SSR: Synchronous State Machine Replication with a Timing-Aligned Split Architecture

Yuke Ma
Max Planck Institute for Informatics
Saarbrücken, Germany

Paolo Costa
Microsoft Research
Cambridge, United Kingdom

Yiting Xia
Max Planck Institute for Informatics
Saarbrücken, Germany

Abstract

Modern datacenters exhibit different timing properties across the network fabric and the host software stack: the former offers bounded-delay delivery and precise clock synchronization, whereas the latter provides greater flexibility but substantially greater timing variability. This substrate-level separation matches the functional structure of State Machine Replication (SMR), whose normal case is timing-sensitive and repetitive while recovery requires flexible coordination, state transfer, and membership reconfiguration. We present SSR, a synchronous state machine replication protocol. SSR exploits this alignment with a split protocol design: normal-case replication runs as a round-based synchronous SMR protocol close to the network fabric, where replicas propose concurrently and commit from per-round local evidence without a steady-state leader, while recovery remains in a host-side path. Our preliminary evaluation results show that SSR improves common-case throughput by 14.9× with three replicas and 24.9× with five replicas, completes recovery in about 0.15 ms, and commits on the fast path after two rounds.

CCS Concepts

• **Computing methodologies** → **Distributed algorithms**; • **Computer systems organization** → **Reliability**; • **Networks** → **Data center networks**.

Keywords

state machine replication, synchronous replication, datacenter networks, network acceleration

ACM Reference Format:
Yuke Ma, Paolo Costa, and Yiting Xia. 2026. SSR: Synchronous State Machine Replication with a Timing-Aligned Split Architecture. In *17th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys '26), September 17–18, 2026, Bangkok, Thailand.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3838177.3841728

APSys '26, Bangkok, Thailand

ACM ISBN 979-8-4007-2912-6/2026/09
https://doi.org/10.1145/3838177.3841728

1 Introduction

Consensus protocols and state machine replication are core building blocks of fault-tolerant distributed systems, and remain critical to many modern datacenter applications [17, 24, 28]. As distributed services place increasing pressure on throughput and latency, the replication path has become a central performance concern [22, 26, 31]. Most high-performance SMR systems respond by reducing host overhead or offloading selected replication functions to the network or specialized hardware [1–3, 9, 14, 15, 20, 30, 34]. These systems make replication faster, but do not fundamentally remove the timing variability that drives long-tail latency in traditional timeout-driven designs. More recently, synchronous SMR systems [27, 32] revisit the protocol itself under stronger timing assumptions. Yet making the entire replication protocol synchronous remains difficult in practice: it requires tight control over host-path timing, highly controlled deployments, or both.

Our key observation, supported by measurements in Section 2, is that modern datacenters expose different timing properties across the network fabric and the host software stack. The network fabric is optimized for bounded-delay delivery: packets traverse NIC and switch pipelines with low jitter and repeatable processing [19, 25, 29, 33], while hardware-assisted clock synchronization can bound clock skew [4, 10, 18, 21]. The host software stack, in contrast, provides richer programmability but is subject to OS scheduling, interrupt moderation, cache effects, PCIe/DMA queueing, and application-level control flow, making its timing substantially more variable. This substrate-level separation matches the functional structure of SMR: normal-case replication is timing-sensitive and repetitive, while failure recovery is irregular, coordination-heavy, and requires flexible state transfer, membership reconfiguration, and recovery coordination. The natural conclusion is not to force all of SMR into a single timing discipline, but to split it so that each function runs where its requirements are best served.

Guided by this observation, we design SSR, a synchronous state machine replication protocol built around a split protocol design. SSR splits SMR into a normal-case fast path and a host-side recovery path. The fast path is a round-based synchronous protocol that runs close to the network fabric (e.g., on a SmartNIC): replicas propose concurrently, derive commits from per-round evidence, and avoid steady-state leaders and timeout-driven coordination. The recovery path coordinates recovery, transfers missing state, and installs a renewed membership. This lets SSR combine the benefits of both directions: it obtains the performance and timing advantages of synchronous SMR without forcing recovery into a rigid timing model, and preserves the flexibility of split designs without merely offloading pieces of an existing timeout-driven protocol.

We present the design and evaluation of SSR, including its synchronous fast path, host-side recovery path, and the interface that separates them. We evaluate SSR across steady-state replication and failure recovery. Our preliminary results show that, compared with representative baselines, SSR improves common-case throughput by 14.9× with three replicas and 24.9× with five replicas, commits on the fast path after two rounds (4 μs with a 2 μs round length), excluding host-side command injection and log-delivery overhead, and completes recovery in about 0.15 ms instead of relying on host-driven timeouts that often range from single-digit to tens of milliseconds.

2 Related Work and Motivation

Existing SMR. Most high-performance SMR systems use modern datacenter capabilities to accelerate existing protocol mechanisms. Some keep consensus in software while reducing ordering and coordination cost: Speculative Paxos [30] uses optimistic execution, NOPaxos [20] relies on ordered network delivery, and Nezha [9] uses synchronized clocks for deadline-ordered multicast with slow-path fallback. Others move protocol work into the network or specialized hardware: NetPaxos [7] and Consensus in a Box [14] implement Paxos logic in programmable switches or FPGA-based devices, while Mu [1] and CAANS [5] use RDMA or programmable NIC support to reduce replication and agreement cost. Systems such as Waverunner [2] and Partitioned Paxos [6] further split placement across SmartNICs, switches, and hosts. These systems improve throughput and latency, but because they still optimize timeout-driven protocol structures, they do not fundamentally remove the timing variability and long-tail behavior behind conservative failure handling.

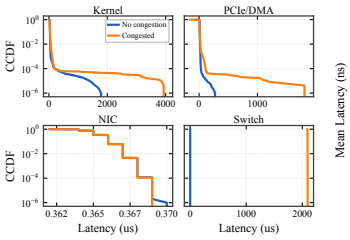

Figure 1: Latency break-down across components.

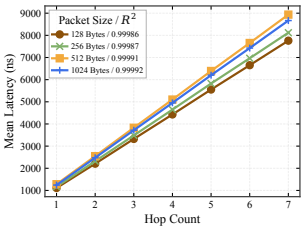

Figure 2: Switch latency versus hop count.

Synchronous SMR. Chora [32] and WatchMaker [27] show that stronger timing assumptions can reshape SMR itself. Chora engineers synchrony through the host/server software path and falls back to timer-driven execution when that discipline fails. WatchMaker explores a quasi-synchronous model in a highly controlled deployment with exclusive replica and clock networks. These systems demonstrate the promise of synchronous SMR, but also show that making the full replication protocol synchronous is difficult in practice: it requires tight control over host-path timing, highly controlled deployments, or both.

Timing measurements. To quantify the timing separation between the network fabric and the host software stack, we build a timestamped testbed with a 6.4 Tbps Tofino switch and eight Xilinx Alveo U200 FPGA SmartNICs running the open-source Corundum NIC [8]. Using port mapping and VLAN isolation, we emulate a logical two-spine, four-leaf topology over 100 Gbps links. We timestamp probe packets across the host software stack, PCIe/DMA path, FPGA NIC pipeline, and switch fabric, and inject controlled incast traffic to compare settings with and without congestion.

Figure 1 decomposes packet latency across four path components, summing the transmit and receive sides where applicable. The host components, kernel processing and PCIe/DMA, show long tails even without congestion due to kernel queueing, interrupt moderation, driver scheduling, and DMA completion effects; congestion further amplifies these tails. In contrast, the FPGA NIC pipeline is clocked and repeatable: its latency stays within roughly two 250 MHz cycles (8 ns) under both settings. Switch latency is tightly concentrated without congestion due to fixed pipeline forwarding; under congestion, output queueing increases latency but leaves the distribution smooth, consistent with prior measurements of switch queueing behavior [16].

The switch behavior in Figure 1 motivates a closer look: without congestion, latency is tightly concentrated, while under incast it grows mainly from output queueing. Figure 2 therefore isolates the uncongested switch fabric. For 128–1024 B packets, mean latency grows almost linearly with hop count; packet size adds a predictable offset but preserves the

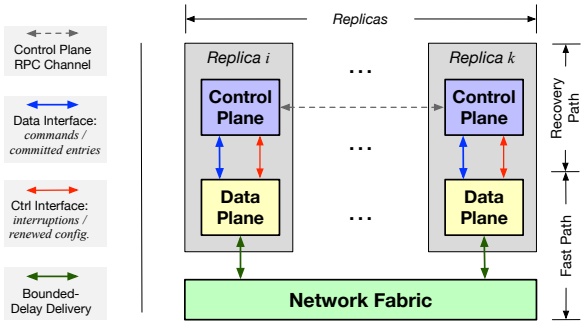

Figure 3: SSR architecture.

linear trend. For all packet sizes, the linear fit yields $R^2 > 0.9998$, where R^2 denotes the coefficient of determination, i.e., the fraction of latency variance explained by hop count. This means that, for a fixed packet size, hop count explains nearly all observed switch-fabric latency variation. Thus, given packet size and topology, fabric latency can be tightly budgeted.

Design implication. Together, the related work and measurements point to a different use of datacenter capabilities. Prior systems either accelerate timeout-driven protocol mechanisms, or pursue synchrony in ways that still depend on host timing discipline or highly controlled deployments. Our measurements show why neither extreme is necessary: tight timing is available where normal-case replication can benefit from it, while the host remains the right place for irregular recovery logic. SSR is built around this principle.

3 SSR Protocol

Figure 3 shows the SSR architecture. Each replica is split into a data plane that implements the synchronous fast path protocol and a control plane for host-side recovery. SSR targets programmable SmartNIC deployments, increasingly common in modern datacenters; our prototype realizes the data plane on FPGA SmartNICs, while the control plane runs in host software. Within a replica, the two planes communicate over standard host-to-NIC mechanisms such as PCIe DMA queues, MMIO control registers, and a host driver: the data interface carries *commands and committed log entries*, while the control interface carries *interruptions and renewed configuration* state. Across replicas, data planes exchange round messages through the network fabric under the bounded-delay delivery assumption, while control planes use an RPC channel for recovery coordination. The rest of this section describes the normal-case fast path and its safety argument, followed by the host-side recovery path.

3.1 Normal-Case Fast Path

The SSR fast path is a round-based synchronous SMR protocol implemented by the replica data planes and connected through the network fabric. Replicas divide time into fixed-length rounds. Within each round, every replica can propose a command concurrently and piggyback evidence about the previous round. At the round boundary, each replica checks whether the collected per-round evidence is sufficient to commit a set of proposals and to justify which replicas may continue in the fast path. If both checks succeed, replicas append the committed proposals in deterministic order and enter the next round; otherwise, the data plane halts and transfers control to the host-side control plane.

Rounds. Replicas use synchronized hardware clocks to align round boundaries. Each fast-path run is configured with a round length Δ, chosen to cover packet delivery through the network fabric, data plane processing at the endpoints, and clock-synchronization error. A message for round r contributes to round-r evidence only if it is received before the round closes; late, stale, or wrong-round messages are ignored. Thus each replica evaluates a finite set of per-round evidence instead of waiting on host-driven timers.

Common-case pipeline. In the common case, every active replica contributes one proposal per round. The round number and replica identifier together determine the log order: proposals from the same round are ordered by replica identifier, and rounds are appended in increasing order. SSR pipelines proposal and acknowledgement. A round-r message carries the sender's proposal for round r together with evidence about round $r-1$, allowing replicas to commit round-$r-1$ proposals while concurrently sending round-r proposals. After the pipeline fills, each round commits the previous round's proposals; each individual proposal commits after two rounds, with no distinguished leader.

Round evidence. A simple quorum acknowledgement is not enough under network faults or asymmetric loss: replicas may collect quorum-sized evidence yet disagree on who can safely remain in the fast path. SSR therefore piggybacks a *sound set* on every proposal. A sound set is not the installed membership; it is a round-by-round survivor set justified by per-round evidence.

Consider a fast path run with installed membership M, where $|M| = N$. At the start of round r, replica i maintains a sound set $S_i^{r-1} \subseteq M$, encoded as a bitmap s_i^{r-1}, derived at the previous round boundary. Replica i sends

$$m_i^r = (run, r, i, p_i^r, s_i^{r-1}),$$

where *run* is the run identifier installed by the control plane and p_i^r is the local proposal for round r. A receiver accepts a message only if the run identifier matches, the round identifier matches, and the sender belongs to M. Messages from

other runs, other rounds, or replicas outside the installed membership are ignored.

At the end of round r, replica i forms a local *sound matrix* A_i^r. Row $A_i^r[j,*]$ is s_j^{r-1} if replica i received a valid message from sender j, and $0^{|M|}$ otherwise. Thus each sender contributes a single row to the matrix at each receiver. Valid nonzero rows must be sender-consistent: a row from j must include bit j. Intuitively, each row records one sender's advertised view, from the previous round boundary, of the replicas that could safely remain in the fast path.

Commit and continuation rules. At the boundary after round r, replica i evaluates the sound matrix collected during round r. It uses its own advertised bitmap s_i^{r-1} as the candidate row, and computes the set of senders that reported exactly this bitmap:

$$\Gamma_i^r = \{\, j \in M \mid A_i^r[j,*] = s_i^{r-1} \,\}.$$

Let $q = \lfloor |M|/2 \rfloor + 1$ be the quorum threshold. Replica i proceeds only if

$$|\Gamma_i^r| \geq q.$$

If this check fails, the data plane halts.

Commit rule. When the local bitmap is quorum-supported, it determines the commit set for round $r-1$:

$$C_i^{r-1} = \{\, j \in M \mid s_i^{r-1}[j] = 1 \,\}.$$

Replica i commits the round-$r-1$ proposals from C_i^{r-1}, ordered deterministically by replica identifier, only if all payloads named by C_i^{r-1} are locally available.

Continuation rule. The senders that support this same bitmap determine the sound set for round r:

$$S_i^r = \Gamma_i^r.$$

Fast path continuation is allowed only by monotone shrinkage:

$$S_i^r \subseteq S_i^{r-1}.$$

Replica i also requires $i \in S_i^r$. If the payload-availability check, self-inclusion check, or shrinkage check fails, the data plane halts and raises an interruption to the control plane.

Thus, the quorum-supported local bitmap determines *what* can be committed from round $r-1$, while its supporters determine *which replicas remain eligible after round* r. This fail-stop rule allows faults to remove replicas from the fast path, but prevents the fast path from expanding or silently forking under ambiguous evidence.

Examples. Before the proof, Figure 4 gives two subtle corner cases that build intuition for the fast path rules. The asymmetric loss case shows that the *sound set may shrink without causing commit divergence*. The split-view partition case shows the complementary behavior: replicas may momentarily derive different sound sets, but the *divergence is exposed in the next round and forces halt* before independent fast path branches can persist.

Safety. We state the fast-path safety argument through two lemmas and a final theorem.

Lemma 3.1 (Same-round commit agreement). *If replicas i and k both accept candidate rows in round r, then they derive the same commit set: $C_i^r = C_k^r$.*

Sketch. Since both replicas accept, their supporter sets Γ_i^r and Γ_k^r each have size at least q. Because $q > |M|/2$, the two sets intersect. Let $j \in \Gamma_i^r \cap \Gamma_k^r$. By definition of support, replica i observed sender j's row as s_i^r, and replica k observed it as s_k^r. Since sender j's row is unique in round r, these rows are identical. Hence $s_i^r = s_k^r$, and therefore $C_i^r = C_k^r$. □

Lemma 3.2 (Next-round exposure of divergent sound sets). *If replicas derive incompatible sound sets after round r, then the disagreement is exposed in round $r+1$, and at least one branch halts before both branches can continue externalizing commands beyond that round.*

Sketch. The sound set derived after round r determines which replicas may send in round $r+1$. If two branches derive incompatible sound sets, the disagreement appears in the round-$r+1$ sound matrices as a missing or mismatched row. Because continuation requires quorum support and monotone shrinkage, at least one branch fails its checks at the next boundary and halts. Thus incompatible branches cannot both continue externalizing commands past round $r+1$. □

Theorem 3.3 (Fast-path safety). *If replicas i and k both continue externalizing commands through round r, then their externalized command sequences through round r are identical.*

Sketch. By Lemma 1, replicas that accept at the same round boundary derive the same commit set for the committed round, and proposals within that set are ordered deterministically by replica identifier. Thus they append the same per-round command sequence whenever they commit. By Lemma 2, if incompatible sound sets arise after some round, the disagreement is exposed in the next round and at least one branch halts before both can continue externalizing commands beyond that point. Therefore two fast-path branches cannot externalize conflicting command sequences across rounds. Any replicas that continue externalizing commands through round r must therefore observe the same sequence of per-round commit sets, and hence the same command sequence. □

3.2 Host-Side Recovery Path

The control plane becomes active when the fast path can no longer continue. Control planes coordinate across replicas over the RPC channel in Figure 3, which carries only lightweight heartbeats in the normal case and becomes the recovery coordination path after an interruption.

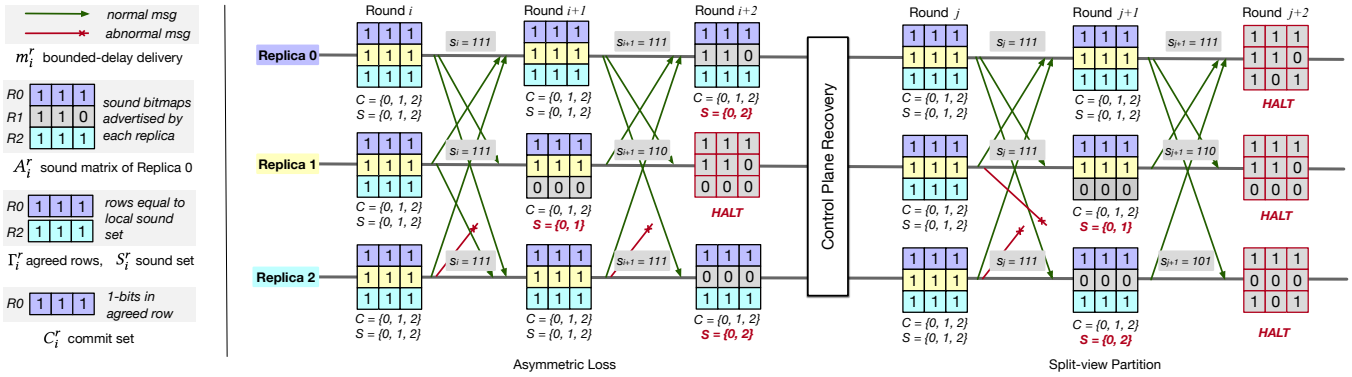

Figure 4: (a) Asymmetric loss: replica 1 cannot observe replica 2, while replica 2 can observe replica 1; replicas 0 and 2 continue with a smaller sound set, while replica 1 halts, without commit divergence. (b) Split-view partition: replicas 1 and 2 cannot observe each other, while replica 0 can observe both; replicas 1 and 2 derive different sound sets with the same commit set, and the divergence is exposed in the next round, causing all replicas to halt.

Recovery protocol. Recovery constructs a fresh fast-path run from a safe committed prefix in two stages. First, the control planes select a recovery coordinator using a standard host-side leader-election protocol [11, 24, 28]. The coordinator gathers each replica's committed prefix and halt status, repairs lagging replicas through state transfer, and proposes a renewed configuration containing a fresh run identifier, membership, and activation round. Second, the coordinator installs this configuration using two-phase commit. Replicas acknowledge PREPARE only after storing the configuration as pending data-plane state, and mark it committed after receiving COMMIT. The data plane activates a committed configuration only at the specified activation round; aborted configurations are discarded.

Recovery may be asynchronous or require retries, but it cannot create conflicting fast-path commits. Because fast-path messages carry run identifiers and replicas accept only messages from their current run, missed or premature configuration changes can at worst prevent quorum formation and force another recovery attempt. Thus safety remains in the data plane, while the control plane restores liveness by repairing state and installing a fresh run.

4 Preliminary Evaluation

We evaluate SSR with a protocol-level simulator[1] that implements the full fast-path rules from §3.1 and the host-side recovery protocol from §3.2. Rather than deriving throughput from a simple formula , the simulator executes replicas round by round, delivering messages under configured timing bounds and injected faults before applying the protocol's commit, continuation, and halt rules. Recovery is event-driven and includes coordinator selection, state transfer, two-phase configuration commit, and fast-path restart. Fast path timing uses the measured round lengths from §2; recovery delays are microsecond-scale estimates informed by contemporary server and NIC behavior [12, 13, 23].

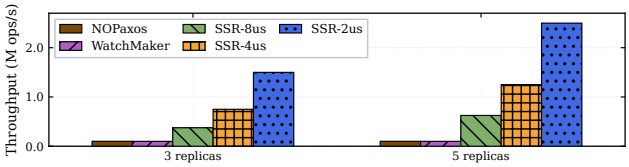

Figure 5: Steady-state throughput comparison.

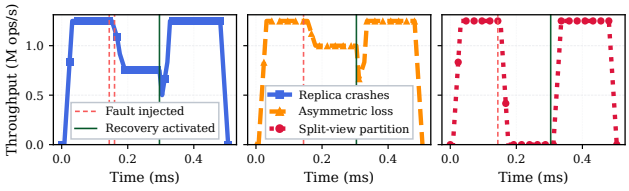

Figure 6: Throughput during failure recovery.

Throughput. We compare SSR with two representative baselines: NOPaxos [20], a classic network-ordering/offload design, and WatchMaker [27], a recent synchronous SMR protocol. For NOPaxos, we implement the Tofino-based sequencer on our measurement testbed, approximating a best-case hardware-sequenced fast path. For WatchMaker, whose deployment relies on a custom OS and dedicated timing infrastructure, we model its reported synchronous timing behavior in our simulator, including its 30 µs bound. Figure 5 shows steady-state throughput for 3- and 5-replica deployments while varying the SSR round length from 8 µs to 2 µs.

[1] https://github.com/yukema/SSR

SSR reaches the highest throughput with 2 µs rounds, improving over the strongest baseline by roughly 14.9× with 3 replicas and 24.9× with 5 replicas. Throughput rises as the round length decreases because each round completes sooner. SSR gains throughput scaling from its leaderless design, which lets replicas propose concurrently as the replica count grows; WatchMaker does not provide the same scaling behavior.

Failure recovery. Figure 6 shows throughput during recovery for a 5-replica SSR deployment with a 4 µs round length. We inject replica crashes, asymmetric loss, and split-view partition. In all cases, the fast path exposes unsafe conditions within a few rounds and completes host-side recovery in about 0.15 ms. Throughput degrades according to the fault pattern: crashes remove proposing replicas in stages, asymmetric loss shrinks the sound set and preserves partial throughput, and split-view partition forces all replicas to halt until recovery installs a renewed configuration. Thus SSR is conservative under ambiguous evidence, but preserves throughput when a smaller sound set can safely continue.

Fast path latency. SSR's protocol latency is fixed by the round structure. In the normal case, each proposal commits after two fast-path rounds, i.e., 4 μs with a 2 μs round length. End-to-end latency additionally includes host-to-NIC command injection and committed-log delivery overheads, such as PCIe, DMA, driver, and kernel processing. These overheads depend on the implementation and are not explicitly modeled here. Thus, our evaluation reports the protocol-level fast-path latency; a complete implementation would add the host-side delivery cost, which can be optimized independently of the SSR protocol.

Use of AI tools. We used OpenAI Codex to assist in generating test code for validating the simulator and plotting scripts for visualizing evaluation results. The protocol design, simulator implementation, experimental methodology, and data generation were performed by the authors. All AI-assisted code was reviewed and validated by the authors.

5 Discussion and Future Work

SSR shows that synchronous SMR can be made practical by splitting the protocol path: a round-based fast path exploits tight timing, while recovery remains in the host-side control plane. Our current implementation is a protocol-level simulator that executes the fast path and event-driven recovery protocol, allowing us to evaluate throughput, protocol-level latency, and recovery activation under controlled timing and fault scenarios, but not the full host-to-NIC data path.

SSR targets capabilities increasingly common in modern datacenters: programmable NICs, precise hardware clocks, and fast switching fabrics. We are actively building a full-system FPGA SmartNIC implementation of SSR, with ongoing development released alongside the simulator in the same repository. This prototype will validate SSR's end-to-end performance under realistic datacenter workloads and operating conditions.

References

[1] Marcos K. Aguilera, Naama Ben-David, Rachid Guerraoui, Virendra J. Marathe, Athanasios Xygkis, and Igor Zablotchi. 2020. Microsecond Consensus for Microsecond Applications. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX Association, Virtual Event, 599–616.

[2] Mohammadreza Alimadadi, Hieu Mai, Shenghsun Cho, Michael Ferdman, Peter Milder, and Shuai Mu. 2023. Waverunner: An Elegant Approach to Hardware Acceleration of State Machine Replication. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX Association, Boston, MA, 357–374.

[3] Inho Choi, Ellis Michael, Yunfan Li, Dan R. K. Ports, and Jialin Li. 2023. Hydra: Serialization-Free Network Ordering for Strongly Consistent Distributed Applications. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX Association, Boston, MA, USA, 293–320.

[4] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, JJ Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. 2012. Spanner: Google's Globally-Distributed Database. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX Association, Hollywood, CA, 261–264.

[5] Huynh Tu Dang, Pietro Bressana, Han Wang, Ki Suh Lee, Hakim Weatherspoon, Marco Canini, Fernando Pedone, and Robert Soulé. 2016. Network Hardware-Accelerated Consensus. arXiv:1605.05619 [cs.DC] https://arxiv.org/abs/1605.05619

[6] Huynh Tu Dang, Pietro Bressana, Han Wang, Ki Suh Lee, Noa Zilberman, Hakim Weatherspoon, Marco Canini, Fernando Pedone, and Robert Soulé. 2019. Partitioned Paxos via the Network Data Plane. arXiv:1901.08806 [cs.DC] https://arxiv.org/abs/1901.08806

[7] Huynh Tu Dang, Daniele Sciascia, Marco Canini, Fernando Pedone, and Robert Soulé. 2015. NetPaxos: consensus at network speed. In *Proceedings of the ACM SIGCOMM Symposium on Software Defined Networking Research SOSR*. ACM, Santa Clara, CA, USA, 5:1–5:7.

[8] Alex Forencich, Alex C. Snoeren, George Porter, and George Papen. 2020. Corundum: An Open-Source 100-Gbps NIC. In *IEEE International Symposium on Field-Programmable Custom Computing Machines*.

[9] Jinkun Geng, Anirudh Sivaraman, Balaji Prabhakar, and Mendel Rosenblum. 2022. Nezha: Deployable and High-Performance Consensus Using Synchronized Clocks. *Proc. VLDB Endow.* 16, 4 (2022), 629–642.

[10] Yilong Geng, Shiyu Liu, Zi Yin, Ashish Naik, Balaji Prabhakar, Mendel Rosenblum, and Amin Vahdat. 2018. Exploiting a Natural Network Effect for Scalable, Fine-grained Clock Synchronization. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX Association, Renton, WA, 81–94.

[11] Patrick Hunt, Mahadev Konar, Flavio P. Junqueira, and Benjamin Reed. 2010. ZooKeeper: wait-free coordination for internet-scale systems. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC)*. USENIX Association, Boston, MA, 11.

[12] Intel. 2024. DMA Coalescing. https://www.intel.com/content/www/us/en/support/articles/000007456/ethernet-products.html. Accessed: 2026-05-14.

[13] Intel. 2024. Interrupt Moderation. https://edc.intel.com/. Intel Ethernet 700 Series Linux Performance Tuning Guide, "Interrupt Moderation". Accessed: 2026-05-14.

[14] Zsolt István, David Sidler, Gustavo Alonso, and Marko Vukolic. 2016. Consensus in a Box: Inexpensive Coordination in Hardware. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation, (NSDI)*. USENIX Association, Santa Clara, CA, USA, 425–438.

[15] Xin Jin, Xiaozhou Li, Haoyu Zhang, Nate Foster, Jeongkeun Lee, Robert Soulé, Changhoon Kim, and Ion Stoica. 2018. Netchain: scale-free sub-RTT coordination. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation (NSDI)*. USENIX Association, Renton, WA, USA, 35–49.

[16] Pravein Govindan Kannan, Raj Joshi, and Mun Choon Chan. 2019. Precise Time-synchronization in the Data-Plane using Programmable Switching ASICs. In *Proceedings of the ACM Symposium on SDN Research (SOSR)*. ACM, San Jose, CA, USA, 8–20.

[17] Leslie Lamport. 1998. The Part-Time Parliament. *ACM Transactions on Computer Systems* 16, 2 (1998), 133–169.

[18] Yiming Lei, Jialong Li, Zhengqing Liu, Raj Joshi, and Yiting Xia. 2026. SyncWise: Error-Aware Time Synchronization for Reconfigurable Data Center Networks. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX Association, Renton, WA, 951–967.

[19] Bojie Li, Kun Tan, Layong (Larry) Luo, Yanqing Peng, Renqian Luo, Ningyi Xu, Yongqiang Xiong, Peng Cheng, and Enhong Chen. 2016. ClickNP: Highly Flexible and High Performance Network Processing with Reconfigurable Hardware. In *Proceedings of the ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*. ACM, Florianopolis, Brazil, 1–14.

[20] Jialin Li, Ellis Michael, Naveen Kr. Sharma, Adriana Szekeres, and Dan R. K. Ports. 2016. Just Say NO to Paxos Overhead: Replacing Consensus with Network Ordering. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX Association, Savannah, GA, USA, 467–483.

[21] Yuliang Li, Gautam Kumar, Hema Hariharan, Hassan Wassel, Peter Hochschild, Dave Platt, Simon Sabato, Minlan Yu, Nandita Dukkipati, Prashant Chandra, and Amin Vahdat. 2020. Sundial: fault-tolerant clock synchronization for datacenters. In *Proceedings of the USENIX Conference on Operating Systems Design and Implementation (OSDI)*. USENIX Association, USA.

[22] Yu-Shan Lin, Shao-Kan Pi, Meng-Kai Liao, Ching Tsai, Aaron Elmore, and Shan-Hung Wu. 2019. MgCrab: transaction crabbing for live migration in deterministic database systems. *Proceedings of the VLDB Endowment* 12, 5 (2019), 597–610.

[23] Linux Kernel Documentation. 2024. NAPI. https://docs.kernel.org/networking/napi.html. Accessed: 2026-05-14.

[24] Barbara Liskov and James Cowling. 2012. Viewstamped replication revisited. (2012).

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

[26] Zhengqing Liu, Musa Unal, Matthew J. Parkinson, and Marios Kogias. 2025. DORADD: Deterministic Parallel Execution in the Era of Microsecond-Scale Computing. In *Proceedings of the ACM Annual Symposium on Principles and Practice of Parallel Programming (PPoPP)*. ACM, Las Vegas, NV, USA, 282–296.

[27] Karan Newatia, Robert Gifford, Qingjie Lu, Andreas Haeberlen, and Linh Thi Xuan Phan. 2026. Running Distributed Systems like Clockwork. In *Proceedings of the New Ideas in Networked Systems, NINeS (OASIcs)*. Virtual Conference, 26:1–26:31.

[28] Diego Ongaro and John Ousterhout. 2014. In Search of an Understandable Consensus Algorithm. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC)*. USENIX Association, Philadelphia, PA, 305–319.

[29] Phitchaya Mangpo Phothilimthana, Ming Liu, Antoine Kaufmann, Simon Peter, Rastislav Bodik, and Thomas Anderson. 2018. Floem: A Programming System for NIC-Accelerated Network Applications. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX Association, Carlsbad, CA, 663–679.

[30] Dan R. K. Ports, Jialin Li, Vincent Liu, Naveen Kr. Sharma, and Arvind Krishnamurthy. 2015. Designing Distributed Systems Using Approximate Synchrony in Data Center Networks. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX Association, Oakland, CA, USA, 43–57.

[31] Alexander Thomson, Thaddeus Diamond, Shu-Chun Weng, Kun Ren, Philip Shao, and Daniel J. Abadi. 2012. Calvin: fast distributed transactions for partitioned database systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Association for Computing Machinery, New York, NY, USA, 1–12.

[32] Yiliang Wan, Nitin Shivaraman, Akshaye Shenoi, Xiang Liu, Tao Luo, and Jialin Li. 2025. Building State Machine Replication Using Practical Network Synchrony. arXiv:2507.12792 [cs.DC] https://arxiv.org/abs/2507.12792

[33] Irene Zhang, Amanda Raybuck, Pratyush Patel, Kirk Olynyk, Jacob Nelson, Omar S. Navarro Leija, Ashlie Martinez, Jing Liu, Anna Kornfeld Simpson, Sujay Jayakar, Pedro Henrique Penna, Max Demoulin, Piali Choudhury, and Anirudh Badam. 2021. The Demikernel Datapath OS Architecture for Microsecond-scale Datacenter Systems. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*. ACM, Virtual Event, Germany, 195–211.

[34] Hang Zhu, Zhihao Bai, Jialin Li, Ellis Michael, Dan R. K. Ports, Ion Stoica, and Xin Jin. 2019. Harmonia: near-linear scalability for replicated storage with in-network conflict detection. *Proc. VLDB Endow.* 13, 3 (2019), 376–389.