Co-Designing Traffic Control with NVMe-oF for Disaggregated Storage: A Comparative Study of Switched and Switchless SAN Architectures

Chendong Wang, Joontaek Oh, and Ming Liu

University of Wisconsin-Madison

Abstract

Disaggregated storage is a pivotal component of today's cluster infrastructures. With the advent of high-bandwidth server interconnects and new NVMe form factors, commodity storage appliances are becoming denser, delivering tens of millions of IOPS. This calls for today's storage area network (SAN) fabric to expand the bandwidth capacity drastically. Industry practices tackle this issue via either (i) a scale-up approach, upgrading the per-port bandwidth in a switched SAN, or (ii) a scale-out strategy, integrating more paths in a switchless SAN. However, it is unclear which network architecture is more suitable for scaling storage disaggregation.

This paper presents a comparative study of switched and switchless SAN architectures from several angles. We begin by developing an experimental methodology that integrates both small-scale real-system prototypes and large-scale simulations, providing the flexibility needed to explore architectural trade-offs. We then characterize NVMe-oF I/O flows and co-design SAN traffic control mechanisms around these characteristics to improve I/O transmission efficiency in both settings. Our evaluation yields several key findings. First, the switchless SAN achieves throughput comparable to that of the switched SAN, despite involving additional routing hops, while simultaneously reducing latency through the use of multiple load-aware I/O paths that mitigate interference. Second, the switchless SAN reduces capital costs by obviating the need for expensive high-radix switches, scales effectively under heterogeneous I/O workloads, and avoids the single point of failure associated with top-of-rack (ToR) switches. Collectively, these results demonstrate that switchless SANs provide a compelling alternative to traditional switched designs for disaggregated storage environments.

1 Introduction

Storage disaggregation is becoming widely deployed in today's public clouds, enterprise on-premise clusters, and edge data centers [18, 20, 52, 61, 100, 119]. Disaggregation allows independent scaling of compute and storage resources, improving hardware utilization and reducing the infrastructure TCO (total cost of ownership). Driven by the availability of high-speed network fabric (e.g., 400+ Gbps) and fast remote storage protocol (like NVMe-over-Fabric [36]), a disaggregated NVMe SSD could provide tens of microseconds latencies and millions of IOPS, approaching the performance of direct-attached storage but without any limitations on capacity extension.

Lately, storage servers have become much denser than before. This is due to two architectural trends. First, the compounding effect of continuously improving PCIe interconnect [37], increasing PCIe lanes per root complex [38], and newly-induced compute express link (CXL) [12] offers hundreds of gigabytes per second I/O throughput. Second, the emerging Enterprise and Data Center Standard Form Factor (EDSFF) [15] for NVMe SSDs is physically more compact and power/thermal efficient than traditional 2.5" and M.2 ones, yielding high NVMe drive consolidation. As a result, when deploying such dense storage nodes under disaggregation, there is a pressing need to expand the storage area network (SAN) bandwidth capacity to fully unleash their I/O capabilities.

People have developed two general ways to tackle this issue. One is to use beefy high-radix SAN switches [9, 10, 26] and upgrade per-port bandwidth to satisfy the I/O demand. The other one is to apply a switchless architecture and equip more I/O paths at the storage node via a specialized low-radix adapter [23, 24, 66]. The former takes a scale-up strategy: straightforward, high-performance, and easy to deploy. However, it is prohibitively expensive, and its scale is hindered by switching technology. The latter applies a scale-out philosophy, which is cheap and adaptive, but is performance-suboptimal (due to multi-hop routing) and non-transparent to the upper storage stack, requiring non-trivial modifications to integrate the multiple paths. Researchers have explored switched and switchless network design for HPC clusters and data centers [53, 59, 64, 70, 72, 73, 86, 114, 116]. However, it is unclear which network design would be more suitable for storage disaggregation, especially given that its traffic pattern is somewhat regular with a few characteristics (§4.1).

In this paper, we conduct a quantitative comparative study

of switched and switchless SAN architectures across multiple dimensions, including performance, cost, scalability, and reliability. Our results demonstrate that the proposed techniques provide clear advantages, thereby answering the above question in the affirmative.

First, we design an experimental methodology (§3) consisting of small-scale real-system prototypes and large-scale simulations. In particular, the switched SAN testbed uses commodity high-radix storage switches and adapters, whereas the switchless one is built atop SmartNIC-based low-radix adapters. Second, we co-design the SAN traffic control with NVMe-oF and integrate it into both networking architectures to optimize storage flow transmission performance and efficiency. Essentially, three techniques are proposed and developed: (i) INT-assisted symmetric routing (§4.3), which constructs in-network telemetry at each switching hop that captures the congestion probability of different forward paths to remote SSDs and then chooses the least congested route; (ii) eager bandwidth reservation (§4.4), performing proactive bandwidth allocation based on the pairwise and asymmetric characteristic of an NVMe-oF command pair; (iii) storage-driven flow scheduling (§4.5), where we employ the end-to-end design principle, disseminate storage loading status to the network, and use it to instruct the packet scheduling.

Third, we run synthetic workloads and real applications over switched and switchless SAN in emulation and simulation (§5). Our learnings are summarized as follows:

- *Performance.* Both networking architectures can achieve comparable I/O throughput. Additional routing hops jeopardize performance a little since SSD access dominates. However, the switchless SAN achieves lower average/tail latency (up to 43.1%) because its multi-path capability, coupled with the end-to-end I/O load awareness, enables better load balancing of reads/writes across different paths, reducing I/O contention and interference.
- *Cost.* A switchless SAN replaces prohibitively expensive high-radix switches with cheap low-radix adapters, drastically reducing the capital costs of networking infrastructures and yielding a lower per-port deployment expense. For example, our in-house prototype sees 50.5% cost savings. We believe this trend will continue due to the increasing complexity of designing future tens (or even hundreds) of terabit high-radix switching chips [8, 33].
- *Scalability.* We find that both switched and switchless SANs scale well with the number of clients and storage flows. With our proposed traffic control, the switchless architecture is more adept at handling traffic heterogeneity due to its inherent path diversity.
- *Reliability.* The switchless SAN can tolerate single-point failure of the ToR switch as in the switched case, because it introduces multiple paths for I/O delivery. When failures happen, all ongoing requests are affected and should be rerouted when the network becomes stable, causing a temporary throughput drop (up to 6.8% in our evaluation);

Limitation. Our study sheds some light on comparing the switched and switchless SAN under bandwidth expansion. Many other important factors are not discussed, e.g., management complexity, networking monitoring, and fault localization at scale. We make some initial steps toward rethinking network architecture for future storage disaggregation.

2 Background and Motivation

This section provides the necessary background about SAN and NVMe-oF, and motivates our comparative study.

2.1 Storage Area Network and NVMe-over-Fabric

A storage area network (SAN) is a dedicated, specialized, and fast networking fabric. It decouples computing servers and storage devices, and allows independent resource scaling. SANs have gone through several evolutions in the past decades, driven by both advances in storage mediums and network fabrics and the proliferation of application demands.

A typical SAN deployment consists of three hardware pieces: (1) host bus adapter (HBA), a PCIe device that runs the hardware-offloaded remote storage protocol and provides storage access to the SAN; (2) interconnect fabrics, including switches, routers, and protocol bridges, together forwarding I/O requests; (3) storage adapter and storage appliances (comprising storage media, and sometimes organized as a RAID subsystem [105]). Fibre Channel (FC) [16] has been the primary and established interconnect technology for building SANs. Unlike the network-attached storage (NAS) [67], SANs expose the block abstraction, where storage applications build volume stores, file systems, and databases atop.

NVMe-over-Fabrics. Modern SANs use NVMe-over-Fabric (NVMe-oF) [36] as the remote storage protocol. The base NVMe specification defines the architecture, command sets, and queueing interface for local NVMe drives. NVMe-oF extends it by (1) adding the data transport capability (via RDMA, TCP, or FC) to carry command and response capsules; (2) exposing the multi-paired queue of an NVMe and its controller through memory-mapped I/O registers; (3) providing the discovery and connection establishment mechanisms for NVMe subsystem setup; (4) supporting the scatter-gather list (SGLs) for large in-capsule data transfers. An NVMe-oF client initiator first attaches to an NVMe-oF target server and then builds a one-to-one mapping between I/O submission and completion queues. To fully use the I/O parallelism, NVMe queues are usually aligned with CPU cores and paired with the transport queueing model. Next, when transmitting a command, the initiator driver encapsulates the NVMe command into a fabric-agnostic capsule and passes it to the transport layer. The capsule traverses the network fabric and is delivered to a target NVMe subsystem for I/O execution. Upon the NVMe I/O completion, the target-side driver fabricates the result into a response capsule and returns it to the client.

Take the NVMe-over-TCP protocol as an example. Com-

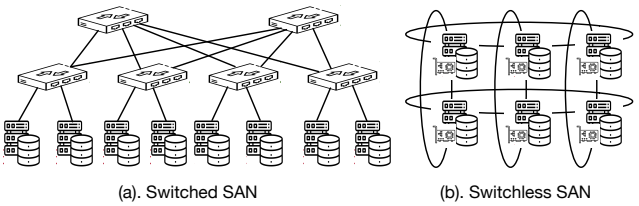


Figure 1: The switched and switchless SAN architecture.

munications between a client (compute node) and a controller within an NVM subsystem are facilitated through Protocol Data Units (NVMe/TCP PDUs). These PDUs, encapsulated into TCP packets, contain data or control status information. The communication process is initiated when an NVMe host (initiator) establishes a TCP connection to an I/O queue pair to submit NVMe commands. Upon establishment, the initiator can submit I/O requests. For a read I/O, the initiator sends the read command to the target through the associated TCP connection. The target then retrieves the requested data and sends it back to the initiator. Regarding writes, the initiator first sends a write command, then waits for the Ready-to-Transfer signal from the controller. Upon reception, it sends the data to be written. The target returns an acknowledgment to the initiator after writing the data to the storage.

2.2 SAN Needs More Networking Bandwidth

Hardware Trends. More recently, storage appliances have become much denser, with a significant increase in storage capacity and I/O bandwidth. One can easily configure a storage node with dozens of NVMe drives [25, 42, 45], offering tens to hundreds of terabytes of raw flash capacity and millions of IOPS throughput. Such considerable I/O capability improvements are mainly driven by two architectural trends.

- **#1:** The PCIe root complex is evolving and can hold many PCIe lanes per CPU socket. For example, a Dell R7725 box with two AMD EPYC 9005 series processors supports 256 PCIe Gen5 lanes, delivering up to 1.0 TB/s of theoretical bandwidth. With PCIe Gen6/7 [37] and upcoming server processors [4, 47], newly-induced storage nodes will be equipped with even higher PCIe bandwidth;
- **#2:** Enterprise and Data Center Standard Form Factor (EDSFF) [15] is being widely deployed. Compared with existing 2.5" and M.2 form factors, it is physically compact and provides superior power/thermal efficiency, manageability, and serviceability. An EDSFF E1 mechanical study [14] shows that a conventional 1U server enclosing 10× 15mm 2.5" NVMe drives is now able to hold 32× 9.5mm E1.S NVMe SSDs, yielding 3× more density.

The Problem. Given such hardware trends, when deploying these storage nodes remotely, to fully unleash their I/O capacity, one needs to increase the per-node networking bandwidth drastically. We have already seen some early industry products. For example, a Microsoft/Fungible FS1600 node [2] pairs 12× 100Gb/s Ethernet ports with 24× NVMe SSDs and delivers up to 15M random 4KB IOPS. Such rising networking demands require the NVMe-oF SAN switching fabric to extend the bandwidth capacity considerably. Today's industry practices tackle this issue in two ways:

- One is to take a *switched* networking architecture and upgrade per-port bandwidth (Figure 1-a). This is a scale-up and easy-to-deploy approach, but it requires replacing all networking gear (including cables, adapters, and switches) along the I/O path, which is prohibitively expensive. For example, a high-radix 400/800GbE switch (like Arista 7280R3 [5] and Cisco Nexus 9300 [10]) costs much more than the same type with 25/40/100GbE;
- The other is to employ a *switchless* design which equips more I/O paths with the storage node (Figure 1-b). This scale-out strategy is more cost-effective but is non-transparent to the storage stack. For example, the multi-hop forwarding, path selection, and congestion control add system design complexities and would incur performance overheads. People usually build such SAN architecture using a low-radix storage adapter or endpoint-switching module, which is available in the market [23, 24, 66].

However, it is unclear which networking architecture is more suitable for storage disaggregation when accommodating future dense storage appliances. This paper sheds some light on it through a comparative study.

3 Experimental Methodology

This section describes our hardware testbed and simulation setups for rack-scale and cluster-scale evaluation.

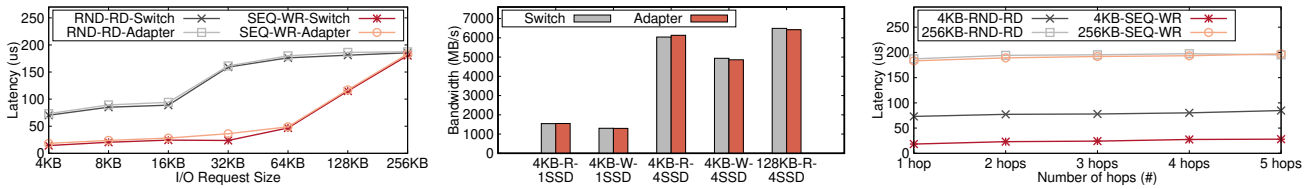
3.1 Switched SAN

Our rack-scale testbed comprises (1) a 32-port 100GbE Dell Z9100-ON SAN switch; (2) a 32-port 100GbE Netberg Aurora 710 Tofino P4 switch; (3) x86 storage servers, which have two Intel Xeon processors, 256GB DDR4 memory, dual-port 100GbE Nvidia/Mellanox ConnectX-6 NICs, an Intel D3-S4610 960GB SSD, and Samsung PM9A3 960GB NVMe SSDs; (4) x86 client servers that have similar configurations as storage nodes without NVMe drives, which we use to issue I/O traffic. When scaling up bandwidth, we use NIC teaming to combine several ports and create a link aggregation group on the switch [32]. All the servers run Ubuntu 22.04.

3.2 Switchless SAN

The switchless architecture requires low-radix switching at the storage node. We develop such a storage adapter using Nvidia/Mellanox BlueField-2 SmartNICs. This brings us two benefits compared with using a commodity one. One is that it has data-plane programmability that allows us to explore different traffic control strategies (§4); the other is that it has the same NIC substrate as the CX6 card used in switched SAN, eliminating the NIC-induced performance anomaly.

Note that SmartNIC-based storage adapters have emerged and are becoming increasingly attractive. Take the Fungible



(a) Latency varying I/O block size. **(b) Bandwidth varying I/O profiles.** **(c) Latency varying the number of hops.**

Figure 2: We compare the performance between a high-radix SAN switch and a low-radix SAN adapter. (a) compares the latency when varying I/O block sizes. (b) shows the bandwidth of 4KB random read and 128KB sequential write for one and four SSDs. (c) reports the latency when varying the number of hops in the switchless SAN. RND-RD=Random Read. SEQ-WR=Sequential Write.

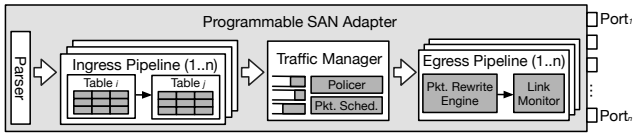


Figure 3: The architecture of a low-radix storage adapter.

FAC200 [1] as an example. It not only offloads the NVMe-oF protocol stack and provides fast NVMe accesses, but also enables in-line stateful per-IO computations using embedded execution engines (e.g., flow processors and domain-specific accelerators) and memory resources (i.e., scratchpad, SRAM, DRAM). Similar designs also appear in the Falcon transport [115]. Thus, our emulation strategy is a valid design.

Low-radix Storage Adapter. Its internal switching architecture (Figure 3) has three components: (1) a parser/deparser for identifying NVMe-oF commands and performing packet encapsulation/decapsulation; (2) ingress/egress pipelines, which look up the forwarding table and modify packets; (3) a traffic manager, responsible for traffic control, maintaining flow telemetry, and making scheduling decisions. We prototype the adapter as a software SAN switch [7,30] over DPDK [13]. Our implementation is developed atop the QoS framework [44] over multiple worker threads. When emulating 3+ ports, we combine several BlueField-2 (BF2) cards, where the switching logic spreads between the SmartNIC and a server. For example, routing from port-1 of BF2-A to port-2 of BF2-B requires traversing across PCIe. This inevitably incurs some latency penalties. We quantify them below and show that the overheads are acceptable for I/O requests. Commodity ASIC-based multi-port networking adapters are available, such as Intel E810-XXVDA4 [29] with four 25GbE SFP28 ports. Building a high-bandwidth programmable one is feasible, but it would bring implementation challenges, such as parallel high-speed SerDes lane management, internal packet switching, PCB routing, and on-board buffering.

Network Topology. We then use the low-radix adapter to construct direct-connect networks [64, 86]. Its topology depends on the node degree (i.e., the number of adapter ports). For example, a 1-port adapter only supports a daisy chain. With 2 or 3 ports, one can build a ring or dual-ring topology. If the adapter has 4+ ports, we can develop a more complex network (e.g., dragonfly and 3D torus) with richer routes.

We use the canonical distance-vector routing mechanism [78] to populate the forwarding table for the adapter, where the distance is defined as the number of hops. The table matches an NVMe-oF transport identifier, including NVMe Qualified Name (NQN), transport address, and service ID, and is populated during the NVMe-oF connection setup phase. We maintain the default (preferable or shortest) route, as well as the Top-K candidates for each destination (optional), which is used for our path adaptation (§4.3). Given an N-node cluster, the table size will be up to $K \times N$, where one could bound K to limit the table size and lookup performance.

Characterization. We examined the adapter's performance using FIO [17] under an SPDK NVMe-over-TCP setup [41], and compared it with the commodity SAN switch (§3.1).

- **Latency and bandwidth:** We measured the unloaded latency, QD (queue depth)=1, varying the I/O block size. As shown in Figure 2-a, the adapter only incurs a marginal latency increase because NVMe SSD accesses dominate an NVMe-oF I/O. For example, on average across all cases, the random read experiences 3.6μs more latencies, only 3.2% of the I/O execution. Regarding bandwidth, our adapter achieves similar IOPS as the switch for both 1 and 4 SSD scenarios (Figure 2-b), indicating that it would not become a bottleneck and throttle the I/O transmission.
- **Hop #:** We configured a daisy chain topology and measured the I/O latency (QD=1) as increasing the number of hops (Figure 2-c). We find that traversing one hop only adds 2.9μs and 3.1μs for a 4KB and 256KB read, accounting for 3.4% 1.7% of its end-to-end I/O latency, respectively. This indicates that a multi-hop forwarding path in a switchless SAN is valid and would not significantly jeopardize the disaggregated storage performance (§2.2).

3.3 Storage Protocol and Workloads

Our prototype uses the NVMe-over-TCP protocol and exposes NVMe subsystems to clients. After mounting the drive, applications can issue arbitrary read/write requests without explicit application modification, where our switched and switchless SAN would route I/O submission and completion requests. We use both FIO-based synthetic workloads and real-world applications (e.g., RocksDB [39]) for evaluation.

3.4 Simulation Setup

We conduct scaled comparisons using simulation. We choose a trace-driven simulator (i.e., HTsim [27]) and enhance it

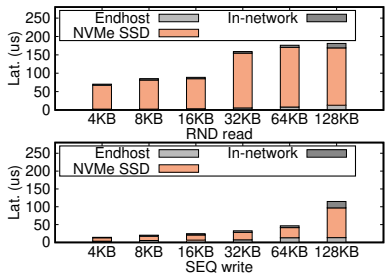


Figure 4: Read/write I/O latency breakdown varying the I/O block size (QD=1).

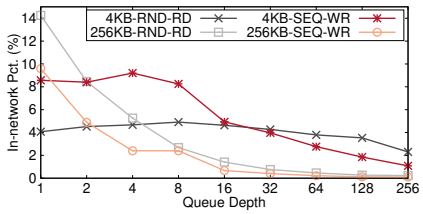


Figure 5: The percentage of in-network delay of the total request latency varying the number of outstanding I/Os.

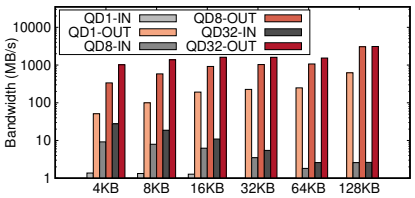


Figure 6: NVMe-oF random read ingress/egress traffic volume comparison varying the I/O block size. y is logscale.

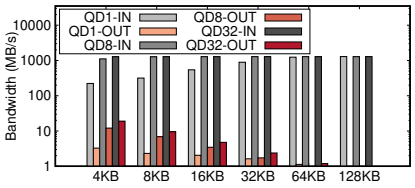


Figure 7: NVMe-oF sequential write ingress/egress traffic volume comparison varying the I/O block size. y is logscale.

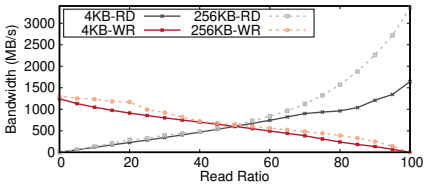


Figure 8: Bandwidth consumption of reads/writes varying the read ratio for two I/O size cases on a clean SSD.

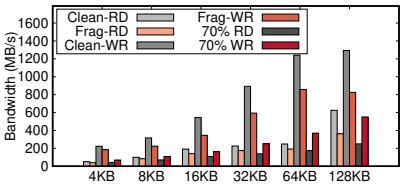


Figure 9: Bandwidth consumption of reads/writes varying the I/O block size on a fragmented SSD. QD=1.

with three parts: (a) the NVMe-oF storage protocol with the read/write interface; (b) a simple SSD model (based on Amber [69]) that emulates read/write latencies for different I/O sizes and queue depths; (c) a SAN adapter with reconfigurable switching capabilities. To validate the fidelity of our simulator, we compare the end-to-end NVMe-oF read/write latencies produced under varying queue depths against corresponding measurements obtained from our hardware testbed. Because the SSD model is externally integrated, we calibrate the simulation cycle counts to ensure that read and write completions are issued at the correct times. The simulation testbed is flexible and supports different topologies at scale. We use three SNIA block traces [40], i.e., Systor, OLTP, and WebSearch.

4 Co-Design Traffic Control with NVMe-oF

NVMe-oF-based disaggregated storage embodies traffic uniqueness, imposing new flow orchestration opportunities. This section presents our characterization, formalizes the traffic control (TC) problem, and describes how we co-design TC with the switched/switchless SAN.

4.1 Understanding NVMe-oF Characteristics

We use FIO-based synthetic workloads as the microbenchmark and analyze NVMe-oF traffic from three perspectives: per-IO, per-command pair, and per-storage session.

#1. Per-IO: Network RTT $\ll$ Storage RTT. We break down the latency of NVMe-oF read/write into three components, i.e., NVMe-oF protocol processing at the endhost, I/O serving from the NVMe drive, and I/O transmission over the network, and vary the I/O block size as well as the number of outstanding I/Os (or queue depth). When the queue depth (QD) is 1, as increasing the I/O block size (Figure 4), on average across all cases, the in-network delay occupies 4.8% and 12.7% of the total execution for a random read and a sequential write, respectively. When increasing the queue depth (Figure 5), the in-network percentage drops drastically. For example, a 4KB read/write I/O only spends 3.8% and 2.8% execution time in the network when the QD is 64, while the 256KB one just spends 0.5% and 0.2%, respectively. Therefore, the in-network delay contributes only a little to the NVMe-oF read/write processing, where the majority of execution time is spent in NVMe SSD accesses.

Design Implication. Adding some network latencies will not adversely impact the NVMe-oF SAN performance. Trading certain in-network delays to streamline I/O execution at the storage node would be a valid design choice.

#2. Per-Command Pair: Pairwise and Asymmetry. NVMe-oF requests and responses are pairwise and present size asymmetry along the sending and receiving paths. An NVMe-oF read issues a small command capsule, including a 64B submission queue entry along with optional data or scatter-gather lists. It then receives a large data block transfer back as well as a response capsule that contains a 16B completion queue entry. An NVMe-oF write operates in a reverse way. We characterize the ingress/egress traffic volume at a storage target for 1/8/32 QD cases. As shown in Figure 6, the incoming traffic of reads (gray bars) is significantly lower than the outgoing one (red bars). For example, when the read block size is 4/32/128KB, on average across three QD cases, the egress traffic rate is 37.0×, 296.7×, 1187.3× higher than the ingress one. For writes (Figure 7), contrarily, the incoming traffic volume outperforms the outgoing one by 75.8×, 609.7×, 2440.7× for the above three scenarios, respectively.

Design Implication. The SAN traffic volume is asymmetric across the sending and receiving paths. The BW requirement hinges on the I/O read/write ratio, indicating that full bisection bandwidth may not be necessary. Such traffic predictability can help with bandwidth reservation and traffic planning.

#3. Per-Storage Session: Backward-Propagated Queueing. The bandwidth capacity of an NVMe-oF session depends on the minimum communication velocity of the in-network transmission pipe and the in-storage serving pipe. Today's SAN fabric offers ample networking bandwidth but ignores the storage target part. When the target NVMe drive is overloaded, its I/O request queue shall be built up, further stalling incoming session traffic. Worse yet, the I/O throughput of an NVMe SSD varies unpredictably based on workload characteristics (such as I/O size and access pattern) and SSD conditions (e.g., fragmented level and internal garbage collection) [48, 56, 101, 109]. For example, a 4KB read-only stream achieves 356.0MB/s higher network bandwidth than the 4KB 50/50 read/write mixed one (Figure 8). A fragmented SSD (pre-conditioned with 128KB random writes) sustains 25.9% and 31.9% less read and write throughput than a clean SSD, when running the same workload (Figure 9).

Design Implication. Simply provisioning the SAN fabric bandwidth is inadequate to ensure the desired NVMe-oF stream performance. One should continuously monitor the target storage drive condition, propagate its service availability into the network fabric, employ an end-to-end flow control, and make the client-side adapt to such varying pipe velocity.

4.2 Problem Formalization

Similar to traditional network flow orchestration, traffic control of disaggregated storage can be viewed as a multi-commodity flow problem [51, 63, 104]. The issue fundamentally boils down to three questions: (a) which transmission path to take (*path selection*); (b) at which rate to send data (*congestion control*); and (c) what queueing discipline to follow at each switching point (*scheduling*). We first describe a generic algorithm and then discuss how to encode NVMe-oF characteristics (§ 4.1) for optimization.

A Generic Solution. We represent the disaggregated storage cluster as a graph $G(V,E)$, where (1) a vertex $v \in V$ is a SAN switch, an adapter (in the switchless case), or a storage node; and (2) an edge $e \in E$ is a link connecting two vertices. An NVMe-oF request (response) i can be defined as a flow f_i with three valuables–(d_i, r_i, p_i), where (1) d_i and r_i are the bandwidth demand and received (or allocated) bandwidth, respectively; (2) p_i is the chosen path, represented as a list of vertices $[v_{src}, ..., v_j, ..., v_{dst}]$. Note that (1) the bandwidth of an edge e is limited by the link capacity; (2) d_i is bounded by the maximum bandwidth of the target NVMe drive.

Thus, we formalize the problem as follows. In a given disaggregated SAN setup $G(V,E)$, for an incoming I/O flow f_k with stipulated bandwidth demand d_k and source/destination vertices (i.e., v_{src} and v_{dst}), the traffic control aims to (a) decide the transmission path (p_k) of f_k; (b) compute a new bandwidth allocation strategy of all existing flow $[r_1, r_2, ..., r_k]$ such that the aggregated performance degradation ($Perf_d$) of all storage flows is minimal. Suppose r_i' is the newly assigned bandwidth for a flow i after f_k joins, following the MCFP

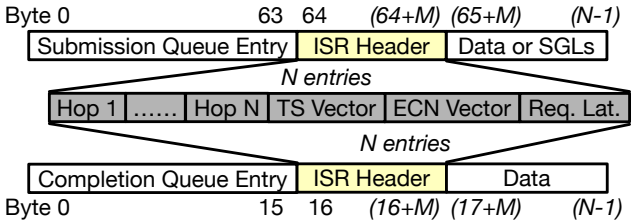


Figure 10: ISR header format, posited after the NVMe-oF header. M depends on the number of hops.

(minimum-cost flow problem) [62, 68, 76, 87], we can calculate $Perf_d$ as $\frac{1}{\sum_{i \in F_p} 1 + 1}(\sum_{i \in F_p} \frac{d_i}{r_i'} + \frac{d_k}{r_k})$. One can use the Lagrange Multiplier [31] to solve this optimization problem. For example, to find the minimal $Perf_d$, a naive strategy is to enumerate all possible paths with bandwidth allocations and choose the one ($<r_k, p_k>$) yielding the least degradation.

NVMe-oF Enabled Co-Design. The above approach is infeasible in practice because it makes several unrealistic assumptions: maintaining a global and timely view of the entire network, relying on heavy computations at each switching hop to make TC decisions, and ignoring I/O latency requirements. However, NVMe-oF flow characteristics bring us several specialization opportunities to streamline the algorithm:

- *In-network coordinated I/O routing.* Given that the network delay only contributes little to the end-to-end latency (*Observation #1* in §4.1), one can develop a distributed approach that enables each switching hop to make a greedy routing decision based on its local flow telemetry to approach the global optimal solution gradually.
- *Informed path selection.* Enumerating all available paths is not necessary because of the pairwise and asymmetric characteristics of NVMe-oF flows (*Observation #2* in §4.1). One can prune path candidates significantly and make early traffic planning when receiving the NVMe-oF requests.
- *Proactive rate estimation.* Bandwidth allocation not only depends on the network, but also storage (*Observation #3* in §4.1). One can monitor the available I/O throughput of the target SSD, disseminate this statistic through end-to-end flow control, and collaborate with the NVMe-oF's transport layer to determine the flow rate.

4.3 INT-assisted Symmetric Routing

SAN delivers I/Os between clients and target SSDs. The goal is to select a capable path that ensures fast transmission and avoids potential storage-induced congestion, regardless of switchless or switched architecture. For example, A heavily loaded SSD could stall corresponding NVMe-oF flows, whose impact would be back-propagated along the forwarding path. As a result, I/O requests that share the impaired path experience head-of-line (HoL) blocking.

We propose an **I**NT (in-network telemetry)-assisted **S**ymmetric **R**outing mechanism (dubbed as ISR). The idea is to construct an in-network telemetry of the congestion propen-

sity of all paths to remote SSDs at each switching hop and then choose the least congested one. To realize this, ISR continuously probes the per-hop congestion along the submission path and leverages the NVMe-oF's pairwise feature to piggyback probing statistics via the completion I/O. Figure 10 depicts our customized header. It stays after the NVMe-oF header, including the ID of each traversed hop, a timestamp vector, an ECN vector, and SSD access latency. We use ECN bits for congestion detection for both switched and switchless cases. Our telemetry construction protocol works as follows:

- **I/O Submission Path:** Each NVMe-oF request packet chooses the next hop with the least congestion estimation, records the switch ID into the ISR header, and attaches a timestamp (for delay measurements). It then queries the switch egress queue and sets the corresponding ECN bit when the queue occupancy exceeds the marker threshold.
- **Storage Node:** When building the completion response, the NVMe-oF target copies the ISR header from the submission packet and fills in the SSD access latency. Thus, the completion word could piggyback both the routing and on-path congestion information.
- **I/O Completion Path:** An NVMe-oF response decides the next hop by querying the ISR header directly and then updates the local telemetry table.

Note that routing symmetry fundamentally allows each switch to learn the congestion probability to a remote SSD when taking different paths. However, as discussed above, such a symmetric requirement only applies to each command pair, not the entire NVMe-oF session, indicating that different I/Os could switch paths when necessary. Further, the congestion degree of *path i* is determined by: (a) delay, the round-trip time (RTT) between the current switching hop and the target drive; (b) ECN bit vector, reporting the queueing occupancy of each traversed hop. We use two metrics to distinguish the congestion within the network or at the target storage. For example, a large delay with a sparse vector mostly indicates that the SSD is overloaded. Based on this, we design the routing protocol as follows (Algorithm 1):

- **Path Selection.** The switch walks through available paths and picks up the one with minimal in-network delay. When there is a tie, the algorithm chooses a shorter path in terms of the number of intermediate hops. There is no routing loop because the ISR header records all traversed hops, and it always chooses a distinct one as the next.
- **Path Adaption.** When receiving an I/O completion, the switch performs two tasks. First, based on the ECN bit vector, it calculates the congestion level. Instead of considering the next hop, we capture all intermediate hops between the current switch and the destination to gain an end-to-end insight. Second, the switch updates the delay table of the corresponding congestion level using current/prior encoded timestamps and the request latency, smoothed by the exponential weighted moving average (EWMA).

Algorithm 1 Path Selection and Adaption

```
1:  cur/dst: the current/destination hop ID
2:  iport: the ingress port
3:  procedure PATH_SELECTION()
4:      net_delay ← delay_MAX; dist ← dist_MAX
5:      for eport in all ports do
6:          if eport == iport then Continue
7:          tdist ← distance_tbl[dst][eport]
8:          tdelay ← net_delay_tbl[dst][eport]
9:          if tdelay ≤ net_delay & tdist < dist then
10:             next_hop ← eport; net_delay ← tdelay; dist ← tdist
11:     return next_hop
1:  procedure PATH_ADAPT()
2:      cong_level ← 0
3:      for i in hops[cur,dst] do
4:          if ECN_vector[hops[i]] is set then clevel++;
5:      qdelay ← cur_timestamp - TS_vector[cur] - req_lat
6:      apply_EWMA(delay_cong_tbl[iport][clevel], qdelay)
7:      net_delay_tbl[dst][iport] = delay_cong_tbl[iport][clevel]
8:      return
```

SAN Implementation. ISR, similar to source routing [83, 85, 118], can be realized in both switched and switchless architectures. Maintaining in-network telemetry requires some programmability from the data plane. In a switched SAN, multiple paths are offered by different ToRs, and thereby, we implement ISR at the client host, as it is the vantage point to make the routing decision. In a switchless SAN, ISR is directly built into the low-radix storage adapter. With the advent of ultra–low-latency SSDs [46], minimizing in-network delay has become increasingly critical. We believe that ISR remains valuable in both switched and switchless cases, as its path selection mechanism can identify and utilize the route with the lowest available network latency. Moreover, routing paths can be categorized according to the type of destination NVMe device, enabling differentiated path adaptation strategies that account for the tradeoff between latency and bandwidth.

4.4 Eager Bandwidth Reservation

Next, we tackle the bandwidth allocation issue. As discussed above, NVMe-oF storage flows are a little more predictable (e.g., pairwise) than data center traffic. We therefore take a proactive strategy instead of a reactive one. Specifically, when observing an NVMe-oF submission request from $Port_{in}$, one could pre-allocate an estimated bandwidth (BW_{est}) based on the I/O size and type and install it in the traffic policer of $Port_{in}$ egress queue for later NVMe-oF completed I/Os.

This can be achieved as follows. First, we obtain the bandwidth requirement (BW_{est}) of different I/O profiles via offline profiling as what today's cloud storage does [6, 11, 19]. Note that this reserved bandwidth only sets up the upper bound because it ignores many factors, such as the SSD drive condition. Second, we associate a traffic policer (through a token bucket mechanism) with a refilling rate of BW_{est} to ensure

that an I/O stream cannot exceed its quota. Third, on the I/O data path, we differentiate request processing according to their characteristics, i.e., assigning NVMe-oF read and write I/O completion requests to different priority queues because a read fetches a large chunk of data and a write generates a completion acknowledgment packet (less than a hundred bytes). This mitigates the I/O interference from mixed streams. Our reservation scheme operates at the NVMe-oF session granularity, so the installed rate is updated frequently when an NVMe-oF request(response) comes(leaves). When the aggregated bandwidth reservation of a switching port exceeds the limit, the switch rejects subsequent I/Os.

Over-commitment. Since bandwidth reservation happens eagerly and reserved completion I/Os would arrive in the future, a strict traffic policer would experience low bandwidth utilization, especially when the available bandwidth has been reserved but completion requests are not returning yet. We address this with an over-commitment mechanism, where a factor R_{over} (≤ 1) is applied when calculating the reserved rate. Such bandwidth deflation temporarily boosts the number of I/Os. R_{over} is a step function, varying with the bandwidth headroom, where we obtained it empirically.

SAN Implementation. We rely on the rate limiter to realize the reservation mechanism. Our switched prototype associates traffic classes with different NVMe-oF sessions and updates the reserved bandwidth through the control plane. However, the reservation requests are issued by clients when issuing I/Os. The switchless implementation uses a flexible software rate limiter from the DPDK QoS framework [44].

4.5 Storage-driven Traffic Scheduling

We apply the end-to-end principle and enhance the packet scheduler with the target drive capability. Specifically, the storage node actively monitors the bandwidth headroom of its housed SSDs, encodes this information into the NVMe-oF completion word, and carries it back to the client. All traversed switches then record these statistics and use them for scheduling. Such an end-to-end scheme eschews an NVMe-oF client overwhelming the remote SSD from both the networking and storage perspectives.

Load Estimation. Prior studies [71, 101] have exploited the use of I/O delay to monitor the SSD congestion and infer the bandwidth headroom. We use a similar approach. Our scheme measures the I/O serving latency of each I/O, takes the latency difference between measured and target levels as a congestion signal, and then adjusts the submission rate. Akin to Gimbal [101], the latency threshold is dynamically scaled to capture the SSD sensitivity. We normalize the I/O submission rate into available credits and divide it among contending I/O streams in a weighted-fair manner.

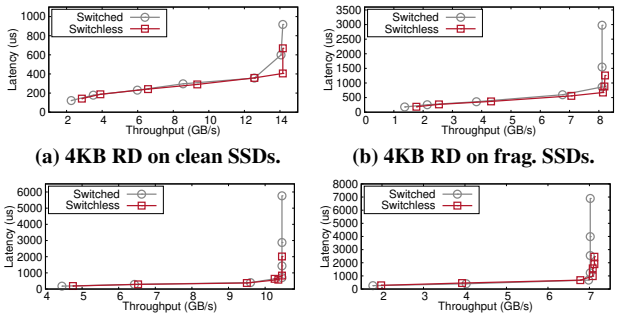
PIFO-based NVMe-oF Packet Scheduler. A switch performs the weighted round-robin scheduling across multiple priority queues. For similar-sized I/Os, one whose target SSD has more available bandwidth would be served faster, which should receive higher priority during transmission. Thus, using the above credit information, we can design a QoS-aware scheduling by computing a "meaningful" priority. This motivates us to apply the PIFO scheduling scheme [117]. Our design generates the rank level for an NVMe-oF packet and chooses an associated queue to insert. The rank calculation is a step function based on the quantum of available credits. The more the credit is, the lower (higher) the rank (priority) will be. Akin to SP-PIFO [49], a packet cannot be inserted at an arbitrary position, simplifying the implementation complexities. Instead, we dynamically adapt the mapping between packet ranks and priority queues. The switch (a) scans from the lowest-priority queue and enqueues a packet in the first queue whose bound is smaller than or equal to the packet rank; (b) triggers the queue bound adaptation using *push-up* and *push-down* primitives when a rank mismatch happens.

SAN Implementation. For the switched SAN, we use the Aurora P4 switch to realize the above scheduler based on SP-PIFO [49]. Our switchless SAN prototype is built atop Gimbal [101]. In both cases, we only compute the rank of the first packet from an NVMe-oF I/O and apply it to all subsequent packets, avoiding packet reordering for the same I/O. Since NVMe-oF completion packets are delivered to clients, rank computation would not be applicable. All I/O read completions share the same rank level. Write completions will use the highest priority queue exclusively.

4.6 System Deployment

Failure Handling. We detect failures similarly to today's SANs by monitoring the link status. A failed link causes a routing detour; a failed NVMe drive (or storage node) would stall ongoing NVMe-oF sessions and stop the remote storage service; a failed switch then entails a large blast radius. Our proposed techniques tolerate failures as follows:

- **Routing Distance and Symmetry.** A failed link or intermediate hop makes the routing distance infinite. Incoming I/O requests avoid these paths during the path selection phase (§4.3). In-flight I/Os, whose completions have passed the failed hop, are not affected. Other I/Os that experience failures during transmission would be routed via another path, temporarily breaking the symmetric property.
- **Reserved Bandwidth.** When a failure is detected, the switch revokes all registered (§4.4) tokens of the associated egress queues. Hence, a returning I/O would go through a new route with no bandwidth control. To minimize the I/O interference, we carry write completions via the highest priority queue and push read responses to the lowest priority one. Hence, one would observe some read I/O throughput degradation during failures.
- **Piggybacked Credit.** The scheduler (§4.5) is unaffected if the switch maintains the available credits of the target drive. When an SSD or a storage node is down, the associated credit information is removed.



(a) 4KB RD on clean SSDs. (b) 4KB RD on frag. SSDs.

(c) 128KB WR on clean SSDs. (d) 128KB WR on frag. SSDs.

Figure 11: Latency versus throughput for 4KB random read and 128KB sequential write on clean and fragmented SSDs.

When a link, switch, or target drive becomes active and rejoins the SAN, the network repopulates the routing table and expands the path candidate vector for choosing the next hop. All the associated routing metadata would be filled in and updated accordingly. A switching hop requires packets belonging to the same I/O to take the same path only if the next hop is down. Packets cannot jump across different paths during transmission to avoid the reordering issue.

Client Connectivity. A SAN provides external connectivity by exposing a number of ports to clients. In a switched SAN, this connectivity is achieved through dedicated upstream switch ports, whereas in a switchless SAN, external access is provisioned via reserved adapter ports. One could combine multiple ports into a single logical interface via NIC teaming [35] and then apply some load-balancing policies.

5 Evaluation: Switched *v.s.* Switchless

We compare the switched and switchless SAN architecture from several dimensions and examine the efficiency of the proposed NVMe-oF-based traffic control techniques. Our evaluations aim to answer the following questions:

- How does the switchless SAN compare with the switched one when delivering NVMe-oF traffic? (§5.1)
- How much cost savings can the switchless SAN achieve in today's deployments? (§5.2)
- How well does the switched and switchless SAN perform at scale? How well do they handle failures? (§5.3, §5.4)
- How effective are the proposed traffic control techniques (i.e., INT-assisted symmetric routing, eager bandwidth reservation, and storage-driven traffic scheduling)? (§5.5)

5.1 Performance: Latency and Throughput

Testbed Results. We configured an experiment with four storage nodes under 4× 100GbE bandwidth. The switched SAN has one path to the ToR combining four ports, while the switchless one has four paths in a 2D Torus topology. Clients mount remote drives via NVMe-over-TCP and run applications atop. First, we issued FIO requests to four NVMe drives from one storage target. Figure 11 reports the latency versus throughput as we gradually increase the request rate. When achieving the maximum bandwidth, on clean SSDs, switchless SAN achieves 32.3% and 18.0% lower latencies than switched SAN for 4KB random read and 128KB sequential write cases. This is because a switchless SAN distributes I/O loads to multiple disjoint paths, where each has a lower queueing delay than the single path in a switched SAN. Reads achieve more latency savings in a switchless SAN because there are fewer concurrent I/Os at each hop, making the token bucket mechanism (used by bandwidth reservation) more effective. Fragmented SSDs present similar behavior with higher latency reduction, i.e., 43.1% and 20.3%. In both switching architectures, we disseminate the target SSD bandwidth availability across the network. However, the switched SAN is more conservative (in terms of rate limiter and rank calculation) when bandwidth over-subscription happens, entailing high queueing latency.

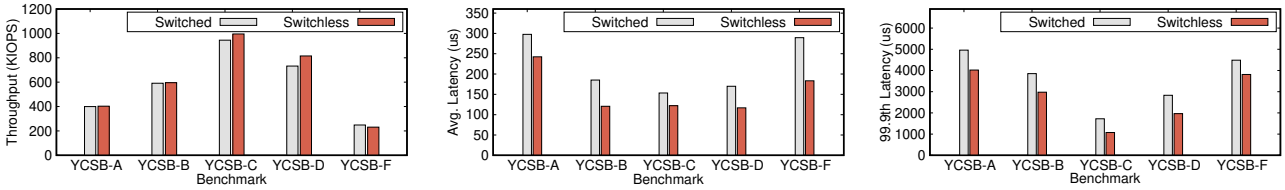
Next, we ported BlobFS on our prototype testbed, ran RocksDB, and evaluated real-world applications' performance. In this experiment, we exposed 8 NVMe SSDs from two storage targets to four clients, each of which ran 2 RocksDB instances (8 in total). Similar to the microbenchmarks, as shown in Figures 12, though achieving nearly the same throughput, the switchless SAN saves the request average/99.9th latencies by 28.3%/25.0% compared to the switched SAN because multiple paths mitigate I/O contention.

Simulation Results. We configured a SAN with 27 storage nodes, and each has 600GbE bandwidth. The switchless case uses the 3D-Torus topology (each node has six paths). Figures 13 compare the read/write request completion time when running three SNIA block traces [40]. First, our proposed traffic control helps reduce latency significantly for both SAN architectures because it embeds the I/O load awareness into the end-to-end data path. As a result, less SSD overloading and request queueing happen. Second, the switchless SAN still outperforms the switched one, e.g., 50.0%, 19.5%, and 15.9% lower average read latency on OLTP, Systor, and WebSearch scenarios, respectively. This is because the switchless SAN provides multiple non-overlapping paths, enabling proportional distribution of I/O requests across them according to the runtime load, thereby reducing queueing delays. Our simulation results align with what we observed in the testbed.

Takeaway. The switchless SAN achieves comparable bandwidth as the switched SAN. Additional routing hops jeopardize performance little because SSD accesses dominate. Instead, the multi-path capability plus the end-to-end I/O load awareness allows the switchless SAN to load balance I/O reads/writes over different paths, reducing I/O contention/interference, and yielding better average/tail latency.

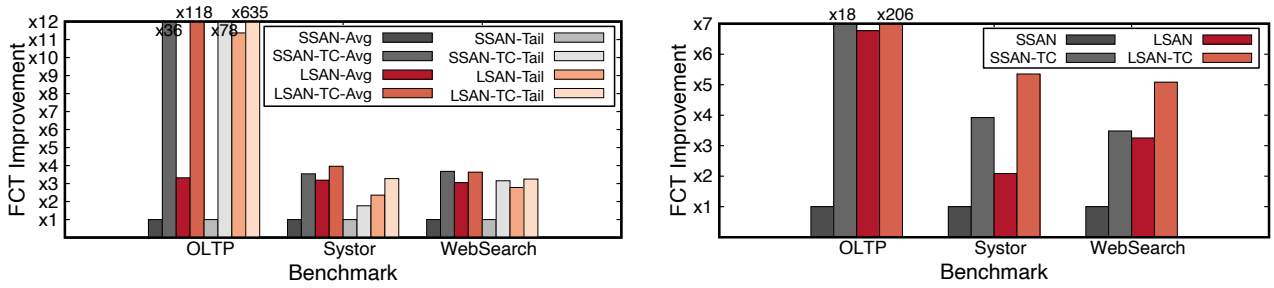
5.2 Cost Analysis

This section compares the capital cost of building a switched and switchless SAN when achieving the same bandwidth. Note that our analyses focus on networking infrastructure (including switches and adapters). We assume that (a) the



(a) Throughput. (b) Average read latency. (c) 99.9th read latency.

Figure 12: RocksDB performance comparison over the Switched/Switchless SAN. We configure the YCSB [58] to generate 10M 1KB key-value pairs with a Zipfian distribution of skewness 0.99 for each DB instance.



(a) Read I/O. (b) Write I/O.

Figure 13: Average and tail FCT (flow completion time) improvement of read and write requests of real applications traces. The switchless SAN uses the 3D-Torus topology. We also report the results when disabling the traffic control (§4) for switched and switchless SAN. SSAN/SSAN-TC=Switched SAN with/without traffic control. LSAN/LSAN-TC=Switchless SAN with/without traffic control.

Parameter	Value	Parameter	Value
Storage Node # (Y)	20	Adapters Per Node (B)	2
Ports Per SSAN Adapter (P)	2	Ports Per SSAN Switch (Q)	32
Ports Per LSAN Adapter (R)	4	SSAN Adapter Price ($)	900
SSAN Switch Price ($)	23,000	LSAN Adapter Price ($)	2,600

Table 1: A SAN configuration example for cost analysis. SSAN=Switched SAN. LSAN=Switchless SAN.

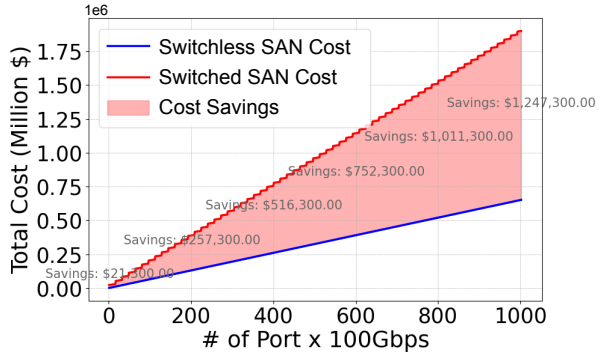


Figure 14: Cost comparison between the switchless and switched SAN when varying storage bandwidth.

aggregated SAN bandwidth and per-port bandwidth are represented as B and B_p; (b) all ports at the switch and adapter have the same bandwidth; (c) the number of ports of a SAN switch, a switched SAN adapter, and a switchless SAN adapter is Q, P, and R, where $R \geq P$; (d) the storage cluster has Y nodes.

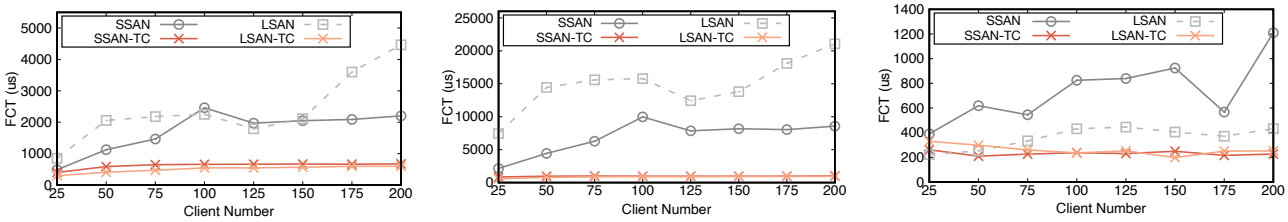
SAN Infrastructure Breakdown. To meet the bandwidth requirement, the switched SAN needs (a) $N_y = \frac{B}{B_p \times P}$ adapters, given each has P ports; (b) $\frac{2 \times B}{B_p \times Q}$ switches, where each has Q ports. The switchless SAN then requires $\frac{B}{B_p \times R}$ since each low-radix SAN adapter has R ports.

Discussion. Table 1 presents a concrete example for our analysis. We use the referenced price of a switch and adapter from CDWG. In the switched SAN, the number of required SAN adapters is $Y \times B = 20 \times 2 = 40$, costing $40 \times \$900 = \$36,000$. Considering (i) a SAN adapter has P ports and (ii) there are B adapters per storage node, the total number of ports is $P \times B \times Y = 2 \times 2 \times 20 = 80$. Thus, we need $\lceil \frac{80}{32} \rceil = 3$ Q-port switches, amounting to $3 \times \$23,000 = \$69,000$. The total expense of a switched SAN is $\$36,000 + \$69,000 = \$105,000$. The switchless SAN needs $R = 4$ ports, $\lceil \frac{80}{4} \rceil = 20$ adapters, costing $20 \times \$2,600 = \$52,000$. Therefore, the switchless SAN architecture yields $53,000 (50.5%) capital savings compared with the switched one. Figure 14 further shows how the capital cost of two SAN architectures scales with the storage bandwidth using the example configuration (Table 1).

Takeaway. A switchless SAN reduces the capital costs of networking infrastructures by replacing the expensive high-radix switches with cheap low-radix adapters, yielding a lower per-port deployment expense.

5.3 Scalability

Workload Diversity. We examined the avg./99.9th read/write I/O completion time when running different synthetic workloads on our prototype testbed. First, the switchless SAN (in a 3D-Torus topology) outperforms the switched one when the I/O size is uniform (i.e., 4KB read/write), reducing the average ready latency by 39.7%. Second, we observe more benefits when mixing 4KB, 32KB, and 128KB I/Os, i.e., 85.4%/91.2% read/write latency reduction. This is because different-sized I/Os can take non-conflict paths, yielding less interface. Third, under I/O incast (where one NVMe drive accepts requests), both switched and switchless SANs behave similarly due to



(a) Average read FCT. (b) 99.9th read FCT. (c) Average write FCT.

Figure 15: Average/Tail FCT (flow completion time) of read/write on 3D-Torus topology as increasing the number of clients. SSAN/SSAN-TC=Switched SAN with/without traffic control. LSAN/LSAN-TC=Switchless SAN with/without traffic control.

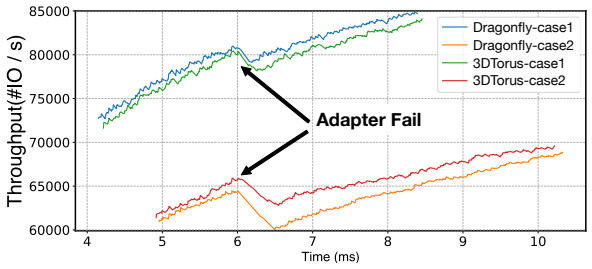


Figure 16: Perf. under an adapter failure in a switchless SAN.

the end-to-end storage control.

Scaling Client Numbers. We increased the number of clients for 4/32/128KB mixed request sizes with 95/5 and 50/50 R/W ratios via simulation. The switched SAN supports more clients using a two-layer FatTree topology. In contrast, the switchless SAN employs a 3D Torus topology, leveraging the additional dimension to scale to a larger number of client nodes. Figure 15 depicts the latency and write latency comparison for four cases. Both switched and switchless architectures scale well when enabling the traffic control techniques, embodying a much more stable read/write latency. When disabled, SSD overloading and in-network queueing emerge with more concurrent client streams.

Takeaway. Both switchless and switched SAN can accommodate client scaling with our proposed traffic control. The switchless architecture is adept at handling traffic heterogeneity due to its inherent path diversity.

5.4 Reliability

The switched SAN has a single-point failure issue unless a redundant ToR is used. The switchless SAN entails better fault-tolerant capability because it can switch to another path. As described in §4.6, only in-flight I/Os are affected as the chosen path might become invalid. The I/O throughput will be recovered when there is enough networking bandwidth. Figure 16 depicts the throughput variation during an adapter failure on our simulation testbed. For a uniform 4KB workload (read/write ratio=95/5), we observe a 1.6%/2.1% throughput drop when the failure happens under 3D-Torus and dragonfly topologies, respectively. It then takes 0.5ms/0.75ms to recover to the normal operational mode, which means that all affected I/Os are completed. For a mixed 4KB/32KB/128KB workload (read/write ratio=50/50), the throughput drop of two topologies is 6.8%/4.5% and the failure handling takes 2ms/1.8ms. It experiences more performance degradation because several paths are in use with more ongoing I/Os.

Takeaway. The switchless SAN can tolerate single-point failure (i.e., ToR switch) that happens in the switched case because it introduces multiple paths for I/O delivery. When failures happen, all ongoing reads/writes are affected and should be rerouted when the network becomes stable.

5.5 Traffic Control Drill-Down

This section examines the effectiveness of the three proposed NVMe-oF-based traffic control techniques.

INT-assisted Symmetric Routing (§4.3). We took the switchless SAN as an example and evaluated the technique under four topologies (i.e., daisy chain, ring, dual-ring, and 2D Torus), given that our developed adapter has up to four ports. We configured four NVMe-oF clients and issued a mixed 70/30 read/write I/O workload to storage nodes. Note that the server is configured with 200/200/300/400 GbE in four topologies. Figure 17 reports the results, where we take switched SAN (400GbE) as a comparison. When the network presents fewer paths to the destination, one could easily observe bandwidth over-subscription and in-network queueing buildup, causing latency increases. For example, under a Ring topology, compared with the switched case, the switchless SAN increases the read and write latency by 25.8% and 38.3%, respectively. When there are more path diversities (such as the 2D mesh case), the switchless SAN outperforms the switched one by 16.1% and 6.4% in terms of reads and writes.

Eager Bandwidth Reservation (§4.4). We evaluated the bandwidth reservation scheme by varying the over-commitment ratio in a switchless SAN (2D Torus topology). Our experiment uses a mixed 4KB/128KB random read workload (as the reservation only works for reads). Figure 18 presents the aggregated bandwidth (y1-axis) and 4KB read latency (y2-axis). Without reservation, one can achieve maximum bandwidth but the latency is high. For example, we observe 4.7× and 9.0× higher latency for the 16/32 concurrent I/O streams cases, respectively. When the reservation performs conservatively (with over-commitment ratio=1), one achieves 49.6% and 62.1% of the maximum bandwidth when there are only 1 or 4 streams as the switching hop throttles the response traffic even if there is available bandwidth. With higher over-commitment, the SAN can max out I/O throughput with a modest latency increase for 16 concurrent streams.

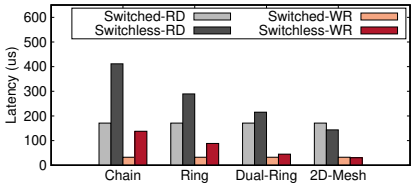


Figure 17: Read/Write latency under different topologies for the 4KB mixed I/O case comparing switched and switchless SAN.

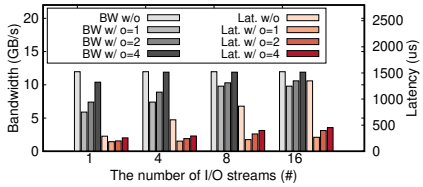


Figure 18: Bandwidth and Latency under different reservations (over-commitment ratio) for the 4KB/128KB mixed scenario.

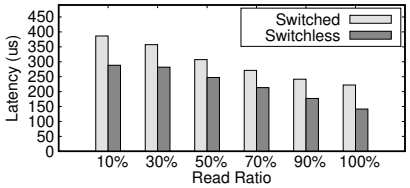


Figure 19: 4KB random read latency varying the mixed read/write ratio comparing switched and switchless SAN.

Storage-driven Traffic Scheduling (§4.5). We varied the workload pattern on fragmented SSDs and examined the efficacy of the storage-driven design. In both switched and switchless SAN, we continuously monitor the SSD condition, estimate the drive I/O serving capacity, translate it to credits, and propagate the information back to the SAN. As described in §4.5, we use these statistics in two ways: clients use it for rate control, and switching hops use it for I/O scheduling. Thus, without such support, one can only use network conditions for congestion control, and clients would issue more I/Os, exceeding drive capacity, and causing drastic queueing delays. On average across all cases (Figure 19), the switchless SAN saves 25.1% of latencies than the switched one.

Takeaway. NVMe-oF disaggregated storage brings in three unique traffic characteristics (§4.1). Co-designing them with the traffic control can fully unleash the capability of the underlying SAN architecture (regardless of switched and switchless), and max out the bandwidth utilization.

6 Related Work

Direct-connect topology. Researchers have explored it extensively in HPC and rack-scale computing [53, 59, 60, 65, 86]. [86] introduces the dragonfly topology using high-radix routers. It applies the global adaptive routing scheme and proposes two optimizations: selective virtual-channel discrimination and global channel congestion detection. Slim Fly [53] proposes a cost-effective direct-connect topology that approaches the theoretically optimal network diameter. R2C2 [59] offers multi-path communication and performs per-flow bandwidth reservation over a broadcast primitive. We study how to co-design NVMe-oF flow characteristics with the traffic control for the switched/switchless SAN.

Storage area network. SANs, a specialized high-performance network fabric, have been widely deployed in enterprise clusters and data centers. There are a plethora of commodity SAN solutions [3, 21, 22, 28, 34, 43] that provide fast remote access to a shared pool of block storage. Researchers have developed HW/SW optimizations to improve the performance and efficiency of existing SAN deployments. For example, Ana et al. [88] characterized the performance of iSCSI-based SAN and proposed several optimizations. QuickSAN [55] proposes a low-latency hardware-accelerated block transport and directly integrates it into an SSD. People also use emerging programmable networks [54, 93, 95, 97–99, 106–108, 111, 112] and cluster interconnects [12, 50, 79, 92, 94, 120] to redesign SAN. We examine whether a switched or switchless SAN architecture is more suitable for expanding bandwidth capacity.

Disaggregated storage system. Researchers have developed several new software I/O stacks and hardware architectures for disaggregated storage [57, 74, 75, 77, 80–82, 84, 89–91, 96, 101–103, 110, 113, 121–123]. For example, Reflex [89] introduces a kernel-bypass data-plane for I/O requests orchestrated by credits. blk-switch [81] integrates the switch abstraction into the Linux storage stack and provides performance isolation. We present a comparative analysis of switched and switchless networking architectures in the context of storage disaggregation. A key learning is that co-designing the NVMe-oF protocol with the underlying network traffic control enables more streamlined and efficient processing of remote read and write I/O requests. This observation highlights the importance of aligning protocol semantics with network characteristics to minimize overhead and reduce end-to-end latency. Our results suggest that incorporating lightweight in-network computing primitives, such as support for I/O-aware telemetry, adaptive rate control, and packet scheduling, can enhance both the performance and efficiency of disaggregated storage systems.

7 Conclusion

This paper quantitatively compares switched and switchless SAN architectures when holding disaggregated dense storage appliances. We first design an experimental methodology–consisting of small-scale real-system prototypes and large-scale simulations–that provides adequate flexibility for SAN architecture exploration. We then co-design the SAN traffic control with NVM-oF traffic characteristics to optimize the storage flow performance. Last, our evaluations draw several findings and show that the switchless SAN entails several advantages: (a) delivering nearly the same throughput as the switched SAN, albeit enclosing more routing hops, but with lower latency, because of the multiple load-aware I/O paths, mitigating I/O interference; (b) saving capital costs by replacing expensive high-radix switches; (c) scaling with heterogenous I/O flows and improving SAN's reliability.

Acknowledgement

We would like to thank the anonymous reviewers and our shepherd, Michio Honda, for their comments and feedback. This work is supported in part by NSF grants CNS-2106199, CNS-2212192, and CAREER-2339755.

References

[1] Fungible Accelerator Card. https://www.servethehome.com/fungible-fc200-fc100-fc50-nvme-tcp-dpu-adapters-launched/, 2021.

[2] Fungible FS1600 Pushes Hyperscale Storage to the Data Center. https://www.storagereview.com/review/fungible-fs1600-pushes-hyperscale-storage-to-the-data-center, 2022.

[3] Dell EMC PowerStore. https://www.delltechnologies.com/en-us/storage/powerstore-storage-appliance.htm, 2024.

[4] AMD Next Generation "Zen 4" Core and 4th Gen AMD EPYCTM 9004 Server CPU. https://hc2023.hotchips.org/assets/program/conference/day1/CPU1/HC_Zen4_Epyc_Final_20230825%20-%20Embargoed%20until%20Aug%2029%202023.pdf, 2025.

[5] Arista 7280R3 Series. https://www.arista.com/en/products/7280r3-series, 2025.

[6] Azure Blob Storage. https://azure.microsoft.com/en-us/products/storage/blobs, 2025.

[7] Berkeley Extensible Software Switch (BESS). https://github.com/NetSys/bess, 2025.

[8] Broadcom Tomahawk 5 Ethernet Switch. https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm78900-series, 2025.

[9] Brocade G730 Switch. https://www.broadcom.com/products/fibre-channel-networking/switches/g730-switch, 2025.

[10] Cisco Nexus 9300 Series. https://www.cisco.com/site/us/en/products/networking/cloud-networking-switches/nexus-9300-series-switches/index.html, 2025.

[11] Cloud Storage on AWS. https://aws.amazon.com/products/storage/, 2025.

[12] Compute Express Link (CXL) Specification. https://computeexpresslink.org/cxl-specification/, 2025.

[13] Data Plane Development Kit (DPDK). https://www.dpdk.org, 2025.

[14] E1 and E3 EDSFF. https://www.servethehome.com/e1-and-e3-edsff-to-take-over-from-m-2-and-2-5-in-ssds-kioxia/, 2025.

[15] Enterprise and Datacenter Standard Form Factor (EDSFF). https://www.snia.org/forums/cmsi/knowledge/formfactors#EDSFF, 2025.

[16] Fibre Channel Standard. https://www.incits.org/committees/t11, 2025.

[17] Flexible I/O tester. https://fio.readthedocs.io/en/latest/fio_doc.html, 2025.

[18] Google AlloyDB Overview. https://cloud.google.com/alloydb/docs/overview, 2025.

[19] Google Cloud Storage. https://cloud.google.com/storage, 2025.

[20] High-Capacity SSDs for AI/ML using Disaggregated Storage Solution: Performance Test Results Show Promise. https://semiconductor.samsung.com/us/news-events/tech-blog/high-capacity-ssds-for-ai-ml-using-disaggregated-storage-solution-performance-test-results-show-promise/, 2025.

[21] Hitachi Virtual Storage Platform G Series. https://www.hitachivantara.com/en-us/products/storage/flash-storage/vsp-g-series.html, 2025.

[22] HP Nimble Storage. https://www.hpe.com/us/en/storage/nimble.html, 2025.

[23] HPE 4-port 16Gb SAN Adapter. https://buy.hpe.com/us/en/options/storage-options/3par-options/3par-storage-options/hpe-3par-storeserv-8000-4%E2%80%91port-16gb-fibre-channel-adapter/p/h6z00a, 2025.

[24] HPE Aruba Networking 8-port Module. https://buy.hpe.com/us/en/options/modules/switch-modules/switch-connectivity-modules/hpe-aruba-networking-8%E2%80%91port-1g-10gbe-sfp-macsec-v3-zl2-module/p/j9993a, 2025.

[25] HPE Nimble Storage. https://www.hpe.com/us/en/storage/nimble.html, 2025.

[26] HPE Storage Switch M-series SN4700. https://www.hpe.com/psnow/doc/PSN1014646700PTEN, 2025.

[27] HTsim Simulator. http://nets.cs.pub.ro/~costin/code.html, 2025.

[28] IBM FlashSystem. https://www.ibm.com/it-infrastructure/storage/flash, 2025.

[29] Intel Ethernet Network Adapter E810-XXVDA4. https://www.intel.com/content/www/us/en/products/sku/192560/intel-ethernet-network-adapter-e810xxvda4/specifications.html, 2025.

[30] L2 Forwarding Sample Application. https://doc.dpdk.org/guides/sample_app_ug/l2_forward_real_virtual.html, 2025.

[31] Lagrange Multiplier. https://en.wikipedia.org/wiki/Lagrange_multiplier, 2025.

[32] Link Aggregation. https://en.wikipedia.org/wiki/Link_aggregation, 2025.

[33] Marvell Teralynx 51.2T Ethernet Switch. https://www.marvell.com/company/newsroom/marvell-teralynx-512t-ethernet-switch-enters-volume-production-for-global-ai-cloud-deployments.html, 2025.

[34] NetApp AFF A-Series Arrays. https://www.netapp.com/data-storage/aff-a-series/, 2025.

[35] NIC Teaming. https://docs.microsoft.com/en-us/windows-server/networking/technologies/nic-teaming/nic-teaming, 2025.

[36] NVMe Specifications Overview. https://nvmexpress.org/specifications/, 2025.

[37] PCI Express Specification. https://pcisig.com/specifications, 2025.

[38] PCIe Lanes and Bandwidth Increase Over a Decade for Intel Xeon and AMD EPYC. https://www.servethehome.com/pcie-lanes-and-bandwidth-increase-over-a-decade-for-intel-xeon-and-amd-epyc/, 2025.

[39] RocksDB: A persistent key-value store for fast storage environments. https://rocksdb.org, 2025.

[40] SNIA Block I/O Traces. http://iotta.snia.org/traces/block-io, 2025.

[41] Storage Performance Development Kit (SPDK). https://spdk.io, 2025.

[42] Supermicro All-Flash NVMe Storage Server. https://www.supermicro.com/en/products/system/storage/2u/ssg-222b-ne3x24r, 2025.

[43] The InfiniBox Enterprise Storage System. https://www.infinidat.com/en/products-technology/infinibox, 2025.

[44] The Quality of Service (QoS) Framework. https://doc.dpdk.org/guides/prog_guide/qos_framework.html, 2025.

[45] ThinkSystem DG5000 All-Flash Array. https://www.lenovo.com/us/en/p/servers-storage/storage/unified-storage/dg-series-all-flash/thinksystem-dg5000-all-flash-array/len22ts0022, 2025.

[46] Ultra-Low Latency with Samsung Z-NAND SSD. https://download.semiconductor.samsung.com/resources/brochure/Ultra-Low%20Latency%20with%20Samsung%20Z-NAND%20SSD.pdf, 2025.

[47] Unveiling Intel's 2024 Xeon Architecture. https://www.intel.com/content/www/us/en/content-details/787432/hot-chips-2023-granite-rapids-and-sierra-forest-xeon-press-briefing-presentation.html, 2025.

[48] Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D Davis, Mark S Manasse, and Rina Panigrahy. Design tradeoffs for ssd performance. In *USENIX Annual Technical Conference*, 2008.

[49] Albert Gran Alcoz, Alexander Dietmüller, and Laurent Vanbever. SP-PIFO: Approximating Push-In First-Out Behaviors using Strict-Priority Queues. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020.

[50] Seunghyun An, Joontaek Oh, and Ming Liu. Server chiplet networking. In *Proceedings of the 24th ACM Workshop on Hot Topics in Networks*, 2025.

[51] Baruch Awerbuch, Rohit Khandekar, and Satish Rao. Distributed Algorithms for Multicommodity Flow Problems via Approximate Steepest Descent Framework. *ACM Trans. Algorithms*, 9(1), dec 2012.

[52] Wei Bai, Shanim Sainul Abdeen, Ankit Agrawal, Krishan Kumar Attre, Paramvir Bahl, Ameya Bhagat, Gowri Bhaskara, Tanya Brokhman, Lei Cao, Ahmad Cheema, Rebecca Chow, Jeff Cohen, Mahmoud Elhaddad, Vivek Ette, Igal Figlin, Daniel Firestone, Mathew George, Ilya German, Lakhmeet Ghai, Eric Green, Albert Greenberg, Manish Gupta, Randy Haagens, Matthew Hendel, Ridwan Howlader, Neetha John, Julia Johnstone, Tom Jolly, Greg Kramer, David Kruse, Ankit Kumar, Erica Lan, Ivan Lee, Avi Levy, Marina Lipshteyn, Xin Liu, Chen Liu, Guohan Lu, Yuemin Lu, Xiakun Lu, Vadim Makhervaks, Ulad Malashanka, David A. Maltz, Ilias Marinos, Rohan Mehta, Sharda Murthi, Anup Namdhari, Aaron Ogus, Jitendra Padhye, Madhav Pandya, Douglas Phillips, Adrian Power, Suraj Puri, Shachar Raindel, Jordan Rhee, Anthony Russo,

Maneesh Sah, Ali Sheriff, Chris Sparacino, Ashutosh Srivastava, Weixiang Sun, Nick Swanson, Fuhou Tian, Lukasz Tomczyk, Vamsi Vadlamuri, Alec Wolman, Ying Xie, Joyce Yom, Lihua Yuan, Yanzhao Zhang, and Brian Zill. Empowering azure storage with RDMA. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI'23)*, pages 49–67, 2023.

[53] Maciej Besta and Torsten Hoefler. Slim fly: A cost effective low-diameter network topology. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'14)*, 2014.

[54] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. Forwarding metamorphosis: fast programmable match-action processing in hardware for SDN. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, page 99–110, 2013.

[55] Adrian M. Caulfield and Steven Swanson. QuickSAN: A Storage Area Network for Fast, Distributed, Solid State Disks. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, 2013.

[56] Feng Chen, David A Koufaty, and Xiaodong Zhang. Understanding intrinsic characteristics and system implications of flash memory based solid state drives. *ACM SIGMETRICS Performance Evaluation Review*, 37(1):181–192, 2009.

[57] Xuzheng Chen, Jie Zhang, Ting Fu, Yifan Shen, Shu Ma, Kun Qian, Lingjun Zhu, Chao Shi, Yin Zhang, Ming Liu, et al. Demystifying datapath accelerator enhanced off-path smartnic. In *2024 IEEE 32nd International Conference on Network Protocols (ICNP'24)*, pages 1–12, 2024.

[58] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, 2010.

[59] Paolo Costa, Hitesh Ballani, Kaveh Razavi, and Ian Kash. R2C2: A Network Stack for Rack-scale Computers. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM'15)*, page 551–564, 2015.

[60] Paolo Costa, Austin Donnelly, Antony Rowstron, and Greg O'Shea. Camdoop: Exploiting In-network Aggregation for Big Data Applications. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, 2012.

[61] Alex Depoutovitch, Chong Chen, Jin Chen, Paul Larson, Shu Lin, Jack Ng, Wenlin Cui, Qiang Liu, Wei Huang, Yong Xiao, et al. Taurus database: How to be fast, available, and frugal in the cloud. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20)*, pages 1463–1478, 2020.

[62] Jack Edmonds and Richard M Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*, 19(2):248–264, 1972.

[63] Shimon Even, Alon Itai, and Adi Shamir. On the complexity of time table and multi-commodity flow problems. In *16th annual symposium on foundations of computer science (SFCS'75)*, pages 184–193, 1975.

[64] Greg Faanes, Abdulla Bataineh, Duncan Roweth, Tom Court, Edwin Froese, Bob Alverson, Tim Johnson, Joe Kopnick, Mike Higgins, and James Reinhard. Cray cascade: A scalable hpc system based on a dragonfly network. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'12)*, 2012.

[65] Greg Faanes, Abdulla Bataineh, Duncan Roweth, Tom Court, Edwin Froese, Bob Alverson, Tim Johnson, Joe Kopnick, Mike Higgins, and James Reinhard. Cray cascade: a scalable HPC system based on a Dragonfly network. In *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2012.

[66] Dan Gibson, Hema Hariharan, Eric Lance, Moray McLaren, Behnam Montazeri, Arjun Singh, Stephen Wang, Hassan M. G. Wassel, Zhehua Wu, Sunghwan Yoo, Raghuraman Balasubramanian, Prashant Chandra, Michael Cutforth, Peter Cuy, David Decotigny, Rakesh Gautam, Alex Iriza, Milo M. K. Martin, Rick Roy, Zuowei Shen, Ming Tan, Ye Tang, Monica Wong-Chan, Joe Zbiciak, and Amin Vahdat. Aquila: A unified, low-latency fabric for datacenter networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI'22)*, pages 1249–1266, 2022.

[67] Garth A Gibson and Rodney Van Meter. Network attached storage architecture. *Communications of the ACM (CACM'20)*, 43(11):37–45, 2000.

[68] Andrew V Goldberg and Robert E Tarjan. Finding minimum-cost circulations by canceling negative cycles. *Journal of the ACM (JACM)*, 36(4):873–886, 1989.

[69] Donghyun Gouk, Miryeong Kwon, Jie Zhang, Sungjoon Koh, Wonil Choi, Nam Sung Kim, Mahmut Kandemir, and Myoungsoo Jung. Amber: Enabling Precise Full-System Simulation with Detailed Modeling of All SSD Resources. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018.

[70] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. VL2: A scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication (SIGCOMM'09)*, pages 51–62, 2009.

[71] Ajay Gulati, Irfan Ahmad, and Carl A. Waldspurger. PARDA: Proportional Allocation of Resources for Distributed Storage Access. In *7th USENIX Conference on File and Storage Technologies (FAST'09)*, 2009.

[72] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. BCube: a high performance, server-centric network architecture for modular data centers. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication (SIGCOMM'09)*, pages 63–74, 2009.

[73] Chuanxiong Guo, Haitao Wu, Kun Tan, Lei Shi, Yongguang Zhang, and Songwu Lu. Dcell: a scalable and fault-tolerant network structure for data centers. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication (SIGCOMM'08)*, page 75–86, 2008.

[74] Zerui Guo, Jiaxin Lin, Yuebin Bai, Daehyeok Kim, Michael Swift, Aditya Akella, and Ming Liu. LogNIC: A High-Level Performance Model for SmartNICs. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'23)*, page 916–929, 2023.

[75] Zerui Guo, Hua Zhang, Chenxingyu Zhao, Yuebin Bai, Michael Swift, and Ming Liu. LEED: A Low-Power, Fast Persistent Key-Value Store on SmartNIC JBOFs. In *Proceedings of the ACM SIGCOMM 2023 Conference (SIGCOMM'23)*, page 1012–1027, 2023.

[76] Refael Hassin. The minimum cost flow problem: a unifying approach to dual algorithms and a new tree-search algorithm. *Mathematical Programming*, 25:228–239, 1983.

[77] Yongchao He, Wenfei Wu, Yanfang Le, Ming Liu, and ChonLam Lao. A Generic Service to Provide In-Network Aggregation for Key-Value Streams. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'23), Volume 2*, page 33–47, 2023.

[78] Charles L Hedrick. RFC1058: Routing information protocol, 1988.

[79] Wentao Hou, Jie Zhang, Zeke Wang, and Ming Liu. Understanding Routable PCIe Performance for Composable Infrastructures. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI'24)*, pages 297–312, 2024.

[80] Jaehyun Hwang, Qizhe Cai, Ao Tang, and Rachit Agarwal. TCP $\approx$ RDMA: CPU-efficient remote storage access with i10. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI'20)*, 2020.

[81] Jaehyun Hwang, Midhul Vuppalapati, Simon Peter, and Rachit Agarwal. Rearchitecting linux storage stack for µs latency and high throughput. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI'21)*, 2021.

[82] Sheng Jiang and Ming Liu. Building an Elastic Block Storage over EBOFs Using Shadow Views. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI'25)*, pages 1137–1153, 2025.

[83] David B Johnson and David A Maltz. Dynamic source routing in ad hoc wireless networks. *Mobile computing*, pages 153–181, 1996.

[84] Yuyuan Kang and Ming Liu. Understanding and Profiling NVMe-over-TCP Using ntprof. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI'25)*, pages 1117–1136, 2025.

[85] Brad Karp and Hsiang-Tsung Kung. GPSR: Greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 243–254, 2000.

[86] John Kim, Wiliam J. Dally, Steve Scott, and Dennis Abts. Technology-driven, highly-scalable dragonfly topology. In *2008 International Symposium on Computer Architecture (ISCA'08)*, pages 77–88, 2008.

[87] Morton Klein. A primal method for minimal cost flows with applications to the assignment and transportation problems. *Management Science*, 14(3):205–220, 1967.

[88] Ana Klimovic, Christos Kozyrakis, Eno Thereska, Binu John, and Sanjeev Kumar. Flash Storage Disaggregation. In *Proceedings of the Eleventh European Conference on Computer Systems*, 2016.

[89] Ana Klimovic, Heiner Litz, and Christos Kozyrakis. ReFlex: Remote Flash ≈ Local Flash. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, 2017.

[90] Sergey Legtchenko, Hugh Williams, Kaveh Razavi, Austin Donnelly, Richard Black, Andrew Douglas, Nathanael Cheriere, Daniel Fryer, Kai Mast, Angela Demke Brown, Ana Klimovic, Andy Slowey, and Antony Rowstron. Understanding Rack-Scale Disaggregated Storage. In *9th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage'17)*, 2017.

[91] Huaicheng Li, Mingzhe Hao, Stanko Novakovic, Vaibhav Gogte, Sriram Govindan, Dan R. K. Ports, Irene Zhang, Ricardo Bianchini, Haryadi S. Gunawi, and Anirudh Badam. LeapIO: Efficient and Portable Virtual NVMe Storage on ARM SoCs. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020.

[92] Xiao Li, Zerui Guo, Yuebin Bai, Mehash Ketkar, Hugh Willkinson, and Ming Liu. Understanding and Profiling CXL.mem Using PathFinder. In *Proceedings of the ACM SIGCOMM 2025 Conference (SIGCOMM'25)*, 2025.

[93] Ming Liu. *Building Distributed Systems Using Programmable Networks*. University of Washington, 2020.

[94] Ming Liu. Fabric-Centric Computing. In *Proceedings of the 19th Workshop on Hot Topics in Operating Systems (HotOS'23)*, page 118–126, 2023.

[95] Ming Liu, Tianyi Cui, Henry Schuh, Arvind Krishnamurthy, Simon Peter, and Karan Gupta. Offloading distributed applications onto smartNICs using iPipe. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM'19)*, page 318–333, 2019.

[96] Ming Liu, Arvind Krishnamurthy, Harsha V. Madhyastha, Rishi Bhardwaj, Karan Gupta, Chinmay Kamat, Huapeng Yuan, Aditya Jaltade, Roger Liao, Pavan Konka, and Anoop Jawahar. Fine-Grained Replicated State Machines for a Cluster Storage System . In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI'20)*, pages 305–323, 2020.

[97] Ming Liu, Liang Luo, Jacob Nelson, Luis Ceze, Arvind Krishnamurthy, and Kishore Atreya. IncBricks: Toward In-Network Computation with an In-Network Cache. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'17)*, page 795–809, 2017.

[98] Ming Liu, Simon Peter, Arvind Krishnamurthy, and Phitchaya Mangpo Phothilimthana. E3: Energy-Efficient Microservices on SmartNIC-Accelerated Servers. In *2019 USENIX Annual Technical Conference (USENIX ATC'19)*, pages 363–378, 2019.

[99] Liang Luo, Ming Liu, Jacob Nelson, Luis Ceze, Amar Phanishayee, and Arvind Krishnamurthy. Motivating in-network aggregation for distributed deep neural network training. In *Workshop on Approximate Computing Across the Stack*, 2017.

[100] Rui Miao, Lingjun Zhu, Shu Ma, Kun Qian, Shujun Zhuang, Bo Li, Shuguang Cheng, Jiaqi Gao, Yan Zhuang, Pengcheng Zhang, et al. From luna to solar: the evolutions of the compute-to-storage networks in alibaba cloud. In *Proceedings of the ACM SIGCOMM 2022 Conference (SIGCOMM'22)*, pages 753–766, 2022.

[101] Jaehong Min, Ming Liu, Tapan Chugh, Chenxingyu Zhao, Andrew Wei, In Hwan Doh, and Arvind Krishnamurthy. Gimbal: enabling multi-tenant storage disaggregation on SmartNIC JBOFs. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference (SIGCOMM'21)*, page 106–122, 2021.

[102] Jaehong Min, Chenxingyu Zhao, Ming Liu, and Arvind Krishnamurthy. eZNS: An elastic zoned namespace for commodity ZNS SSDs. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI'23)*, pages 461–477, 2023.

[103] Jaehong Min, Chenxingyu Zhao, Ming Liu, and Arvind Krishnamurthy. eZNS: Elastic Zoned Namespace for Enhanced Performance Isolation and Device Utilization. *ACM Trans. Storage*, 20(3), June 2024.

[104] Dritan Nace, Nhat-Linh Doan, Eric Gourdin, and Bernard Liau. Computing Optimal Max-Min Fair Resource Allocation for Elastic Flows. *IEEE/ACM Transactions on Networking*, 14(6), 2006.

[105] David A. Patterson, Garth Gibson, and Randy H. Katz. A case for redundant arrays of inexpensive disks (raid). In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data (SIGMOD'88)*, page 109–116, 1988.

[106] Phitchaya Mangpo Phothilimthana, Ming Liu, Antoine Kaufmann, Simon Peter, Rastislav Bodik, and Thomas Anderson. Floem: A Programming System for NIC-Accelerated Network Applications. In *13th USENIX*

Symposium on Operating Systems Design and Implementation (OSDI'18), pages 663–679, 2018.

[107] Yiming Qiu, Qiao Kang, Ming Liu, and Ang Chen. Clara: Performance Clarity for SmartNIC Offloading. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks (HotNets'20)*, page 16–22, 2020.

[108] Yiming Qiu, Jiarong Xing, Kuo-Feng Hsu, Qiao Kang, Ming Liu, Srinivas Narayana, and Ang Chen. Automated SmartNIC Offloading Insights for Network Functions. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP'21)*, page 772–787, 2021.

[109] Mark R Schibilla and Randy J Reiter. Garbage collection for solid state disks, April 24 2012. US Patent 8,166,233.

[110] Henry N. Schuh, Weihao Liang, Ming Liu, Jacob Nelson, and Arvind Krishnamurthy. Xenic: SmartNIC-Accelerated Distributed Transactions. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP'21)*, page 740–755, 2021.

[111] Naveen Kr. Sharma, Ming Liu, Kishore Atreya, and Arvind Krishnamurthy. Approximating Fair Queueing on Reconfigurable Switches. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI'18)*, pages 1–16, 2018.

[112] Naveen Kr. Sharma, Chenxingyu Zhao, Ming Liu, Pravein G Kannan, Changhoon Kim, Arvind Krishnamurthy, and Anirudh Sivaraman. Programmable Calendar Queues for High-speed Packet Scheduling . In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI'20)*, pages 685–699, 2020.

[113] Vishal Shrivastav, Asaf Valadarsky, Hitesh Ballani, Paolo Costa, Ki Suh Lee, Han Wang, Rachit Agarwal, and Hakim Weatherspoon. Shoal: A Network Architecture for Disaggregated Racks. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI'19)*, 2019.

[114] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, Anand Kanagala, Jeff Provost, Jason Simmons, Eiichi Tanda, Jim Wanderer, Urs Hölzle, Stephen Stuart, and Amin Vahdat. Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM'15)*, page 183–197, 2015.

[115] Arjun Singhvi, Nandita Dukkipati, Prashant Chandra, Hassan M. G. Wassel, Naveen Kr. Sharma, Anthony Rebello, Henry Schuh, Praveen Kumar, Behnam Montazeri, Neelesh Bansod, Sarin Thomas, Inho Cho, Hyojeong Lee Seibert, Baijun Wu, Rui Yang, Yuliang Li, Kai Huang, Qianwen Yin, Abhishek Agarwal, Srinivas Vaduvatha, Weihuang Wang, Masoud Moshref, Tao Ji, David Wetherall, and Amin Vahdat. Falcon: A Reliable, Low Latency Hardware Transport. In *Proceedings of the ACM SIGCOMM 2025 Conference*, page 248–263, 2025.

[116] Ankit Singla, Chi-Yao Hong, Lucian Popa, and P Brighten Godfrey. Jellyfish: Networking data centers randomly. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI'12)*, pages 225–238, 2012.

[117] Anirudh Sivaraman, Suvinay Subramanian, Mohammad Alizadeh, Sharad Chole, Shang-Tse Chuang, Anurag Agrawal, Hari Balakrishnan, Tom Edsall, Sachin Katti, and Nick McKeown. Programmable Packet Scheduling at Line Rate. In *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016.

[118] Carl A Sunshine. Source routing in computer networks. *ACM SIGCOMM Computer Communication Review*, 7(1):29–33, 1977.

[119] Alexandre Verbitski, Anurag Gupta, Debanjan Saha, Murali Brahmadesam, Kamal Gupta, Raman Mittal, Sailesh Krishnamurthy, Sandor Maurice, Tengiz Kharatishvili, and Xiaofeng Bao. Amazon aurora: Design considerations for high throughput cloud-native relational databases. In *Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD'17)*, pages 1041–1052, 2017.

[120] Xincheng Xie, Wentao Hou, Zerui Guo, and Ming Liu. Building massive MIMO baseband processing on a Single-Node supercomputer. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI'25)*, pages 1221–1242, 2025.

[121] Jie Zhang, Hongjing Huang, Xuzheng Chen, Xiang Li, Jieru Zhao, Ming Liu, and Zeke Wang. RpcNIC: Enabling Efficient Datacenter RPC Offloading on PCIe-attached SmartNICs. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA'25)*, pages 1379–1394, 2025.

[122] Chenxingyu Zhao, Tapan Chugh, Jaehong Min, Ming Liu, and Arvind Krishnamurthy. Dremel: Adaptive Configuration Tuning of RocksDB KV-Store. *Proc. ACM Meas. Anal. Comput. Syst.*, 6(2), June 2022.

[123] Chenxingyu Zhao, Jaehong Min, Ming Liu, and Arvind Krishnamurthy. White-Boxing RDMA with Packet-Granular Software Control. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI'25)*, pages 427–449, 2025.