	24.6%	25.2%	24.9%	25.3%

TABLE 1: Percentage of flowcells sent on each path.

5 Discussion

In this section, we elaborate on certain design choices, and discuss possible future improvements.

Adaptive design of Luopan. As discussed in §4.3, more frequent sampling will improve flow performance but at the cost of more probing overhead. To make Luopan simple and practical, expiry time is kept fixed. However, static expiry time is not able to strike the best balance between performance and overhead under changing network dynamics. When no congestion is present, there is no need to change path and we can slow down sampling. On the other hand, the sampling frequency should increase when congestion is observed. If all paths are heavily congested, we should not exacerbate congestion with more sampling. We should have a range for expiry time as well. In this way, adaptive probing could further reduce probing overhead while maintaining good performance.

Choice of congestion metric. In our design, we measure congestion using queue length. ECN is also a possible solution [23], but ECN only reflects congestion when queue length exceeds a threshold. It is less accurate in measuring congestion than queue length, but available at many commodity switches. Another direction is to use end-to-end delay as congestion signal [27]. Recent NICs provide support for timestamping of packet events and hardware-generated ACKs, which allows us to measure data center RTTs in high precision. Luopan is able to use all the above metrics based on the requirements of data center operators.

Effect of reordering buffer. Luopan adopts flowcell as the load balancing granularity, and spreads flowcells among multiple equal-cost paths. A major concern is the impact on TCP performance. To handle reordering, a reordering buffer is implemented below TCP to re-sort out-of-order packets. It guarantees that out-of-order packets observed by TCP are due to packet losses, and not the load balancing protocol. TCP then correctly reduces transmission rate. The challenge is that the size of the reordering buffer needs to be carefully tuned. Accumulating too many packets in the reordering buffer could cause traffic burst after all buffered packets are submitted to the TCP stack. Luopan balances load more effectively and thus alleviates the impact of reordering on TCP. More sophisticated reordering schemes can be investigated in future work.

Implementation in programmable switches. Luopan requires randomized sampling and maintaining sampled congestion information for flows. These operations are supported by the emerging programmable switching ASICs such as Cavium's XPliant [1] and Barefoot's Tofino chips [2] even at line rate. Recent works [24], [32] have demonstrated the feasibility of periodic probing and maintaining congestion information in programmable switches with the P4 language. Luopan has simple packet processing logic and does not require complex computation, and programmable switches are able to run Luopan at line rate.

Deployment. Luopan requires modifications at commodity switches, which hinders near-term deployment in the real world. Yet, it is simple enough to be implemented with programmable switches. We wish to evaluate Luopan in real testbed, but lack the hardware to build such a networked system with emerging programmable switches. Our simulations provide the first step to analyze Luopan's performance. We expect to deploy Luopan in programmable switches in the near future.

6 Related Work

We now discuss prior work closely related to Luopan.

There are in general two lines of work dealing with ECMP's drawbacks in data center networks. The first is centralized routing that aims to assign elephant flows to good paths based on global network view. Schemes such as Hedera [5], MicroTE [11], DevoFlow [13] and myopic flow scheduling [31] rely on a central controller to poll network utilization information and modify flow entries at switches. This approach is too slow to react to transient congestion for mice flows. It also poses scalability challenges for commodity Openflow switches [13]. To improve scheduling latency at scale, some randomized myopic algorithms [31] also leverage the power of two choices to achieve low complexity.

The other thread is distributed data plane load balancing, which is more related to Luopan. A number of works focus on sub-flow level load balancing. Packet spraying adopts per-packet load balancing [14] [12], which induces packet reordering and interacts poorly with TCP. Flare [22] and Presto [20] break flows into flowlets or flowcells. LetFlow [34] works with flowlet switching and randomly selects path for each flowlet. It naturally leverages the elasticity of flowlet switching to balance the traffic on different paths. Luopan builds upon Presto with flowcells [20] as the state-of-the-art sub-flow load balancing. Congestion-aware load balancing schemes are also studied to improve the robustness of the congestion-agnostic approach. LocalFlow [30] uses local load information to reroute flows based on TCP sequence numbers. DRILL [16] is a per-packet load balancing scheme based on local congestion information. CONGA [6] and HULA [24], on the other hand, use global path-wise information. As discussed, they require omniscient per-path information for all switches, and do not scale well to large production networks. Expeditus [35] solves the

scalability issue, but it specifically works for 3-tier Clos. Luopan is topology oblivious. Similar to Luopan, Hermes [38] samples network congestion with a small number of samples to reduce overhead. It re-routes traffic at sub-flow level instead of flowcells. Luopan provides a more comprehensive analysis of sampling based load balancing at flowcell granularity.

Another related work is Multipath TCP [19]. MPTCP creates multiple subflows for one TCP flow, and sends them over different paths. Each subflow reacts to congestion on its own path, shifting traffic from a congested path to a less congested one. MPTCP increases congestion at the edge links because multiple sub-flows cause more burstiness. This hurts the performance of mice flows which are more sensitive to latency and packet drops. It has been shown that mice flows perform worse in MPTCP than TCP [6].

Finally, Luopan is inspired by the power of two choices in randomized load balancing [26]. As explained in the Appendix, the difference is that Luopan adopts periodic sampling where the sampled information is reused for a small time period, whereas the classical model assumes ideal instantaneous sampling. Periodic sampling reduces overhead significantly, and we show that it is as effective as instantaneous sampling asymptotically.

7 Conclusion

In this paper, we presented Luopan, a sampling based load balancing protocol for large scale networks. Luopan samples only two random paths to efficiently balance the traffic with low implementation overhead. Using extensive simulations, we demonstrated the effectiveness of Luopan for large scale fat-tree networks with various settings. It greatly reduces the 99.9%ile FCT for mice flows and the average FCT for medium and elephant flows compared to Presto and other schemes. Luopan also performs consistently well in asymmetric topologies.

References

[1] Cavium xpliant. https://www.cavium.com/pdfFiles/CNX880XX_PB_Rev1.pdf.

[2] The world's fastest and most programmable networks. https://barefootnetworks.com/media/white_papers/Barefoot-Worlds-Fastest-Most-Programmable-Networks.pdf.

[3] A. Agache, R. Deaconescu, and C. Raiciu. Increasing Datacenter Network Utilisation with GRIN. In *Proc. USENIX NSDI*, 2015.

[4] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *Proc. ACM SIGCOMM*, 2008.

[5] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *Proc. USENIX NSDI*, 2010.

[6] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, and G. Varghese. CONGA: Distributed Congestion-Aware Load Balancing for Datacenters. In *Proc. ACM SIGCOMM*, 2014.

[7] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center TCP (DCTCP). In *Proc. ACM SIGCOMM*, 2010.

[8] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. M. B. Prabhakar, and S. Shenker. pFabric: Minimal near-optimal datacenter transport. In *Proc. ACM SIGCOMM*, 2013.

[9] A. Andreyev. Introducing data center fabric, the next-generation Facebook data center network. https://code.facebook.com/posts/360346274145943/introducing-data-center-fabric-the-next-generation-facebook-data-center-network/, November 2014.

[10] Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal. Balanced allocations. *SIAM Journal on Computing*, 29(1):180–200, 1999.

[11] T. Benson, A. Anand, A. Akella, and M. Zhang. Microte: Fine grained traffic engineering for data centers. In *Proc. ACM CoNEXT*, 2011.

[12] J. Cao, R. Xia, P. Yang, C. Guo, G. Lu, L. Yuan, Y. Zheng, H. Wu, Y. Xiong, and D. Maltz. Per-packet load-balanced, low-latency routing for Clos-based data center networks. In *Proc. ACM CoNEXT*, 2013.

[13] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. Devoflow: Scaling flow management for high-performance networks. In *Proc. ACM SIGCOMM*, 2011.

[14] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella. On the impact of packet spraying in data center networks. In *Proc. IEEE INFOCOM*, 2013.

[15] P. X. Gao, A. Narayan, G. Kumar, R. Agarwal, S. Ratnasamy, and S. Shenker. pHost: Distributed Near-optimal Datacenter Transport Over Commodity Network Fabric. In *Proc. ACM CoNEXT*, 2015.

[16] S. Ghorbani, Z. Yang, B. Godfrey, Y. Ganjali, and A. Firoozshahian. DRILL: Micro Load Balancing for Low-latency Data Center Networks. In *Proc. ACM SIGCOMM*, 2017.

[17] P. Gill, N. Jain, and N. Nagappan. Understanding network failures in data centers: Measurement, analysis, and implications. In *Proc. ACM SIGCOMM*, 2011.

[18] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, P. Patel, and S. Sengupta. VL2: A Scalable and Flexible Data Center Network. In *Proc. ACM SIGCOMM*, 2009.

[19] H. Han, S. Shakkottai, C. V. Hollot, R. Srikant, and D. Towsley. Multi-path tcp: A joint congestion control and routing scheme to exploit path diversity in the Internet. *IEEE/ACM Trans. Netw.*, 14(6):1260–1271, December 2006.

[20] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella. Presto: Edge-based Load Balancing for Fast Datacenter Networks. In *Proc. ACM SIGCOMM*, 2015.

[21] X. Jin, N. Farrington, and J. Rexford. Your Data Center Switch is Trying Too Hard. In *Proc. ACM SOSR*, 2016.

[22] S. Kandula, D. Katabi, S. Sinha, and A. Berger. Dynamic load balancing without packet reordering. *SIGCOMM Comput. Commun. Rev.*, 37(2):51–62, April 2007.

[23] N. Katta, A. Ghag, M. Hira, I. Keslassy, A. Bergman, C. Kim, and J. Rexford. Waze: Congestion-aware load balancing at the virtual edge for asymmetric topologies. 2017.

[24] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford. HULA: Scalable Load Balancing Using Programmable Data Planes. In *Proc. ACM SOSR*, 2016.

[25] S. Liu, H. Xu, and Z. Cai. Low latency datacenter networking: A short survey. http://arxiv.org/abs/1312.3455, 2014.

[26] M. Michael. The power of two choices in randomized load balancing. *IEEE Trans. Parallel Distrib. Syst.*, 12(10):1094–1104, 2001.

[27] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats. TIMELY: RTT-based Congestion Control for the Datacenter. In *Proc. ACM SIGCOMM*, 2015.

[28] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal. Fastpass: A Centralized "Zero-Queue" Datacenter Network. In *Proc. ACM SIGCOMM*, 2014.

[29] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren. Inside the Social Network's (Datacenter) Network. In *Proc. ACM SIGCOMM*, 2015.

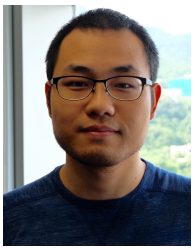
[30] S. Sen, D. Shue, S. Ihm, and M. J. Freedman. Scalable, optimal flow routing in datacenters via local link balancing. In *Proc. ACM CoNEXT*, 2013.

[31] M. Shafiee and J. Ghaderi. A simple congestion-aware algorithm for load balancing in datacenter networks. *IEEE/ACM Transactions on Networking*, 25(6):3670–3682, Dec 2017.

[32] N. K. Sharma, A. Kaufmann, T. Anderson, C. Kim, A. Krishnamurthy, J. Nelson, and S. Peter. Evaluating the power of flexible packet processing for network resource allocation. In *Proc. USENIX NSDI*, 2017.

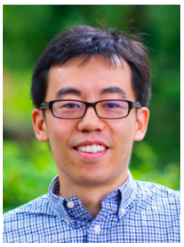
[33] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano, A. Kanagala, J. Provost, J. Simmons, E. Tanda, J. Wanderer, U. Hölzle, S. Stuart, and A. Vahdat. Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network. In *Proc. ACM SIGCOMM*, 2015.

[34] E. Vanini, R. Pan, M. Alizadeh, P. Taheri, and T. Edsall. Let it flow: Resilient asymmetric load balancing with flowlet switching. In *Proc. USENIX NSDI*, 2017.
[35] P. Wang, H. Xu, Z. Niu, D. Han, and Y. Xiong. Expeditus: Congestion-aware load balancing in clos data center networks. In *Proc. ACM SoCC*, 2016.
[36] L. Ying, R. Srikant, and X. Kang. The power of slightly more than one sample in randomized load balancing. In *Proc. IEEE INFOCOM*, 2015.
[37] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz. DeTail: Reducing the flow completion time tail in datacenter networks. In *Proc. ACM SIGCOMM*, 2012.
[38] H. Zhang, J. Zhang, W. Bai, K. Chen, and M. Chowdhury. Resilient datacenter load balancing in the wild. In *Proc. ACM SIGCOMM*, 2017.

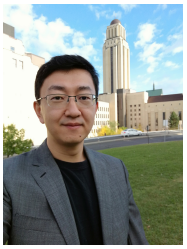


Peng Wang is currently a Ph.D. candidate in Department of Computer Science, City University of Hong Kong. He received the B.S. degree in information engineering from Xidian University in 2013. His research interests include data center networking and cloud computing. He received the best paper award from ACM CoNEXT Student Workshop 2014.

George Trimponias received his PhD from the Department of Computer Science and Engineering at the Hong Kong University of Science and Technology. Prior to that he obtained a five-year diploma in Electrical and Computer Engineering from the National Technical University of Athens, Greece. He is currently a researcher at Huawei Noah's Ark Lab in Hong Kong. His research interests include algorithmic design, combinatorial optimization, and game theory.



Hong Xu is an assistant professor in Department of Computer Science, City University of Hong Kong. He received his M.A.Sc. and Ph.D. degrees from the Department of Electrical and Computer Engineering, University of Toronto. His research interests include data center networking, NFV, and cloud computing. He was the recipient of an Early Career Scheme Grant from Hong Kong Research Grants Council in 2014. He also received the best paper awards from ACM TURC (Sigcomm China) 2017, IEEE ICNP 2015, and ACM CoNEXT Student Workshop 2014. He is a member of ACM and IEEE.



Yanhui Geng is currently the Director of Huawei Montreal Research Centre. Before that, he was a Senior Researcher and Project Manager at Huawei Noah's Ark Lab (Hong Kong). He received his B.Eng. and M.Eng. in Electronic Engineering and Information Science from the University of Science and Technology of China (USTC) in 2002 and 2005, respectively. He obtained his Ph.D. degree in Electrical and Electronic Engineering from the University of Hong Kong (HKU) in 2009. His research interests include artificial intelligence, machine learning, big data analytics, SDN (Software-Defined Networks), and data-center networking. He has filed 20+ patents globally and he has 26 technical publications on international journals and conferences. He received the IEEE ICC 2010 Best Paper Award.

APPENDIX

This section summarizes the theoretical underpinning of sampling based load balancing. Note that the purpose of our analysis is not to accurately model Luopan, which in itself is challenging. Instead, we aim to analyze the effect of periodic sampling and reusing the sampled information, which may be outdated at the time of decision-making, in the design of Luopan.

The classical randomized load balancing model [10], [26] assumes instantaneous sampling, where load balancing protocols should be designed to sample for each individual flowcell upon its arrival. Instantaneous sampling incurs large overhead in high-speed networks and extra latency waiting for the probe to come back. Thus Luopan adopts periodic sampling: flowcells arriving within the same period use the same information locally available at the ToR switch. We extend the original framework [10], [26] to consider the effect of sample reuse, where the sampled congestion is valid for a fixed number of flowcells; that is, the information may be stalled. We establish that the asymptotic behavior of periodic sampling with sample reuse is the same as instantaneous sampling, demonstrating the theoretical effectiveness of Luopan's design.

The "Balls into Bins" Model

Two well-studied paradigms for randomized load balancing are the continuous *supermarket model* [26], [36] and the discrete *balls into bins* model [10]. In this work, we adopt the latter because it simplifies our analysis while providing a good abstraction of randomized load balancing.

Under this model, we sequentially place m balls into $n \leq m$ bins. At each time step a ball is placed in the least full among d bins, chosen independently and uniformly at random. Load balancing studies the asymptotic behavior of the aforementioned stochastic process in the limit as $n \to \infty$, and, in particular, the number of balls (load) on the fullest bin.

One of the most fundamental results concerns a profound dichotomy between the cases $d = 1$, where the ball is placed in one randomly chosen bin, and $d \geq 2$, where at least two bins are sampled. In particular, when $d = 1$ the fullest bin has with high probability $(1 + o(1)) \log n / \log \log n$ balls in it. On the other hand, when $d \geq 2$, the fullest bin has with high probability $\log \log n / \log d + O(1)$ balls, an exponential improvement over $d = 1$ [10].

Note that in our scenario the balls correspond to the flowcells (of variable size) and the bins to the paths for each source-destination pair; the bin load thus corresponds to the path congestion. The above results naturally motivate a randomized load balancing scheme, where each time we sample a small number of d paths, and select the least congested one.

New Model with Sample Reuse

A practical challenge with the original mechanism is that we need to sample paths for each individual flowcell. This incurs large overhead in high-speed networks. For this reason, we propose an extension of the original framework for Luopan where the sampled congestion is valid for a time period τ; after τ time units have expired, the sample is no longer valid, so we need to resample the network.

Because this model is more challenging to analyze, we can consider an alternative scheme where the sampled paths are used for k consecutive flowcells instead of just one. Note that this is equivalent to the previous scheme as long as the flowcells arrive at a constant rate of 1 cell per τ/k time units.

Proposition 1. *For fixed k and $d \geq 2$, the new scheme has the same asymptotic complexity as the original without sample reuse.*

Proof. Azar et al. [10] prove that the fullest bin has with high probability $\Theta(m/n) + (1 + o(1)) \log \log n / \log d$ balls, when $m \geq n$. Now, assume $m = k \cdot m_0$ balls. The idea is to consider each k consecutive balls using the same samples as one large ball, and analyze the system in terms of these $m_0 = m/k$ large balls. Indeed, the fullest bin has $\Theta(m_0/n) + (1 + o(1)) \log \log n / \log d$ large balls. Since a large ball consists of k smaller ones, this implies that the fullest bin contains $k \cdot (\Theta(m_0/n) + (1 + o(1)) \log \log n / \log d) = \Theta(m/n) + (1 + o(1)) \cdot k \cdot \log \log n / \log d$. For fixed k, the asymptotic complexity of sample reuse is trivially the same as that of the original mechanism. □

On the other hand, note that when $k = \omega(\log n)$, $d = 1$ asymptotically performs better. Thus, as long as k is sufficiently small, performance is better than ECMP; but as soon as k increases beyond a threshold, a large number of flowcells are routed through the same links, so performance may become even worse than $d = 1$ (§4.3).