SyncWise: Error-Aware Time Synchronization for Reconfigurable Data Center Networks

Yiming Lei
Max Planck Institute for Informatics

Jialong Li*
Shenzhen University of Advanced Technology

Zhengqing Liu*
Imperial College London

Raj Joshi
Harvard University

Yiting Xia
Max Planck Institute for Informatics

Abstract

Time synchronization is critical for emerging reconfigurable data center networks (RDCNs) built on optical circuit switches (OCSes), where accurate identification of dynamic optical circuits is required for reliable data transmission. As circuit durations shrink to the microsecond and sub-microsecond range, sync accuracy demands become increasingly stringent. Yet, this problem has been largely overlooked, both practically and theoretically, with no general, easily integrable solutions or established accuracy limits. We present SyncWise, the first sync protocol customized for RDCNs that closes these gaps. SyncWise formalizes the synchronization problem for RDCNs, models errors through an RDCN-specific analysis of error sources, and leverages this error model to guide the "wise" selection of optimal sync parents. SyncWise achieves provably optimal performance and is prototyped on a testbed with an OCS and Intel Tofino2 switches. Large-scale simulations demonstrate that SyncWise is the first protocol to attain sub-10 ns maximum sync error, ensuring reliable RDCN fabrics and outperforming the most accurate protocols in traditional DCNs by a large margin.

1 Introduction

With the slowdown of Moore's Law in merchant silicon, reconfigurable data center networks (RDCNs) based on optical circuit switches (OCSes) are arising as the solution to high-speed, scalable, economical, and energy-efficient cloud infrastructure [4, 7–9, 14, 18, 24, 25, 27, 31, 32, 35–38]. In an RDCN (Fig. 1), a (logically) centralized optical controller reconfigures OCSes to establish dynamic optical circuits interconnecting electrical endpoints such as pods, top-of-rack (ToR) switches, or host NICs. Precise synchronization between the endpoints and the controller is thus a must to align data transmission with circuit availability. This fundamental departure from traditional data center networks (DCNs) makes time synchronization not merely a requirement of cloud applications, but a prerequisite for the correct functioning of routing.

*Work done while at Max Planck Institute for Informatics.

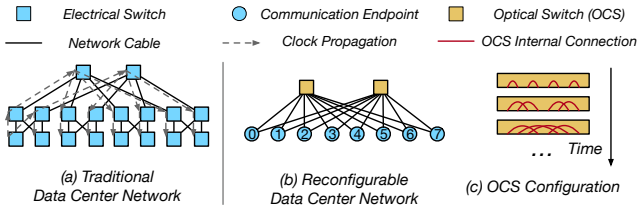


Figure 1: A traditional static DCN vs. an RDCN.

Advances in OCS technologies have enabled increasingly frequent reconfigurations, reducing circuit duration to microseconds or even sub-microseconds to minimize circuit-waiting delay [4–6, 10, 13, 16, 27, 30]. This trend imposes ever more stringent synchronization requirements: synchronization error is required to be roughly two orders of magnitude smaller than the circuit duration to reliably identify circuit boundaries and maintain a high duty cycle, with the error protected by a guardband during reconfiguration when data transmission is prohibited [4].

Despite its critical role, time synchronization remains under explored in RDCNs. Most architectures take it as a given and omit it from system design, demonstrating functionality with a shared clock. Those that do address it rely on narrow, architecture-specific solutions: some use out-of-band synchronization over a low-speed management network, providing microsecond-level precision—sufficient for the targeted architectures but inadequate for today's shrinking circuit durations [15, 25, 27]; others develop dedicated schemes tied to properties of specific optical hardware, which do not generalize across architectures [4]. This reliance on case-by-case synchronization mechanisms adds significant complexity to RDCN design, and the lack of an easily integrable solution hampers implementation and broader adoption of RDCNs.

In this paper, we raise a fundamental question: is it possible to develop a *general* time synchronization solution that is *applicable* and *easily integrable* across diverse RDCN architectures? With nanosecond-precision synchronization already commonplace in production DCNs [17, 20, 22, 28], we further ask whether *comparable*—or even *superior*—accuracy can be achieved in RDCNs, and what the *accuracy limit* is.

To answer these questions, we present *SyncWise*, an accurate, architecture-independent synchronization solution for RDCNs. Its design faces two essential challenges arising from the unique characteristics of RDCNs.

Dynamic topology. A general and easily integrable solution must not rely on support beyond the RDCN fabric—such as assuming a parallel management network, specific OCS types, or non-standard sync protocols that need specialized optical transceivers —since these are not guaranteed to be available in a given architecture. This necessitates *in-band synchronization* that operates solely over the RDCN fabric. Sync protocols for traditional DCNs assume a static, always-connected network for continual exchange of sync messages [12, 22, 29], albeit RDCN fabrics connect endpoint pairs only in discrete time periods. As each endpoint exposes a small number of optical uplinks, e.g., $\leq$2 NIC ports [2, 4] or only a handful (typically $<$10) switch ports to contain cost [25, 27], the instantaneous topology is sparse. Consequently, the transient optical circuits fundamentally limit *when* and *where* sync messages can be sent.

Strict and known error bound. Sync accuracy is typically assessed only after a sync protocol is deployed. Tail error is often characterized at the 99$^{\text{th}}$ or 99.9$^{\text{th}}$ percentile, as cloud applications generally tolerate occasional outlier errors. In RDCNs, however, the *absolute worst-case error*, or *error bound* (i.e., the 100$^{\text{th}}$ percentile), directly determines the minimum guardband size, which in turn constrains achievable network utilization. For example, a conservative guardband set to 10% of the slice duration results in a 10% network utilization reduction. Conversely, configuring a guardband smaller than the actual synchronization error bound causes persistent packet loss and severely degrades network throughput. Therefore, minimizing the synchronization error bound and accurately deriving it for guardband configuration are critical for maximizing RDCN performance. Moreover, this bound must be known *a priori*, before the network becomes operational; otherwise, circuits may be misidentified, leading to routing failures and an unreliable network fabric. Critically, this guarantee cannot rely on empirical measurement alone but requires *theoretical grounding* to ensure correctness under all conditions.

In SyncWise, we propose *error-aware synchronization* to address these challenges, exploiting the temporal diversity of RDCN connectivity to maximize accuracy. It is widely established that clock quality is heterogeneous [28, 29]. At any instant the RDCN topology is sparse—each endpoint has only a few candidate sync parents—but reconfiguration over time exposes many peers with differing clock quality. Nodes must therefore *synchronize wisely*: select high-quality parents and synchronize only when advantageous, since settling for a poor clock can be worse than deferring to a later opportunity.

Recent studies revealed that synchronization error is largely predictable: device clocks exhibit drift with a static expectation and a variance [28], and per-hop synchronization artifacts are characterizable [29]. Reconfigurability enables, prior to operation, direct connections between arbitrary endpoint pairs via circuit permutations, allowing accurate profiling of pairwise error parameters and, in turn, reliable error prediction.

Following this insight, we formalize the time synchronization problem for RDCNs (§2) and derive error bounds by analyzing the error sources (§2.2). Using these bounds, we design the SyncWise error-(bound)-aware sync protocol, where the endpoints predict their error bounds to opportunistically select sync parents (§3.2). We further mitigate synchronization artifacts (§3.4) and tailor the sync procedure to the ultra-short circuit durations in the latest RDCNs (§3.3). SyncWise's error bound is provably minimal and computable *a priori* for a given optical circuit schedule (§3.5).

We have implemented SyncWise and evaluated it on a hardware testbed with an OCS and four Intel Tofino2 switches, and via simulation of a 108-ToR RDCN. The protocol logic is lightweight and hardware-agnostic, easily portable to NICs and FPGAs. SyncWise achieves 9 ns maximum sync error and 15 ns theoretical error bound, sufficient for RDCN designs with circuit durations of only a few microseconds, demonstrating the feasibility of an architecture-independent synchronization solution. Thanks to its error-aware design, SyncWise on RDCNs attains sync accuracy comparable to, and often better than, state-of-the-art protocols on static DCNs: the maximum error is only 39.9% of Firefly [29], 16.0% of Sundial [22], 8.9% of DTP [21] and 26.8% of Graham [28].

We have open-sourced SyncWise and its simulator[1] to support further research. Since SyncWise yields an optimal, *a priori* computable sync error bound given an optical circuit schedule and profiled error statistics, RDCN designers can use the simulator to determine the minimum safe circuit duration supported by this general approach; shorter durations would require specialized synchronization.

[This work does not raise any ethical issues.]

2 Problem Formulation

In this section, we define the new problem of time synchronization in RDCNs. We first outline how RDCNs function (§2.1), then derive sync error bound by decomposing error sources (§2.2), and finally evaluate the applicability of existing synchronization schemes to RDCNs, motivating the need for error-aware synchronization in SyncWise.

2.1 Reconfigurable Data Center Networks

As illustrated in Fig. 1b, an RDCN contains a number of (two in the figure) OCSes in the network fabric. The OCSes connect the electrical endpoint nodes and provide dedicated optical circuits between node pairs. Once established, an optical circuit is retained for a fixed interval of time, called a *time slice*, during which the connected nodes have exclusive use of it, namely no contention with other nodes. An *op-*

[1] https://github.com/mpi-ncs/SyncWise

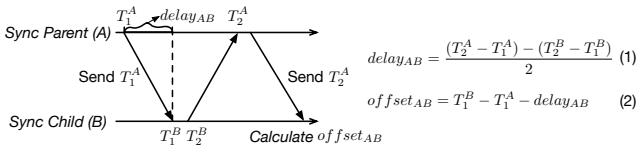


Figure 2: Message exchanges to synchronize two clocks.

tical controller[2], typically on a host machine or an FPGA board, controls the OCSes to reconfigure the optical circuits continually, once per time slice. The sequence of node-wise connections associated with their time slices constitutes a *circuit schedule*.

RDCNs can be broadly classified as *traffic-aware* or *traffic-oblivious*. Traffic-aware architectures [7, 14, 18, 32, 35] dynamically reconfigure circuits to fit instantaneous traffic patterns. A network controller collects traffic demands via an out-of-band management network and computes the network topology centrally. This control loop takes milliseconds to seconds, limiting time slices to similar or longer durations. At these timescales, synchronization is straightforward: the network controller notifies endpoints of upcoming circuits via control messages over the management network, and no specialized high-precision sync protocol is required.

Traffic-oblivious designs [2,4,25,27] cycle through a fixed circuit schedule independent of instantaneous demand. They target microsecond- to sub-microsecond–scale slices, making the time-consuming control loop infeasible. Endpoints must therefore be tightly synchronized with the optical controller and self-regulate transmissions according to the preloaded circuit schedule. A *semi-oblivious* variant [33] was proposed recently, which still follows a circuit schedule but allows schedule updates to reflect evolution of traffic characteristics.

Design scope. SyncWise is designed for traffic-oblivious RDCNs with stringent synchronization requirements. It also supports semi-oblivious RDCNs (§7.5). Traffic-aware RDCNs typically require much lower synchronization accuracy due to millisecond-level (or longer) control loops, and centralized circuit reconfiguration eliminates the need for prior knowledge of error bounds. SyncWise therefore readily applies to this more relaxed setting. As further discussed in §5, the benefits of error-aware synchronization further extend to static DCNs.

2.2 Synchronization Error Bound

The sync problem in RDCNs. In an RDCN, it is critical to align data transmission with the circuit schedule to prevent packet loss in the fabric. The optical controller that drives reconfiguration maintains the *reference clock*, so the problem reduces to synchronizing all endpoints with the controller. In practice, the controller (a host or FPGA) attaches to an endpoint (host or switch) designated as the *reference node*. The problem is thus to synchronize all endpoints to this reference. In switch-centric RDCNs, whose endpoints are ToRs or pod switches, hosts can—if needed—be further synchronized over static host-switch links trivially using standard protocols (e.g., PTP [12]). Accordingly, in this paper, we focus on endpoint-reference synchronization within the RDCN fabric.

The sync operation. Most sync schemes rely on the exchange of sync messages to synchronize the clock of a child node with that of a parent node. Fig. 2 presents this base operation, which involves three sync messages [12, 20, 22]. The first message is for *clock propagation*, where the sync parent A takes the Transmit (TX) timestamp of the message T_1^A as its clock and sends it to the child node B. B notes down the Receive (RX) timestamp of the message T_1^B and immediately sends back another message, whose TX timestamp is T_2^B. Node A captures the RX timestamp of getting the second message T_2^A and sends it back to B in the third message. The second and third messages are for *measurement of the propagation delay* between A and B $delay_{AB}$. After collecting the four timestamps, node B can calculate $delay_{AB}$ according to Eqn. 1 in Fig. 2. The clock offset of B, or how much it should adjust its clock relative to A, can be calculated from Eqn. 2. The offset is the difference between B's clock (T_1^B) and A's clock compensated by the propagation delay ($T_1^A + delay_{AB}$).

Error sources. Since only a limited number of nodes can connect directly to the reference, the reference clock must propagate hop-by-hop through intermediate nodes to the remaining nodes. The post-synchronization clock error of a non-reference node (Eqn. 3) has two components: *clock drift* and *hop error*.

$$Error = \sum_{i=0}^{l} \int_0^{\tau_i} drift_i(t)dt + \sum_{i=1}^{l} h_{i,i-1} \tag{3}$$

If a clock has drift rate $drift(t)$ at time t, its drift over elapsed time τ since the last synchronization is $\int_0^{\tau} drift(t) \cdot dt$. For a node at hop distance l from the reference, its drift (first term in Eqn. 3) is the cumulative drift of all upstream clocks inherited hop-by-hop, each since their last synchronization.

Hop error is the per-hop noise introduced by factors such as inaccurate device timestamping and bidirectional link delay asymmetry [17, 29]. The hop error for a node l hops from the reference (second term in Eqn. 3) is the sum of the per-hop noise $h_{i,i-1}$ with respect to each previous hop along the path.

Drift modeling. Prior studies have shown that drift varies across devices and temperature. At a given temperature, a clock's drift rates cluster around a central value, on top of which lies high-frequency fluctuations, e.g., ± 10 ppm[3], observable only at fine time scales (sub-ms) [20, 28], but negligible when averaged over coarser intervals (e.g., seconds) [29].

We model drift as the sum of a static expectation D and a variance v, i.e., $drift(t) = D + v(t)$. This allows nodes to estimate and compensate for D to mitigate drift. The resulting clock error with drift compensation is as follows.

[2] SyncWise can be adapted to controller-free architectures like Sirius [4], as discussed in §5.

[3] Parts per million. 1 ppm corresponds to 1 ns drift per ms.

$$Error = \sum_{i=0}^{l} \int_{0}^{\tau_i} v_i(t)dt + \sum_{i=1}^{l} h_{i,i-1} \quad (4)$$

Hop error modeling. Hop error stems from three sources [29]. (1) *Path asymmetry* arises from unequal forward and reverse paths (e.g., differing hop counts or fiber lengths). (2) *Delay asymmetry* refers to fixed delay offsets introduced by factors such as optical transceiver DSP design, switch ASIC layout, or the distance between the NIC's oscillator and its MAC/PHY timestamping point. (3) *Timestamping noise* is jitter due to NIC clock inaccuracy, whose range depends on the NIC hardware model. In RDCNs, if nodes synchronize only with direct neighbors, path asymmetry is minimal. We mitigate delay asymmetry as described in §3.4.

Error bound. As established in §1, the worst-case sync error dictates the minimum time slice duration and guardband size, which must be known *before* an RDCN enters operation. Because the instantaneous error in Eqn. 4 is not knowable a priori, we derive an upper bound as below.

$$Error\ Bound = \sum_{i=0}^{l} V_i \times \tau_i + H \times l \quad (5)$$

Drift is device-specific [28, 29], so we model each node with a drift variance bound V_i, assuming its static expectation D_i is compensated. Exploiting RDCN circuit reconfigurability, we profile V_i pre-deployment by sweeping circuit connections between each endpoint and the reference (§3.2).

Hop error is port-pair specific between connected nodes. Directly measuring it across devices without a shared clock is difficult. Instead, we configure OCSes to create loopback links between NIC ports[4] within each node, allowing direct measurement of the node's timestamp noise range—the dominant contributor to hop error. Since RDCN nodes are typically homogeneous[5], we model the hop error bound H network-wide as the maximum hop error observed across all nodes.

The path length l and elapsed time τ_i are tracked during clock propagation. SyncWise then uses these error parameters to opportunistically choose sync parents and minimize the error bound per node, and hereby network-wide (§3.2).

2.3 Adaptability of Existing Solutions

Despite rich studies, existing DCN sync schemes fall short of our goal: a general, easily integrable solution for RDCNs with a tight pre-computable error bound. We assess their suitability along the dimensions in Table 1 to inform our design.

Tree-based protocols. Classic solutions like PTP [12] and its variants [20, 23] build a static clock propagation tree (e.g., spanning tree as in Fig. 1a) to distribute the reference clock. The tree limits synchronization hops and avoids loops, preventing hop error from accumulating indefinitely. Drift is mitigated through continuous synchronization along the tree. Sundial further employs primary/backup trees to improve fault tolerance and increases sync frequency to reduce drift [22]. While effective on static networks, this method is ill-suited for RDCNs, as maintaining consistent, loop-free trees over the constantly changing circuits is extremely challenging.

Gossip algorithms. Some DCN schemes draw on gossip-style synchronization from wireless networks: nodes synchronize with many peers to mitigate drift variance and hop errors. Firefly randomly selects a large number of peers per synchronization to average out their errors [29]. Huygens collects comprehensive error data through pairwise synchronization and uses SVM learning to infer each node's true time [17]. DTP continuously floods all clocks to the network, driving convergence toward the highest-frequency clock [21].

Most such solutions target internal synchronization without a reference clock, where nodes converge to a common but floating "swarm virtual clock". However, RDCNs require the optical controller to maintain a stable, absolute clock to deterministically orchestrate OCS reconfiguration, ensuring consistent time slice durations. While some approaches also synchronize the group with an external reference clock [21, 29], this is best-effort at microsecond-level accuracy, insufficient for the fine time scales of modern RDCNs.

Besides, as the RDCN topology per time slice is sparse (§1), synchronizing with a few direct neighbors is statistically insufficient for error cancellation. Expanding the peer set to nonadjacent nodes in an RDCN forces sync messages to either traverse multi-hop routes, accumulating errors from intermediate nodes, or wait for direct circuits, accruing additional drift—both breaking the mathematical models underpinning these gossip algorithms. For instance, Huygens' path-specific error data becomes irrelevant if applied to the constantly changing paths of an RDCN.

Drift compensation. Graham [28], building on the drift characterization in §2.2, compensates each device's drift expectation locally, only synchronizing infrequently to reduce the sync overhead and improve failure resilience. It provides microsecond-scale accuracy and is commonly used as an auxiliary to sync protocols, e.g., Huygens [17] and Firefly [29], to further reduce drift. Since Graham is topology-agnostic, it can extend to RDCNs. Yet, it is unclear how to adapt it to synchronize over RDCN's dynamic topologies. We adapt Graham to RDCNs preserving its gist of infrequent synchronization, synchronizing each node with the reference only when a direct circuit appears, and use it as a comparison baseline.

Hardware specialized solutions. OCSes are physical-layer devices and cannot execute sync protocols. Consequently, time synchronization must be performed at electrical endpoints. Sirius [4] uses syntonization to extract clock from data streams and relies on tunable lasers to cooperate with AWGR-based OCSes. White-Rabbit [23] likewise requires

[4] If an endpoint has only a single optical uplink, it can be looped back to itself, since commercial OCSes typically use unidirectional links.

[5] For heterogeneous deployments with different NIC models, we estimate port-pair-specific hop error by averaging the measured timestamp noise from both sides, reflecting the combined effect of both NICs.

Table 1: Characteristics of synchronization protocols.

Metric	PTP	Sundial	Hyugens	DTP	Firefly	Graham	Sirius	SyncWise
Work on RDCNs	No	No	No	No	No	Adaptable	Yes	**Yes**
Error Bound	Hop Error	Hop Error + Drift (Loose)	No	Hop Error	Hop Error	No	Hop Error	**Hop Error + Drift**
Hardware Independent	Yes	No	Yes	No	Yes	Yes	No	**Yes**

physical-layer syntonization and, being PTP-based, relies on a static clock propagation tree, thus not adaptable to RDCNs. DTP also relies on specialized NIC ASIC support for high sync frequency in the physical layer [21]. In theory, vendors can produce specialized hardware to support synchronization in RDCNs. However, such solutions are typically architecture-specific. For example, Sirius' tunable-laser-based syntonization mechanism does not apply to RotorNet [26]. Moreover, RDCNs remain an emerging design space under active exploration, and vendors have not yet converged on a dominant architecture. Consequently, there are currently no concrete plans to integrate such architecture-specific synchronization mechanisms into transceivers or switches, limiting ongoing studies and implementation of RDCNs. Our goal to develop a general, hardware-independent synchronization solution is to overcome these problems.

Error bound. Tree-based schemes, such as PTP [12], bound hop error by the tree depth; continuous synchronization only mitigates drift but provides no drift bound. Sundial further gives a loose drift bound of 200 ppm with a conservative drift model [22]. The Firefly [29] gossip algorithm applies drift compensation and assumes fluctuations in drift and hop error cancel out across many peers, yielding a hop error bound considering zero remaining drift. DTP [21], through hardware-enabled frequent synchronization (e.g., every 6.4 ns) captures the instantaneous drift rate and provides a hop error bound, assuming drift effectively eliminated by this method. In contrast, SyncWise provides a complete error bound that explicitly accounts for both clock drift and hop error (§2.2) in RDCNs. This bound formulation guides our protocol's design to minimize the collective effects of error factors, thereby achieving a bound-optimal synchronization solution for RDCNs (§3).

3 SyncWise Design

Based on our formulation (§2), we present SyncWise tailored to the synchronization requirements of RDCNs. We first introduce the error-aware sync protocol minimizing the sync error bound (§3.2). Next, we ensure correctness by preventing sync operations from overlapping with topology changes (§3.3). We then exploit circuit reconfigurability to reduce hop errors, a key contributor to the bound (§3.4). Finally, we prove that SyncWise achieves the minimal, pre-computable error bound.

3.1 Error Parameter Profiling

As described in §2.2, prior to RDCN operation we leverage circuit reconfigurability to profile three critical error parameters: each node's *drift expectation* and *maximum drift variance*, and the *maximum hop error* across the network.

Drift profiling. The optical controller reconfigures the OCSes to sequentially connect each endpoint to the master node before RDCN operation. Each connection is maintained for a pre-configured duration sufficient to complete n repeated sync operations, one per sync interval T_{sync}, as illustrated in Fig. 2. Drift between consecutive sync requests is computed following the standard method [20]. The drift over a period of time is calculated as the sync result difference between two sync operations. The drift rate is then obtained by dividing this value by the sync interval. The drift expectation is taken as the median across the measurements, and the maximum variance is the largest absolute deviation from this mean. Both values are measured and stored locally at each device.

Hop error profiling. We follow the method in §2.2 to sequentially connect each endpoint's port combinations and perform repeated sync operations on the same device. Since the sync parent and child are the same device and thus share a clock, the measured sync error reflects only the hop error, with no drift. Each node records its maximum hop error, which is then collected at the master to determine the global maximum and redistributed to all devices. This collection and dissemination are carried out through two additional rounds of sequential connections between each endpoint and the master.

Propagation delay profiling. Propagation delays need to be profiled per node port pair due to different cable lengths. Once again, we leverage the circuit reconfigurability of optical DCNs to connect every pair of ToR ports, and we measure their propagation delays following Eqn. 1 in Fig. 2. Similar to drift profiling, we collect n data points and take the median.

3.2 Error-Aware Synchronization Protocol

Alg.1 shows the SyncWise sync algorithm executed on every endpoint[6]. Each node maintains its current error bound, defined in Eqn. 5, using the pre-profiled error parameters.

After profiling, each node executes the sync (main) procedure in Alg. 1. The reference $Node_0$ holds the reference clock, giving it zero sync error and error bound ($B_0 = 0$); all other nodes initialize their error bounds to $B_i = \infty$ (*Line 2*). Then the main process spawns two parallel threads to *send* (*Line 3*) and *recv* (*Line 4*) sync messages. In the *send* thread, every sync interval (*Line 8*), the node compensates for its expected drift (*Line 9*), and adjusts its bound to account for the uncompensatable drift variance (*Line 10*). It then broadcasts its current clock value and error bound to all directly connected neighbors (*Line 11*). Broadcasting avoids the need

[6]The simple algorithm logic can be implemented in either hardware or software, depending on accuracy requirements. For example, if the topology changes on the order of 100 ns, sync messages must be sent with sub-100 ns accuracy where a software-based implementation would be insufficient, and a hardware implementation of Alg. 1 would be required.

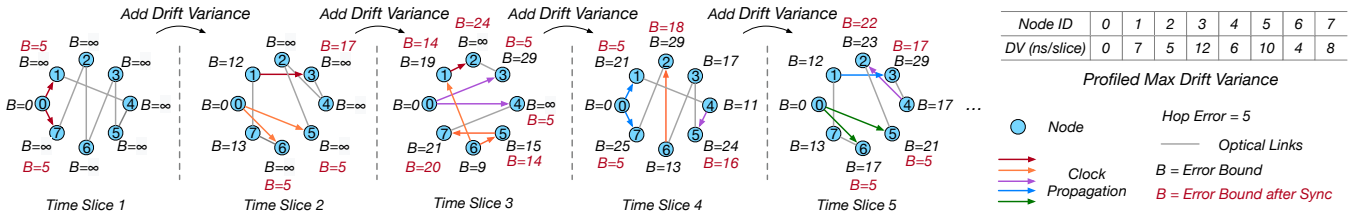


Figure 3: A example of Alg. 1, where the sync interval is one time slice. Arrows of each color represents a clock propagation tree rooted at the reference. Black and red symbols (e.g., $B = 5$) denote error bounds before and after the sync update, respectively.

Algorithm 1 Sync procedure at each endpoint $Node_i$

T_{sync}: pre-configured sync interval.
D_i: profiled drift expectation of $Node_i$.
V_i: profiled maximum drift variance of $Node_i$.
H: profiled maximum hop error in the RDCN.
clk_i: current clock value of $Node_i$.
B_i: current clock error bound of $Node_i$.

1: **function** MAIN
2: $B_i \leftarrow 0$ **if** $i = 0$ **else** $B_i \leftarrow \infty$ ▷ Initialize error bound
3: **thread.spawn** SEND ▷ Start the send thread
4: **thread.spawn** RECV ▷ Start the receive thread
5: **wait** ▷ Run indefinitely
6: **function** SEND ▷ Function to send sync messages
7: **while true do**
8: wait T_{sync}
9: $clk_i \leftarrow clk_i - D_i \times T_{sync}$ ▷ Compensate drift
10: $B_i \leftarrow B_i + V_i \times T_{sync}$ ▷ Update error bound
11: Broadcast $msg(B_i, clk_i)$
12: **function** RECV ▷ Function to receive sync messages
13: **while true do**
14: wait until receive $msg(B_j, clk_j)$
15: **if** $B_i > B_j + H$ **then**
16: $clk_i \leftarrow clk_j$ ▷ Adopt the clock
17: $B_i \leftarrow B_j + H$ ▷ Adopt tighter bound
18: MAIN()

to identify specific circuits, which is not possible until nodes converge to the pre-computed global error bound (§3.5). In the *recv* thread, upon receiving a sync message (*Lines 14*), the node checks whether adopting the received clock would tighten its error bound (*Lines 15*). If so, the node updates its local clock (*Line 16*) and sets its error bound to the received bound plus the hop error (*Line 17*).

Fig. 3 illustrates eight endpoints synchronizing once per time slice. The table (top right) shows the profiled maximum drift variance (with drift expectation compensated), and the maximum hop error is set to $H = 5$ ns. The reference's bound is fixed at $B_0 = 0$, while all other nodes start with $B_i = \infty$.

At each sync interval, the reference synchronizes its neighbors, forming a new clock propagation tree. A sync child's bound equals its parent's bound plus the hop error. For example, at time slice 1 ($ts = 1$), $Node_1$ and $Node_5$ each have $B = 5$ ns. After a sync interval, every node's bound increases by its profiled drift variance. Thus, $Node_1$'s bound at $ts = 2$ becomes 12 ns. Already-synchronized nodes may further synchronize others: e.g., at $ts = 2$, $Node_1$ synchronizes $Node_3$, yielding $B = 12\,\text{ns} + 5\,\text{ns} = 17\,\text{ns}$ for $Node_3$.

A node only updates when the prospective parent's bound plus H is lower than its current bound, ensuring bounds monotonically improve. For instance, at $ts = 4$, $Node_6$ synchronizes $Node_2$ (not vice versa), and $Node_3$ and $Node_6$ do not synchronize each other, since neither reduces the other's bound.

3.3 Single-Message Synchronization

Recall from Fig. 2 that each sync operation in traditional sync protocols requires three sync messages, one for clock propagation and two for measuring the propagation delay. This procedure takes three (one-way) propagation delays, along with two processing delays taking at least 1 µs each [20]. The cable length in production DCNs varies between several to hundreds of meters [19]. With the propagation delay of 5 ns/m [1], the sync process over a 300 m cable takes at least 6.5 µs, exceeding the time slice duration of many RDCNs.

To ensure the success of the sync action, the exchange of the three sync messages must finish within one time slice, otherwise the optical circuit would be gone. The remaining messages can either wait until the next circuit cycle when the circuit reappears or be routed through other nodes where the circuit is available. However, in the former case, the two successive messages have a time gap of one cycle, incurring large drifts and sync errors. For the latter, the messages traverse paths of different lengths, violating the basic assumption of path symmetry in the delay measurement shown in Fig. 2.

We solve this problem with single-message sync: only keep the initial sync message for clock propagation and eliminate the following two for real-time propagation delay measurement. Instead, we pre-profile propagation delays before network operation, similar to drift and hop error profiling, with the specific profiling procedure described in §3.1. This method benefits from direct connections of endpoints in RDCNs, making one-hop synchronization capture the pure propagation delay with minimal jitter. In contrast, traditional DCNs usually synchronize across multiple hops, where processing, queuing, and transmission delays introduce significant jitter.

3.4 Hop Error Mitigation

As explained in our error model (§2.2), among the three sources of hop error, path asymmetry from unequal forward and reverse paths does not arise in RDCNs when synchronizing with direct neighbors only, and timestamping noise, jitter due to NIC clock precision limit, is unavoidable. We aim to reduce the remaining factor, delay asymmetry, to minimize hop error and thus SyncWise's overall error bound, advancing toward the accuracy limit of synchronization in RDCNs.

For two connected nodes A and B, delay asymmetry is the

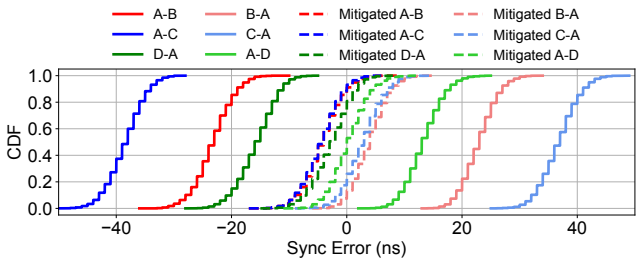


Figure 4: Delay asymmetry mitigation for different port pairs.

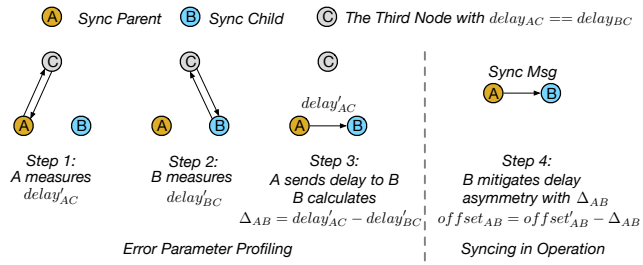


Figure 5: Delay asymmetry mitigation based on Eqn. 6 – 9.

difference in message transmission delays between the two directions, i.e., $A \to B$ and $B \to A$, and is caused by intrinsic hardware processing delays and wiring layouts that do not vary over time. So we model its effect on the sync result for each node pair as a constant offset, denoted by Δ_{AB}, and compensate for it with a mathematical approach. We specify the TX timestamp inaccuracy as e, so the actual TX timestamp is $T = T' - e$, where T' is the measured TX timestamp. Similarly, in the following derivation, we use symbols with primes to denote measured values and symbols without primes to denote ground-truth values. Among the four timestamps in Fig. 2, T_1^A and T_2^B are TX timestamps, and T_2^A and T_1^B are RX timestamps. Because RX timestamps are accurate while TX timestamps are not, we can rewrite Eqn. 1 and Eqn. 2 in Fig. 2 by considering the TX timestamp inaccuracy, i.e., $T = T' - e$.

$$\begin{aligned} delay'_{AB} &= \frac{T_2^A - T_1^{A'}}{2} - \frac{T_2^{B'} - T_1^B}{2} \\ &= \frac{T_2^A - (T_1^A - e^A)}{2} - \frac{(T_2^B - e^B) - T_1^B}{2} \\ &= \frac{T_2^A - T_1^A}{2} - \frac{T_2^B - T_1^B}{2} + \frac{e^A + e^B}{2} \\ &= delay_{AB} + \frac{e^A + e^B}{2} \end{aligned} \tag{6}$$

$$\begin{aligned} offset'_{AB} &= T_1^B - delay'_{AB} - T_1^{A'} \\ &= T_1^B - (delay_{AB} + \frac{e^A + e^B}{2}) - (T_1^A - e^A) \\ &= (T_1^B - T_1^A - delay_{AB}) + e^A - \frac{e^A + e^B}{2} \\ &= offset_{AB} + \frac{e^A - e^B}{2} \end{aligned} \tag{7}$$

Hence, the effect of delay asymmetry on the sync result, Δ_{AB}, is the difference between the measured and true clock offsets in Eqn. 7, i.e. $(e^A - e^B)/2$. We eliminate that term by introducing a reference node C. For simplicity, here we present the case where node C has the same propagation delay (or same cable length) with node A and node B, i.e., $delay_{AC} = delay_{BC}$. The general case for any arbitrary reference nodes is shown in Appx. A. If $delay_{AC} = delay_{BC}$, reusing Eqn. 6 on $delay_{AC}$ and $delay_{BC}$, we have Eqn. 8.

$$\begin{aligned} delay_{AC} &= delay_{BC} \\ delay'_{AC} &= delay_{AC} + \frac{e^A + e^C}{2} \\ delay'_{BC} &= delay_{BC} + \frac{e^B + e^C}{2} \end{aligned} \tag{8}$$

Combining the equations in Eqn. 8, we derive Δ_{AB} in Eqn. 9, indicating Δ_{AB} can be obtained by measuring the propagation delays between nodes A and C and between nodes B and C.

$$delay'_{AC} - delay'_{BC} = \frac{e^A - e^B}{2} = \Delta_{AB} \tag{9}$$

We validate this method in Fig. 4, where we sync different port pairs of an Intel Tofino2 switch. The ports A–D on the same switch share the same physical clock, so the ground-truth clock offset after synced by another port ($offset_{AB}$ in Eqn. 7) is always zero. Taking that into Eqn. 7, we have $offset'_{AB} = (e^A - e^B)/2$, which shows the delay asymmetry $(e^A - e^B)/2$ manifests directly in the measured clock offset $offset'_{AB}$ (or sync error in Fig. 4). Delay asymmetry is port-pair-specific, and swapping the sync parent and child on the same ports negates the asymmetry. This explains why different port pairs exhibit different sync errors (offsets) in Fig. 4, and why flipping ports reverses the error distribution.

These results motivate the need for delay asymmetry mitigation. In Fig. 4, the median of sync errors is up to 38 ns before mitigation, and within 4 ns after. The variance range of sync errors corresponds to unavoidable timestamping noise.

The delay asymmetry mitigation process in SyncWise is illustrated in Fig. 5. We find a reference node C for every node pair A and B, where the cable length difference between AC and BC can be calculated from the DCN wiring plan. Here we assume AC and BC have equal cable lengths, consistent with Eqn. 6 – 9. The generalized method, which modifies Eqn. 6 – 9 to incorporate unequal cable lengths in computing Δ_{AB} is detailed in Appx. A.

As explained in §3.3, propagation delays of every pair of node ports are pre-profiled, including $delay'_{AC}$ and $delay'_{BC}$ (*Steps 1, 2*). After that, the nodes exchange these delays, so that a child node, such as B in Fig. 5, can calculate Δ_{AB} regarding every potential parent node A (*Step 3*). Then at synchronization time, the child node can apply the stored Δ_{AB} to get the accurate clock offset relative to a sync parent (*Step 4*).

3.5 SyncWise Properties

Unlike conventional time sync protocols that rely on best-effort clock estimation, SyncWise's error-aware synchronization provides bounded sync errors, where the bound is minimal, and computable a priori. Here, we prove these properties.

We model an RDCN as a dynamic (time-varying) graph with *bounded-joint connectivity*, defined as follows.

Definition 3.1 (Bounded-Joint Connectivity)**.** Let $G(t) = (N, E(t))$ be a dynamic graph, where N is the set of nodes and $E(t)$ is the set of edges present at time t. We say that $G(t)$ satisfies *bounded-joint connectivity* if there exists a constant $\Delta > 0$ such that, for every time $t \geq 0$, the union of edges over

the interval $[t, t+\Delta]$ forms a connected graph:

$$\left(N, \bigcup_{\tau \in [t, t+\Delta]} E(\tau)\right) \text{ is connected.}$$

Property 3.1. SyncWise guarantees bounded sync errors.

Proof. We prove by contradiction. Suppose the sync errors are unbounded. Then there exists at least one node $Node_i$ whose error bound B_i (as in Alg. 1) grows without bound.

Consider any neighbor $Node_j$ that is ever connected to $Node_i$. Assume, for contradiction, that B_j is bounded. Then there exists a time t when $B_j + H < B_i$. At the next synchronization, $Node_i$ updates its bound to $B_i \leftarrow B_j + H$, which is finite. Between synchronizations, its clock drifts by at most $V_i \cdot \Delta$, where V_i is the max drift variance of $Node_i$. Thus, $B_i \leq B_j + H + V_i \cdot \Delta$, which remains bounded. This contradicts the assumption that B_i grows unbounded, so all $Node_i$'s neighbors, like $Node_j$, must have infinite bounds.

By induction, all neighbors ever connected to $Node_i$'s neighbors, e.g., $Node_j$'s neighbors, must also have infinite bounds. By *bounded-joint connectivity*, however, all nodes become connected within every interval of length Δ, including the reference node $Node_0$, which has $B_0 = 0$. This is a contradiction. Therefore, all node error bounds remain bounded, the worst-case sync error in the RDCN, given by the largest node bound, is also bounded. □

Property 3.2. SyncWise achieves the minimal (i.e., optimal) error bound under synchronization with direct neighbors[7].

Proof. Assume, for contradiction, that some node $Node_i$ attains a non-minimal error bound during a sync step. By Alg. 1, a node always synchronizes with the neighbor that yields the smallest possible bound. Thus, if $Node_i$'s bound were non-minimal, at least one neighbor $Node_j$'s bound must also be non-minimal; otherwise, $Node_i$ would adopt $Node_j$'s bound and become minimal. Similarly, $Node_j$ must have had a non-minimal neighbor in a previous sync step. Inductively, we trace back to the initial step to synchronize with the reference, whose bound $B_0 = 0$ is minimal, giving a contradiction.

Every node maintains a minimal bound, so the global error bound (the maximum over all nodes) is also minimal. □

Property 3.3. Given a periodic circuit schedule and fixed error parameters, the error bound of nodes is periodic and can be computed a priori.

We present a high-level proof sketch here and defer the full details to Appx. B. We define a mapping function f that characterizes how the error bound evolves after the sync operations within a single circuit period P. Because the circuit schedule is periodic, the same mapping f is applied in every period. Thus, at circuit-period granularity, the error bound $B(t)$ evolves as

$$\{B(0), f(B(0)), f(f(B(0))), ...\}.$$

To establish periodicity of the error bound, it suffices to show that (1) f has a fixed point [11] $x^\star$ such that $f(x^\star) = x^\star$, and (2) the sequence $B(t)$ converges to this fixed point.

We first prove the existence of a fixed point using Tarski's fixed-point theorem [34]. Specifically, we show that SyncWise's synchronization mechanism satisfies the required preconditions: f is monotone and $f(D) \subseteq D$, where D is a complete lattice. We then prove reachability of the fixed point. In addition to being monotone, f is non-increasing, and D as a complete lattice is finite. Therefore, by applying f a finite number of times, the error bound will reach the fixed point. As f contains sync operations within a single circuit period P, this implies that the error bound eventually becomes periodic with period P, i.e., $B(t+P) = B(t)$.

Finally, we bound the number of sync intervals required for the error bound to reach the periodic phase. By the sync procedure and Property 3.2, in each circuit period, at least one additional non-reference node attains its minimum achievable error bound. Since there are $n-1$ non-reference nodes, $B(t)$ reaches the periodic phase within at most $n-1$ circuit periods, i.e., within $(n-1)P$ sync intervals.

Thus, we can simulate the execution of Alg. 1 for $(n-1)P$ sync intervals to reach the periodic phase, and then for an additional P intervals to complete one full period. We take the maximum sync error across all nodes during this one full period as the global error bound. The guardband is then set to the sum of the OCS reconfiguration delay and this global error bound.

4 Implementation

The design of SyncWise is general and not tied to specific hardware. We prototype SyncWise on Intel Tofino2 programmable switches and briefly describe our implementation. It is easily portable to other hardware, e.g., NICs and FPGAs.

The optical controller (a server on our testbed, §6) initiates the parameter profiling procedure. It informs the control plane of the involved nodes of the upcoming profiling operations, such as drift, hop error, or propagation delay measurements and their destinations, by constructing circuits to them (via the reference node), and then reconfigures the OCSes to establish the required circuits for profiling. The nodes then execute the profiling on the data plane. Afterward, the controller re-establishes connections with the nodes to verify success. The profiling process is delay-insensitive but success-critical, making the coordination and acknowledgment necessary.

Nodes collect profiling results and compute parameters on the control plane. The controller aggregates these error parameters from all nodes and simulates the procedure of

[7] Synchronizing with non-neighbors may achieve smaller error bounds. However, this requires circuits to persist while sync messages propagate over multiple hops, which may not hold in modern RDCNs [2, 4]. To maintain generality, nodes in SyncWise only synchronize with their direct neighbors.

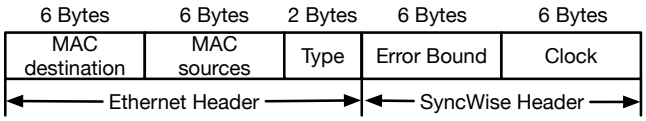


Figure 6: SyncWise sync packet format.

nodes running Alg. 1 until reaching the period P to determine the largest error bound (§3.5). The largest bound within a period defines the global error bound. The guardband is then set to the OCS reconfiguration delay plus this bound, and the time slice duration is chosen accordingly (e.g., two orders of magnitude larger than the bound). Finally, the controller informs the nodes of these system parameters, including the number of sync steps for Alg. 1 to enter steady periods (nodes can reliably carry traffic reaching this point). The controller then configures the RDCN to begin operation.

We implement SyncWise sync messages as Ethernet packets, though they could be realized at other layers. As shown in Fig. 6, these packets carry a dedicated *EtherType* to identify them as SyncWise sync messages. Each node stores its profiled error parameters, error bound, and clock offset in registers. The product of maximum variance and sync interval ($V_i \times T_{sync}$) is pre-computed on the control plane, so all state updates in Alg. 1 reduce to simple additions and overwrites. Every T_{sync}, a sync message is generated and broadcast using Tofino2's packet generator, which guarantees transmission at the configured interval. The error bound is retrieved from registers and written into the packet, along with the node's clock value, as the hardware TX timestamp in the egress pipeline plus the clock offset.

5 Discussion

Controller-free architectures. SyncWise can be adapted for controller-free RDCNs, e.g., those with AWGR-based OCSes [2, 4], which are passive and require no controller to establish explicit circuits. These OCSes support arbitrary connections provided no port conflicts occur, purely relying on endpoint coordination to prevent collisions. One endpoint can serve as the reference, with its clock as the reference. The sync messages (Alg. 1) shifts from broadcast to targeted sync children. Once the reference synchronizes its direct neighbors, these nodes can locate themselves in the pre-loaded circuit schedule and propagate the clock further. When two nodes connect, each sends a sync message without collision, thanks to unidirectional links on optical fabrics, and adopts the clock only if it reduces the error bound. Error parameter profiling is initiated and coordinated by the reference without ambiguity.

Link and node failures. The guardband must be large enough to absorb failures that may inflate the error bound. To tolerate *N link or node failures*, we enumerate the failure scenarios, simulate the execution of nodes running Alg. 1 for the degraded topologies (§7.5), and take the largest error bound across all scenarios as the global error bound.

The guardband is designed to cover the maximum possible sync error under the specified failure conditions. In the paper, we focus on frequent failures commonly observed in DCNs, such as node failures, link failures, and overheating. If a particular type of failure occurs frequently in a given RDCN, users can include it in the simulator to derive the corresponding error bound under that failure scenario.

Multiple references. We assume the optical controller, essential to the optical fabric, cannot fail, and its resilience (e.g., primary-backup redundancy) is provided by the RDCN architecture. The reference endpoint attached to the controller, however, may fail. To tolerate such *reference failures*, SyncWise connects the controller (and its backups, if any) to multiple reference nodes that can synchronize other nodes in parallel. Our sync protocol (Alg. 1) naturally adapts to this setting, treating multiple references as a single logical node with higher connection degree. The guardband is set to the worst bound under the protected level of failures (e.g., at least one reference node).

Temperature change. We assume stable room temperature without cooling malfunctions, as such events would be detected by temperature sensors and require human intervention. Yet, we consider the more frequent localized endpoint *overheating*, e.g., due to heavy traffic workloads [22].

Prior work shows that temperature primarily shifts the *drift expectation*, while the *variance range* remains relatively stable [28]. Thus, under varying traffic loads, SyncWise only needs to compensate for the current drift expectation; the protocol based on drift variance remains unchanged. Beyond the one-time profiling performed before deployment, every sync operation effectively serves as a drift measurement[8]. Each endpoint tracks drift expectation using a sliding window and compensates with the most recent value. In case of sudden workload changes where compensation may lag, the uncompensated portion is covered by the guardband. Specifically, in addition to profiling under idle conditions (§3.2), we also profile each endpoint under continuous line-rate traffic. The difference in drift expectations between idle and fully loaded cases defines the maximum uncompensated drift. This value is added to the variance range when computing the worst-case error bound, which then determines the guardband.

Benefits within and beyond RDCNs. SyncWise's error-aware synchronization enables nodes to select sync parents with the most accurate clocks. In RDCNs, it leverages the dynamic topology, where access to a larger set of candidate parents amplifies the benefit of this selection. Beyond RDCNs, we expect that SyncWise can also enhance existing error-agnostic sync protocols in static DCNs by enabling nodes to always synchronize with the most accurate parents.

6 Prototype Testbed

We implement SyncWise on programmable switches and build an optical testbed to validate our implementation. Large-scale evaluation of SyncWise is in §7.

As Fig. 7 shows, our testbed consists of a 24×24 MEMS

[8] Sync messages have higher priority than regular traffic.

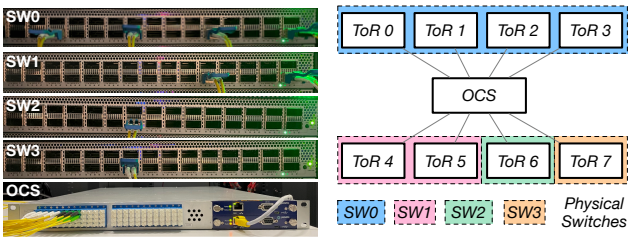


Figure 7: Setup of the optical testbed.

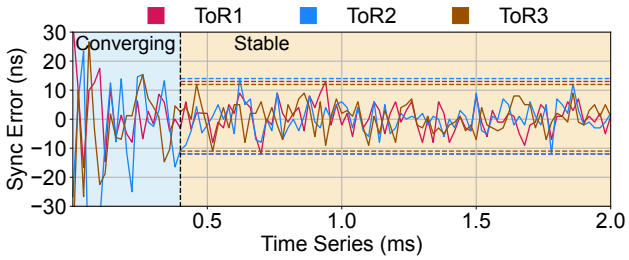


Figure 8: Sync errors of ToR_1-ToR_3 on the testbed (Fig. 7).

OCS and 4 EdgeCore DCS810 Intel Tofino2 switches, where 8 logical ToRs are realized on the physical switches (colored in the figure) and each connected to the OCS through a 3 m long fiber at 100 Gbps. ToR_0 is the master ToR for synchronization. Optical fabrics for modern RDCNs are not yet commercialized, though many have been demonstrated in labs. We therefore use commercial MEMS-based OCSes with a 10 ms reconfiguration delay and set the time slice duration to 1 s. From a system perspective, MEMS behaves like the microsecond- and nanosecond-speed OCSes targeted by modern RDCNs, except for the longer time slices. These longer slices increase drift, serving as a stress test for SyncWise.

Effectiveness of error-aware synchronization. First, we validate the error-aware synchronization on the testbed. As illustrated in Fig. 7, we make ToR_1-ToR_3 observed ToRs and co-locate them with ToR_0 on SW_0 to measure their sync errors. The ground-truth sync offset between co-located ToRs is zero, so the offset produced by SyncWise is the sync error. As ToR_1-ToR_3 have no drift relative to ToR_0, we add profiled drift and drift variance from SW_1-SW_3 to them.

Fig. 8 shows the evolution of sync errors over time. During the initial converging, ToRs experience sync errors of up to 81 ns; afterwards, sync errors stabilize and remain consistently below 14 ns. The slight differences in error ranges among ToR_1–ToR_3 arise from their distinct positions in the clock propagation paths, reflecting the combined influence of the circuit schedule, hop error and drift variance.

Hardware resource usage. Finally, we compare the hardware resource usage of SyncWise with DPTP [20], a PTP implementation in the programmable switch data plane. Table 2 reports the hardware usage of SyncWise on Tofino2 programmable switches. Overall, SyncWise consumes minimal hardware resources, less than 6% resources of all types. Compared to DPTP, it requires slightly more crossbar and SRAM to store profiled parameters for hop-error mitigation

Table 2: Hardware resource usage on Tofino2 switches.

	Match Xbar	SRAM	VLIW Instructions	Register ALU
PTP	3.39%	3.33%	6.25%	3.12%
SyncWise	5.96%	5.62%	5.21%	2.78%

(§3.4) and single-message synchronization (§3.3). The size of these stored parameters grows linearly with the number of nodes in the RDCN; in this implementation, we consider a network of up to 1024 ToRs, representing an extremely large RDCN. The additional resources needed for error-aware synchronization, such as comparing and updating error bounds, are negligible.

7 Evaluation

We validated SyncWise's functionality on a testbed with an OCS and multiple programmable switches (§6). We now present large-scale simulations using production drift and hop error statistics to evaluate its sync accuracy (§7.2), generality (§7.3), design effectiveness (§7.4), and robustness (§7.5).

7.1 Simulation Setup.

We simulate Opera [25], a well-recognized RDCN architecture, whose round-robin circuit schedule forms an *expander* topology every time slice, as illustration in Fig. 3. The simulated RDCN comprises 108 ToRs, each with 6 optical uplinks (648 NICs total), with time slice duration of 50 µs.

Error sources follow prior DCN synchronization measurements [22, 28, 29]: hop error ±3 ns, static drift expectation ±100 ppm, and drift variance ±10 ppm. We set the maximum hop error H =3 ns and sample a random hop error within ±3 ns at each sync step. Each ToR is assigned a static drift expectation within ±100 ppm and a maximum drift variance V_i within ±10 ppm. At runtime, drift variance is randomly generated within the assigned V_i, and each ToR's clock drifts according to its static expectation plus this runtime variance. The default sync interval is 300 µs, unless specified otherwise.

Comparison baselines. As the first of its kind work, it is hard to find fair baselines for SyncWise. We compare SyncWise with an RDCN adapted version of **Graham** [28], where each ToR compensates static drift expectation locally and sync with the reference node once it has connections. We run SyncWise and the adapted Graham on Opera [25].

To put the numbers in context and break down the performance, we also compare SyncWise against sync protocols designed for static DCNs, even though they **do not** apply to RDCNs. We evaluate the baselines using an *m-ary tree* topology (as in DTP [21]) with the same number of ToRs and uplinks per ToR as in the above Opera setting. In this 6-ary tree, each ToR connects to a single parent and five children, using its 6 uplinks. This topology yields the smallest average hop count between node pairs, making it more favorable to sync solutions on static DCNs than any other static topology.

We select **Sundial** [22], **Firefly** [29], and **DTP** [21] for comparison, as they represent the most accurate sync solutions for static DCNs across the main categories of tree-based

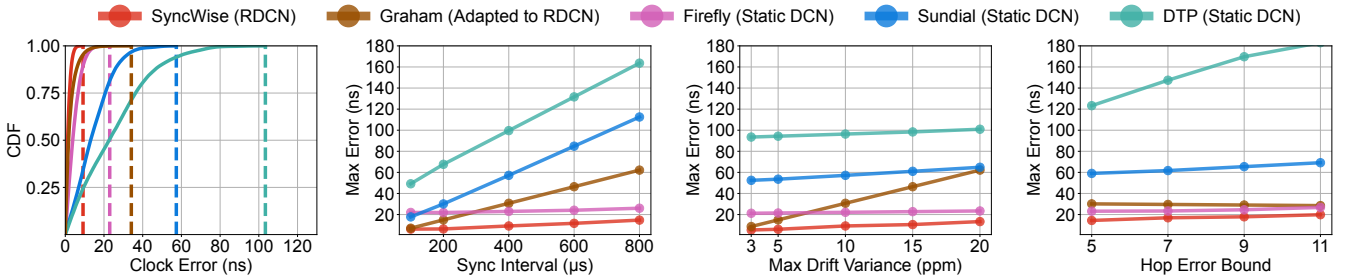


Figure 9: Sync error distributions of various sync solutions.

Figure 10: Max sync errors under different sync intervals.

Figure 11: Max sync errors under diff. drift variance ranges.

Figure 12: Max sync errors under diff. hop errors.

and gossip-based approaches (§2.3). For a fair comparison, all baselines are configured with the same drift and hop error statistics and sync frequency as SyncWise, isolating the evaluation to the sync mechanisms themselves, although some may benefit from specialized hardware.

7.2 Performance Overview

We first give an overview of the SyncWise accuracy. Fig. 9 presents the sync error distributions of SyncWise and other baselines under the default setting described in §7.1. We evaluate sync accuracy using the maximum sync error (100^{th} percentile). As discussed in §1, this worst-case error is critical in RDCNs for determining the time slice duration and the guardband size.

SyncWise achieves 9 ns sync accuracy (maximum sync error) and 15 ns error bound. This maximal sync error represents a 3.8× improvement over the RDCN-adapted Graham. This improvement stems from its error-aware clock propagation tree, which enables nodes to sync with any available neighbor offering a more accurate clock. In contrast, while Graham compensates for drift locally, its errors continue to grow when not synced, as drift variance cannot be corrected precisely. Moreover, without a clock propagation tree, ToRs are limited to synchronizing directly with the reference, but such opportunities are rare, resulting in large sync errors.

Compared to Sundial, a tree-based protocol for static DCNs, SyncWise delivers 6.3× higher accuracy. Unlike tree-based methods that select parents solely by hop count and ignore drift, SyncWise explicitly models sync error, ensuring nodes sync with the most accurate clocks.

Against DTP, SyncWise achieves 11.4× higher accuracy. DTP floods clocks throughout the network, always converging to the highest-frequency clock. This method heavily relies on high-frequency synchronization enabled by specialized hardware; otherwise, drift accumulates and hop errors propagate through the m-ary tree, despite bounded hop counts. SyncWise, in contrast, explicitly bounds both drift and hop error, and its error-aware synchronization protocol guarantees selection of the neighbor clock with minimal error, maintaining robustness even at relatively low sync frequencies.

While Firefly also achieves sub-10 ns sync error at the 99^{th} percentile, its maximal sync error exceeds 20 ns. This is because synchronizing with a large number of peers helps reduce expected error by canceling fluctuations but provides no guarantee on the worst-case error.

7.3 Influence of Different Factors

Next, we analyze different aspects of SyncWise by quantifying the influence of various factors, highlighting its generality.

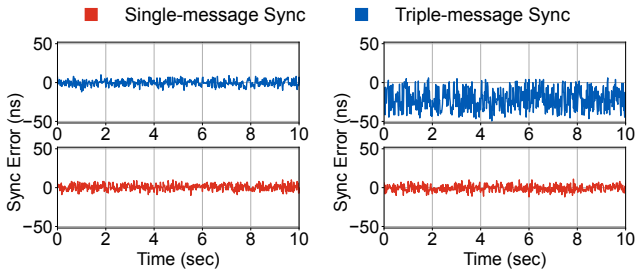
Impact of sync frequency. In Fig. 10, we evaluate SyncWise's generality under varying sync intervals. As the sync interval increases from 100 µs to 800 µs, SyncWise's maximum sync error rises by only 9 ns. In contrast, Graham, DTP, and Sundial see increases of 55 ns, 114 ns, and 95 ns, respectively. SyncWise's generality to diverse sync intervals attributes to its error-aware design. When synchronization becomes less frequent, the drift term in our error bound becomes increasingly dominant. This encourages nodes to sync with neighbors most recently synced to reduce their error bounds. DTP, Sundial, and Graham do not adapt their clock propagation strategies to changing sync intervals, leading to significantly higher accuracy degradation due to higher drift.

Interestingly, Firefly shows the smallest increase (4 ns), though across all intervals its maximum error remains at least 11 ns higher than that of SyncWise. Firefly benefits from the averaging effect of synchronizing with a large number of peers, whose clocks drift in different directions. Nevertheless, its maximum error remains suboptimal.

Impact of drift variance. We further evaluate SyncWise under different drift variance ranges to assess its accuracy across varying clock crystal qualities. As shown in Fig. 11, we vary the drift variance range from ±3 ppm to ±20 ppm, representing clocks from high to low quality. SyncWise's maximum sync error increases by only 7 ns, 7.7× smaller than that of adapted Graham (54 ns). Graham's local drift compensation alone cannot mitigate the effect of drift variance, as it is only eliminated when synchronizing directly with the reference, which is infrequent. DTP, Sundial, and Firefly show similar trends as Fig. 10, as these experiments both increase drift.

Instead, SyncWise this time tends to sync with neighbors exhibiting the smallest drift variance. Regardless of changing conditions, SyncWise consistently uses the error bound to guide synchronization, ensuring decisions minimize the impact of the most adverse factor.

Impact of hop error. Finally, we evaluate SyncWise under varying hop errors. For fairness to baselines without hop-error



(a) 10 µs slice duration (b) 1 µs slice duration

Figure 13: Time series of errors in triple-message and single-message sync with slice durations of (a) 10 µs and (b) 1 µs.

mitigation (§3.4), we attribute hop error solely to unavoidable timestamping noise. The effectiveness of our hop-error mitigation is evaluated separately in §7.4.

In Fig. 12, as the maximum hop error increases from 5 ns to 11 ns, the maximum sync error of SyncWise rises by only 5 ns, which is smaller than that of both Sundial and DTP. Although Firefly and Graham are relatively robust to hop errors—Firefly via averaging across peers and Graham via infrequent one-hop synchronization with the reference—SyncWise consistently achieves the lowest maximum sync error across all settings. This advantage comes from SyncWise's ability to model hop error in its error bound and perform error-aware clock propagation, guiding nodes to sync with more accurate clocks, ones with less drift in this case.

Note: Factors evaluated in this subsection (Fig. 10 – 12) stretch to unrealistically large values, serving as stress tests. Even under these extreme conditions, SyncWise achieves the smallest sync errors, demonstrating its ability to adapt efficiently to adverse environments.

7.4 Optimization Effectiveness

We have designed single-message sync (§3.3) to support short time slices and hop error mitigation (§3.4) to improve sync accuracy. Here, we evaluate their effectiveness on Tofino2 programmable switches. We connect a loopback cable between two ports of the programmable switch to exchange sync messages and measure the sync error, following the same methodology described in §6.

Relaxation of slice duration with single-message sync. In Fig. 13, we test the triple-message and single-message sync mechanisms under 10 µs and 1 µs time slices, with 40% background traffic. Triple-message sync requires at least two processing delays and three propagation delays to finish synchronization (Fig. 2), i.e., 5.72 µs on our testbed, whereas single-message sync contains no processing delay and only one propagation delay, i.e., 15 ns delay in total.

As Fig. 13 shows, single-message sync can relax the time slice duration to 1 µs. In Fig. 13a, both triple-message and single-message sync can fit into the 10 µs slice duration, achieving sync accuracy under 11 ns. For 1 µs slices in Fig. 13b, single-message sync can still complete within one time slice, but triple-message sync cannot. The unfinished messages will be sent out in the next time slice via a multi-hop path [25, 27], introducing errors into delay measurement due to the delay asymmetry of the RDCN. Therefore, single-message sync maintains the same level of accuracy, while the sync errors in triple-message sync increase to 50 ns.

Sync error reduction with hop error mitigation. Recall in Fig. 4 we present the sync error distributions among 4 ports, where we observe that mitigation of delay asymmetry reduces the median sync errors, but does not change the range of the sync error variance, which stems from the unavoidable timestamping noise. We repeat this experiment but enlarge its coverage to all 32 ports of the switch to confirm effectiveness of our mitigation approach. Fig. 14 presents the absolute median sync errors before and after mitigation for 12 randomly sampled port pairs. The results are consistent with Fig. 4: the maximum sync error before delay asymmetry mitigation is 53 ns, which decreases to 7 ns after mitigation.

7.5 Deep Dive into Error Bound

Finally, we evaluate SyncWise's performance under diverse failure scenarios and imbalanced RDCN topologies to demonstrate its robustness.

Error bound under diverse failures. Failures in RDCN could influence the sync error bound, creating the risk of exceeding the guardband and causing packet loss. We focus on three representative cases that most significantly affect the error bound: node failures, link failures, and device overheating. A node failure prevents all of its neighbors from synchronizing with it; a link failure blocks sync messages between the two connected nodes; and overheating alters a device's drift rate, introducing sync errors even with drift compensation. To evaluate the impact, we inject these failures at varying levels (5–20% of nodes, 5–20% of links, and 0–100% of nodes for overheating), repeating each experiment with randomly selected failure points and measuring the maximum sync error across runs.

These failures are protected by simulating Alg. 1 under the targeted scenarios and using the resulting largest error bound to set the guardband (§5). As shown in Fig. 15, we find that 5% node or link failures increase the error bound by at most 0.2 ns, while even the extreme case of 20% node or link failures raises it by only at most 1.3 ns. Overheating produces slightly higher increases, since drift rate changes directly affect sync accuracy: 20% overheated devices raise the bound by just 0.5 ns, and even when all devices overheat, the bound increases by only 2.6 ns.

The robustness of SyncWise arises from two factors. First, error-aware sync enables nodes to sync with any available neighbors, providing next-optimal backups under node and link failures. Second, the error model (Eqn. 5) incorporates drift variance in the error bound. Device overheating is essentially enlarging the drift variance range before drift caused by temperature change is compensated. Our error-aware design avoids synchronizing with overheated devices, maintaining

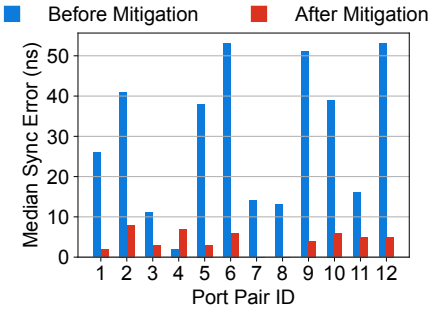


Figure 14: Absolute median sync errors before and after hop error mitigation.

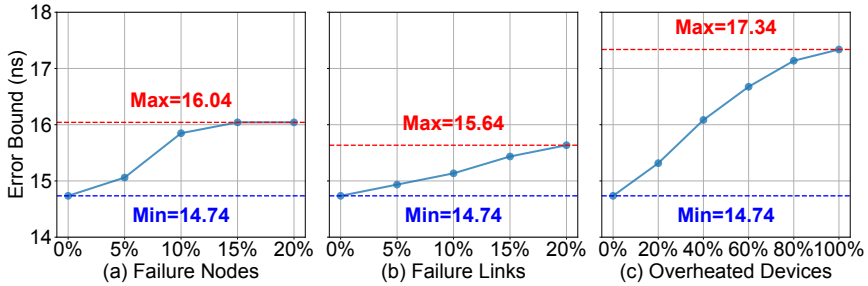


Figure 15: Error bound under (a) node and (b) link failures, (c) device overheat.

Table 3: Error bound of different topologies.

Skewness	Bound (ns)
1	15.4
2	16.2
3	16.4
4	17.2
5	18.0

Table 4: Convergence time (in ms) after topology changes.

Skewness	1	2	3	4	5
1	0	20	21	20	30
2	14	0	27	31	27
3	14	14	0	32	14
4	9	20	23	0	30
5	14	20	23	32	0

minimal error even under such conditions.

Error bounds under imbalanced RDCN topologies. So far, we have evaluated SyncWise on balanced RDCNs, where the circuit schedule provides even connectivity between node pairs. Semi-oblivious RDCNs [33] explores imbalanced circuit schedules, which forms hierarchical intra-group and inter-group clusters adapting the topology to traffic patterns. We measure RDCN imbalance by the skewness of the circuit schedule, defined as the ratio of aggregated bandwidth of intra-group vs. inter-group connections within a circuit cycle.

We next evaluate SyncWise's performance under varying skewness. As shown in Table 3, the sync error bound across different skewness levels ranges from 15.4 ns to 18.0 ns with a variation of only 2.6 ns. This small difference demonstrates that SyncWise maintains high sync accuracy even under extremely imbalanced connectivity. Since imbalanced topologies are designed to match temporal traffic patterns, they may change as traffic shifts. The small bound variation in Table 3 shows that a narrow guardband suffices to absorb dynamic schedule changes. Table 4 further demonstrates that after a topology change, RDCN stabilizes to a new error bound within 32 ms—well below the typical schedule update frequency. Thus, SyncWise synchronization does not constrain dynamic change of the circuit schedule.

8 Conclusion

We presented SyncWise, the first time synchronization protocol for RDCNs that operates across diverse architectures without requiring specialized hardware or dedicated management networks. For the first time, our error-aware synchronization provides a precise sync error bound, which is essential for RDCNs and many cloud applications, and leverage this bound to guide clock propagation. Unlike gossip-based solutions that synchronize heuristically, SyncWise deterministically selects peers with the most precise clocks, guaranteeing optimal accuracy. Moreover, its dynamic clock propagation tree exploits the reconfigurable topology of RDCNs to synchronize through richer sets of neighbors. SyncWise outperforms state-of-the-art synchronization protocols for static DCNs, overturning the long-standing assumption that accurate synchronization requires static topologies. These results underscore not only the superiority of synchronization on reconfigurable topologies but also the untapped potential of applying error-aware synchronization to static DCNs.

Acknowledgments

We sincerely thank our shepherd, Kyle Jamieson, and the anonymous reviewers for their insightful comments. We thank Zihang Wu for contributing the periodicity proof (Appx. B). We also thank Zihang Wu and Yonggang Jiang for discussions that strengthened the theoretical foundations of this work, and Yuliang Li for valuable feedback.

References

[1] Calculating Optical Fiber Latency. https://www.m2optics.com/blog/bid/70587/calculating-optical-fiber-latency.

[2] Daniel Amir, Nitika Saran, Tegan Wilson, Robert Kleinberg, Vishal Shrivastav, and Hakim Weatherspoon. Shale: A practical, scalable oblivious reconfigurable network. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 449–464, 2024.

[3] François Baccelli, Guy Cohen, Geert Jan Olsder, and Jean-Pierre Quadrat. *Synchronization and linearity*, volume 1. Wiley New York, 1992.

[4] Hitesh Ballani, Paolo Costa, Raphael Behrendt, Daniel Cletheroe, Istvan Haller, Krzysztof Jozwik, Fotini Karinou, Sophie Lange, Kai Shi, Benn Thomsen, et al. Sirius: A flat datacenter network with nanosecond optical switching. In *Proceedings of the Annual conference of*

the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication, pages 782–797, 2020.

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

[6] Joshua Lawrence Benjamin, Adam Funnell, Philip M Watts, and Benn Thomsen. A high speed hardware scheduler for 1000-port optical packet switches to enable scalable data centers. In *2017 IEEE 25th Annual Symposium on High-Performance Interconnects (HOTI)*, pages 41–48. IEEE, 2017.

[7] Kai Chen, Ankit Singla, Atul Singh, Kishore Ramachandran, Lei Xu, Yueping Zhang, Xitao Wen, and Yan Chen. Osa: An optical switching architecture for data center networks with unprecedented flexibility. *IEEE/ACM Transactions on Networking*, 22(2):498–511, 2013.

[8] Kai Chen, Xitao Wen, Xingyu Ma, Yan Chen, Yong Xia, Chengchen Hu, and Qunfeng Dong. Wavecube: A scalable, fault-tolerant, high-performance optical data center architecture. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 1903–1911. IEEE, 2015.

[9] Li Chen, Kai Chen, Zhonghua Zhu, Minlan Yu, George Porter, Chunming Qiao, and Shan Zhong. Enabling {Wide-Spread} communications on optical fabric with {MegaSwitch}. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 577–593, 2017.

[10] Yuxin Cheng, Matteo Fiorani, Rui Lin, Lena Wosinska, and Jiajia Chen. Potori: a passive optical top-of-rack interconnect architecture for data centers. *Journal of Optical Communications and Networking*, 9(5):401–411, 2017.

[11] Brian A Davey and Hilary A Priestley. *Introduction to lattices and order*. Cambridge university press, 2002.

[12] John C Eidson, Mike Fischer, and Joe White. Ieee-1588™ standard for a precision clock synchronization protocol for networked measurement and control systems. In *Proceedings of the 34th annual precise time and time interval systems and applications meeting*, pages 243–254, 2002.

[13] Marjan Fariborz, Xian Xiao, Pouya Fotouhi, Roberto Proietti, and SJ Ben Yoo. Silicon photonic flex-lions for reconfigurable multi-gpu systems. *Journal of Lightwave Technology*, 39(4):1212–1220, 2021.

[14] Nathan Farrington, George Porter, Sivasankar Radhakrishnan, Hamid Hajabdolali Bazzaz, Vikram Subramanya, Yeshaiahu Fainman, George Papen, and Amin Vahdat. Helios: a hybrid electrical/optical switch architecture for modular data centers. In *Proceedings of the ACM SIGCOMM 2010 Conference*, pages 339–350, 2010.

[15] Alex Forencich, Alex C Snoeren, George Porter, and George Papen. Corundum: An open-source 100-gbps nic. In *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 38–46. IEEE, 2020.

[16] Adam Christopher Funnell, Kai Shi, Paolo Costa, Philip Watts, Hitesh Ballani, and Benn Charles Thomsen. Hybrid wavelength switched-tdma high port count all-optical data centre switch. *Journal of Lightwave Technology*, 35(20):4438–4444, 2017.

[17] Yilong Geng, Shiyu Liu, Zi Yin, Ashish Naik, Balaji Prabhakar, Mendel Rosenblum, and Amin Vahdat. Exploiting a natural network effect for scalable, fine-grained clock synchronization. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 81–94, 2018.

[18] Monia Ghobadi, Ratul Mahajan, Amar Phanishayee, Nikhil Devanur, Janardhan Kulkarni, Gireeja Ranade, Pierre-Alexandre Blanche, Houman Rastegarfar, Madeleine Glick, and Daniel Kilper. Projector: Agile reconfigurable data center interconnect. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 216–229, 2016.

[19] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. Rdma over commodity ethernet at scale. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 202–215, 2016.

[20] Pravein Govindan Kannan, Raj Joshi, and Mun Choon Chan. Precise time-synchronization in the data-plane using programmable switching asics. In *Proceedings of the 2019 ACM Symposium on SDN Research*, pages 8–20, 2019.

[21] Ki Suh Lee, Han Wang, Vishal Shrivastav, and Hakim Weatherspoon. Globally synchronized time via datacenter networks. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 454–467, 2016.

[22] Yuliang Li, Gautam Kumar, Hema Hariharan, Hassan Wassel, Peter Hochschild, Dave Platt, Simon Sabato, Minlan Yu, Nandita Dukkipati, Prashant Chandra, et al. Sundial: Fault-tolerant clock synchronization for datacenters. In *14th USENIX symposium on operating*

systems design and implementation (OSDI 20), pages 1171–1186, 2020.

[23] Maciej Lipiński, Tomasz Włostowski, Javier Serrano, and Pablo Alvarez. White rabbit: A ptp application for robust sub-nanosecond synchronization. In *2011 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, pages 25–30. IEEE, 2011.

[24] Yunpeng James Liu, Peter Xiang Gao, Bernard Wong, and Srinivasan Keshav. Quartz: a new design element for low-latency dcns. *ACM SIGCOMM Computer Communication Review*, 44(4):283–294, 2014.

[25] William M Mellette, Rajdeep Das, Yibo Guo, Rob McGuinness, Alex C Snoeren, and George Porter. Expanding across time to deliver bandwidth efficiency and low latency. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 1–18, 2020.

[26] William M Mellette, Alex Forencich, Rukshani Athapathu, Alex C Snoeren, George Papen, and George Porter. Realizing rotornet: Toward practical microsecond scale optical networking. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 392–414, 2024.

[27] William M Mellette, Rob McGuinness, Arjun Roy, Alex Forencich, George Papen, Alex C Snoeren, and George Porter. Rotornet: A scalable, low-complexity, optical datacenter network. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 267–280, 2017.

[28] Ali Najafi and Michael Wei. Graham: Synchronizing clocks by leveraging local clock properties. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 453–466, 2022.

[29] Pooria Namyar, Yuliang Li, Weitao Wang, Nandita Dukkipati, Kk Yap, Junzhi Gong, Chen Chen, Peixuan Gao, Devdeep Ray, Gautam Kumar, et al. Firefly: Scalable, ultra-accurate clock synchronization for datacenters. In *Proceedings of the ACM SIGCOMM 2025 Conference*, pages 434–452, 2025.

[30] Alessandro Ottino, Joshua Benjamin, and Georgios Zervas. Ramp: a flat nanosecond optical network and mpi operations for distributed deep learning systems. *Optical Switching and Networking*, 51:100761, 2024.

[31] George Porter, Richard Strong, Nathan Farrington, Alex Forencich, Pang Chen-Sun, Tajana Rosing, Yeshaiahu Fainman, George Papen, and Amin Vahdat. Integrating microsecond circuit switching into the data center. *ACM SIGCOMM Computer Communication Review*, 43(4):447–458, 2013.

[32] Leon Poutievski, Omid Mashayekhi, Joon Ong, Arjun Singh, Mukarram Tariq, Rui Wang, Jianan Zhang, Virginia Beauregard, Patrick Conner, Steve Gribble, et al. Jupiter evolving: transforming google's datacenter network via optical circuit switches and software-defined networking. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 66–85, 2022.

[33] Nitika Saran, Daniel Amir, Tegan Wilson, Robert Kleinberg, Vishal Shrivastav, and Hakim Weatherspoon. Semi-oblivious reconfigurable datacenter networks. In *Proceedings of the ACM HOTNETS*, pages 150–158. ACM, 2024.

[34] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. 1955.

[35] Guohui Wang, David G Andersen, Michael Kaminsky, Konstantina Papagiannaki, TS Eugene Ng, Michael Kozuch, and Michael Ryan. c-through: Part-time optics in data centers. In *Proceedings of the ACM SIGCOMM 2010 Conference*, pages 327–338, 2010.

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

[37] Yiting Xia, Mike Schlansker, TS Eugene Ng, and Jean Tourrilhes. Enabling topological flexibility for data centers using omniswitch. In *HotCloud*, 2015.

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

Appendix

A Supplementary of Timestamp Correction

$$delay'_{AC} - delay'_{BC} - (delay_{AC} - delay_{BC}) = \frac{e^A - e^B}{2} = \Delta_{AB} \quad (7)$$

$$delay'_{AC} - delay'_{BC} - 5 \times (l_{AC} - l_{BC}) = \frac{e^A - e^B}{2} = \Delta_{AB} \quad (8)$$

In the case where $delay_{AC}$ is not equal to $delay_{BC}$, we combine the equations in Eqn. 8 and derive Eqn. 7. By substituting the delays with fiber lengths (l_{AC} and l_{BC}), we show how to calculate Δ_{AB} in Eqn. 8. The unit of the fiber length is meter, and coefficient 5 in Eqn. 7 means the propagation speed for optical signals is $5\,\mathrm{ns/m}$.

B Error Bound Periodicity

We prove that, under a periodic circuit schedule and fixed error parameters, the error bound of SyncWise is periodic[9].

B.1 Problem Formalization

Consider a dynamic network with node set

$$N = \{Node_0, Node_1, \ldots, Node_{n-1}\},$$

where n is the number of nodes. We define a sequence of time instances at which synchronization occurs as *rounds*, $r = 0, 1, 2, \ldots$, where consecutive rounds are separated by one sync interval. At time instance r, the available circuits are represented by

$$E(r) \subseteq N \times N,$$

where $(i, j) \in E(r)$ indicates that a circuit between $Node_i$ and $Node_j$ is available at time instance r. We assume the circuit schedule is periodic with period P (P-periodic), i.e.,

$$E(r+P) = E(r), \quad \forall r.$$

As described in §2.2, each $Node_i$ has clock drift variance $V_i \geq 0$[10], and each synchronization introduces a hop error $H \geq 0$. Let $B(r) \in \mathbb{Z}^n$ denote the vector of error bounds at time instance r. According to Alg. 1, the reference node $Node_0$ maintains zero error bound:

$$B_0(r) \equiv 0, \quad \forall r.$$

The error bound of each non-reference node $Node_i$ ($i \neq 0$) evolves as

$$B_i(r+1) = \min\left\{ B_i(r) + V_i, \min_{j:(i,j)\in E(r)} \left(B_j(r) + V_j + H \right) \right\}.$$

According to the sync procedure, at round $r+1$, $Node_i$ updates its error bound by taking the minimum of: (1) its current bound plus its drift variance V_i (the first term), and (2) the smallest error bound of its neighbors plus that neighbor's drift variance (the second term).

We represent the above transformation as a min-plus matrix multiplication [3]:

$$B(r+1) = A_r \otimes B(r)$$

where $\otimes$ denotes min-plus multiplication defined as

$$(A_r \otimes x)_i = \min_j \{(A_r)_{ij} + x_j\}$$

where $x \in \mathbb{Z}^n$.

The matrix A_r is defined as:

$$(A_r)_{ii} = V_i, \qquad (A_r)_{ij} = \begin{cases} V_j + H, & (i,j) \in E(r), \\ +\infty, & \text{otherwise.} \end{cases}$$

Intuitively:

1. $(A_r)_{ii} = V_i$ corresponds to the case where $Node_i$ does not synchronize with any neighbor at time instance r, so its error bound increases by V_i.
2. $(A_r)_{ij} = V_j + H$ corresponds to synchronization with $Node_j$, yielding $B_i(r+1) = B_j(r) + V_j + H$.
3. $(A_r)_{ij} = +\infty$ indicates that no circuit exists between $Node_i$ and $Node_j$ at time instance r, such that *min* operations never select this term to forbid sync operations over non-existent circuits.

Since the circuit schedule is periodic with period P, the sequence of matrices $\{A_r\}$ is also periodic:

$$A_{r+P} = A_r, \quad \forall r.$$

We define the transformation function of the composed P consecutive min-plus matrix multiplications

$$f(x) = A_{P-1} \otimes A_{P-2} \otimes \cdots \otimes A_0 \otimes x,$$

which represents the evolution of the error bound over every full period P of sync operations.

We reduce the proof of Property 3.3 to the proof of the existence and reachability of a fixed point of f. A fixed point of the transformation f is a state $x^\star \in \mathbb{Z}^n$ such that $f(x^\star) = x^\star$. If f has at least one fixed point and this fixed point is reachable with the initial error bound specified in Alg. 1, then the error bound periodically returns to $x^\star$. Then the periodicity of the error bound is proved.

B.2 Theorem

We leverage Tarski's fixed point theorem [34] to prove the existence of the fixed point. The theorem requires a partially ordered set. We equip $\mathbb{Z}^n$ with the coordinatewise *partial order*: for vectors $x, y \in \mathbb{Z}^n$,

$$x \preceq y \quad \text{if and only if} \quad x_i \leq y_i \ \text{ for all } i.$$

[9] This proof was contributed by Zihang Wu of the Max Planck Institute for Informatics.

[10] In Eq. 5, V_i denotes the drift variance which is a rate. For ease of presentation of this proof, we redefine V_i here as the drift variance accumulated over one sync interval, i.e., $V_i \times \tau$ in Eq. 5.

Tarski's fixed point theorem. Let $(D, \preceq)$ be a complete lattice and let $f(D) \subseteq D$ be a monotone function. Then f has at least one fixed point in D.

A function f is monotone (with respect to $\preceq$) if $x \preceq y \Rightarrow f(x) \preceq f(y)$. A partially ordered set D is a complete lattice if every subset of D has both a least upper bound and a greatest lower bound in D.

To apply Tarski's theorem in our setting, we need

1. the previously defined synchronization transformation function f is monotone with respect to $\preceq$; and
2. the set of possible error bounds forms a complete lattice D such that $f(D) \subseteq D$.

Convergence to the fixed point. To prove the reachability of the fixed point, we consider an infinite sequence $\{x^{(k)}\}$ defined by $x^{(k+1)} = f\big(x^{(k)}\big)$. We aim to prove that, starting from the initial state $x^{(0)} = B(0)$, the sequence converges in finitely many sync rounds to a fixed point $x^\star$. Since f represents the composition of P consecutive sync rounds, once the sequence reaches $x^\star$, the error bound returns to $x^\star$ every P sync rounds. Consequently, $B(r)$ is P-periodic, i.e., $B(r+P) = B(r)$.

B.3 Proof

We now prove that the above required conditions are satisfied.

(1) Monotonicity

Let $\preceq$ be the coordinatewise partial order on $\mathbb{Z}^n$. Recall the definition of $\otimes$,

$$(A_r \otimes x)_i = \min_j (A_{ij} + x_j)$$

If $x \preceq y$, then for every i, we have $x_i \le y_i$. Therefore

$$\min_j (A_{ij} + x_j) \le \min_j (A_{ij} + y_j)$$

So we have

$$(A_r \otimes x)_i = \min_j (A_{ij} + x_j) \preceq \min_j (A_{ij} + y_j) = (A_r \otimes y)_i.$$

Thus, the operator $A_r \otimes (\cdot)$ is monotone. Since the composition of monotone operators is also monotone, and f is defined as a composition of $A_r \otimes (\cdot)$, f is monotone.

(2) $f(D) \subseteq D$ and Complete Lattice

We first construct a lattice D such that $f(D) \subseteq D$, and then show that D is a complete lattice.

Because $G(r) = (N, E(r))$ is bounded-joint connected (§3.5) and periodic, every node can reach the reference node in at most $n-1$ hops. Each hop adds at most H and $V_{\max} = \max_i V_i$ (the max drift variance of all nodes) to the error bound. Therefore, after at most $n-1$ hops,

$$B_i(r) \le (n-1)(H + V_{\max}) \quad \forall i, r.$$

Let

$$M = (n-1)(H + V_{\max}).$$

Therefore, all possible error bounds $B_i(r)$ satisfy $0 \le B_i(r) \le M$. In Alg. 1, each node initializes its error bound to infinity. In practice, each node initializes its error bound to $W \ge M$, where W is the maximum value representable given the precision of the synchronization node. Let $D = [0, W]^n \cap \mathbb{Z}^n$. Then every element of $f(D)$ lies in $[0, W]$, and therefore $f(D) \subseteq D$.

Since $D = [0, W]^n \cap \mathbb{Z}^n$ is a finite product of bounded integer intervals, it is closed under coordinatewise minimum and maximum operations. Hence for any subset $S \subseteq D$, the greatest lower bound and least upper bound are given by

$$(\inf S)_i = \min_{s \in S} s_i, \qquad (\sup S)_i = \max_{s \in S} s_i \quad (i = 1, ..., n),$$

both of which remain in D. Thus, every subset $S \subseteq D$ has a greatest lower bound and a least upper bound in D, implying that D is a complete lattice.

Since f is monotone and D is a complete lattice, by Tarski's fixed-point theorem, f has at least one fixed point in D.

(3) Convergence to the Fixed Point

After proving the existence of a fixed point, we now show that the error bound converges to this fixed point in finite sync rounds.

We consider the sequence $\{x^{(k)}\}$ where $x^{(k+1)} = f\big(x^{(k)}\big)$. Suppose $x^{(0)} = B(0)$, according to Alg. 1, we initialize the error bounds of all nodes to the W (except for the reference node $Node_0$). We have

$$x^{(0)} = B(0) = (W, W, \ldots, W) \in D,$$

which is the maximal element of D. Thus we have

$$f(x^{(0)}) \le x^{(0)}.$$

Because f is monotone and $f(x^{(0)}) = x^{(1)} \le x^{(0)}$, we have

$$f(x^{(k+1)}) = f(f(x^{(k)})) \le f(x^{(k)}).$$

Hence the sequence $\{x^{(k)}\}$ is non-increasing.

As the sequence is infinite, D is finite, and $f(D) \subseteq D$, the sequence eventually stabilizes to some $x^\star \in D$ with $x^\star = f(x^\star)$.

We next bound the convergence time. According to Property 3.2, in each circuit period, at least one additional node reaches its minimum achievable error bound. Since f is non-increasing and there are $n-1$ non-reference nodes, $B(r)$ converges to the fixed point in at most $n-1$ circuit periods, i.e., within $(n-1)P$ sync intervals. Consequently, for all $r \ge (n-1)P$, there is $B(r) = B(r+P)$. The sync error bound becomes P-periodic after at most $(n-1)P$ sync intervals.

□