2025 55th Annual IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume (DSN-S) | 979-8-3315-1203-3/25/$31.00 ©2025 IEEE | DOI: 10.1109/DSN-S65789.2025.00065

Digital Cluster Circuits for Reliable Datacenters

Davide Rovelli
Università della Svizzera Italiana
Lugano, Switzerland
SAP SE
Walldorf, Germany
roveld@usi.ch

Patrick Eugster
Università della Svizzera Italiana
Lugano, Switzerland
eugstp@usi.ch

***Abstract*—Promptly reacting to failures is key for highly available datacenter services. At the core, failure detectors (FDs) have to rely on timeouts which are hard to set correctly on top of heavily contended processors and network resources. To overcome unpredictable interaction jitter, designers still have to resort to large timeouts to preserve safety (i.e. false positives) at the cost of availability. The shift toward hybrid architectures – where applications span accelerators, disaggregated memory, and network devices – exacerbates this issue. With common-case latencies dropping to the µs-scale or lower, conservative timeouts on top of best-effort software make FDs and coordination services impractically slow and unreliable.**

Motivated by the growing adoption of programmable network devices (e.g. smartNICs, FPGA-switches), we propose a new paradigm to tackle this issue by pushing self-contained, time-sensitive services to hardware. We introduce the idea of modeling a distributed system as a large *digital circuit* characterized by a stable "clock signal" consisting of periodic packets delivered with ultra-low reliable latency. We highlight how current datacenter technologies can be used to enable synchronous interactions in practice and discuss how a reliable FD trivially built on top can be used to increase robustness and performance of both hardware and software distributed applications.

I. Introduction and motivation

A growing number of dependable applications run in distributed manner on custom off-the-shelf hardware hosted by third-party datacenter infrastructures. To ensure high availability, these distributed services must promptly detect and overcome failures typically achieved by the so-called process failure detector (FD) abstraction. FDs can employ a variety of detection techniques which, at their core, must rely on timeouts to detect the unresponsiveness of a remote process.

Unfortunately, current protocols built on top of commodity systems are subject to unpredictable processing and network latency due to heavy resource contention, e.g. sudden CPU utilization or traffic spikes. This gives rise to the enduring challenge of coordinating distributed processes, requiring a balance between consistency and performance. In most cases, system designers prioritize the former and increase timeouts to take into account worst-case interactions (communication + processing) times between processes in order to avoid false positives which might compromise safety. This comes at the cost of liveness, i.e., when more than few isolated process failures of processes without prompt removal or replacement, widespread systems that depend on majority quorum voting may stall, as the number of operational processes can fall below the required quorum threshold. While some recent systems still assume that large timeouts of multiple seconds are tolerable and can go as far as notifying an operator [1], this fallback mechanism is clearly unaffordable for core services in which even a minor downtime can lead to significant disruption [2], [3]. Such scenarios are particularly problematic for emerging µs-scale coordination services [4] (e.g. high-frequency trading), where the short lifespan of processes needs to be combined with high availability.

The need for fast and reliable interactions to overcome failures is further evidenced by the growing adoption of accelerator-based systems for commodity datacenters [5]. These include numerous accelerators for specific workloads, and programmable network devices (e.g. smart network interface controllers (smartNICs), programmable switches). In general, by significantly improving average performance, novel datacenter technologies put additional pressure on typical FD stacks using conservative timeouts, making them an unaffordable bottleneck. To put the gap into perspective, assume a replicated Apache Kafka [6] cluster using Waverunner [7], a state-of-the-art field programmable gate array (FPGA)-smartNICs accelerated state-machine replication (SMR) protocol, to safely replicate client requests at a high throughput. Kafka's default minimum timeout of 6s to detect a replica failure could result in ∼150 million lost requests which need to be re-transmitted by clients. While state-of-the-art FDs claim that end-to-end timeouts can be safely lowered to the ms-range (∼25000 lost requests in the previous example when using the fastest FD known to us [8]), there is still a significant disparity between common-case performance and responsiveness to failures under heavy system load.

The lack of end-to-end, deployable solutions to achieve reliable timeouts at the µs-scale especially affects the emerging paradigm of disaggregated memory (DM) [9], [10], separating applications across (a) compute nodes (CNs) with powerful central processing units (CPUs) but relatively small (decreasing [9]) amounts of memory and (b) memory nodes (MNs) with more memory. This increases potential for partial failures, as nodes in one tier can fail independently of counterparts in

Work supported in parts by SNSF grants #192121 and #197353, Hasler Foundation, and Meta Research through Research in Distributed Systems award.

the other [11]. Works in DM typically assume the process fail-stop model [12] implying that failures are reliably detectable, and reliable networks [11], [13], without specific system support.

Motivated by the inherent limitations of the asynchronous model for dependable systems, this work aims to open a discussion on the following research question: *(how) can we establish stable and fast remote process interactions in modern hybrid datacenters?*

We begin by observing that distributed applications sitting on top of commodity operating systems (OSs) are inevitably affected by several jitter sources since packets need to traverse the whole system stack, encountering contention on multiple layers of the classical network hardware/software stack [14]. While approaches to bypass some of the layers exist, e.g. kernel-bypass frameworks such as remote direct memory access (RDMA), it is still unclear to what extent they can effectively reduce *worst-case* latency needed for robust timeouts rather than optimizing for common-case (e.g. high-throughput packet processing [15], [16]). Some software approaches provide more comprehensive solutions for stable processing [17], [18] but are based on complex OS fine-tuning which is platform-specific and error-prone significantly limiting portability.

On the other hand, advances in programmable network devices not only allow for optimal offloading of small computing routines but also ensure that network processing is timely deterministic and precise. Specifically, devices such as FPGA-equipped smartNICs and switches directly attached to the wire have uninterrupted access to incoming packets and can dedicate processing resources to specific high-priority traffic preventing costly preemption. Motivated by the growing availability of programmable network hardware and by the establishment of reliable networking protocols, we propose to model distributed systems in datacenters as large *digital circuits*. The main characteristic of such model is the presence of an ultra-stable "clock" constituted by periodic incoming packets at every node. In the following sections we discuss how a digital cluster circuit (DCC) is a practical instance of the more generic synchronous model that can lead to improvements to reliability and performance of coordination primitives.

II. The digital cluster circuit model

A. A practical instance of the synchronous model

The ability of using reliable timeouts is the core advantage of synchronous system models over asynchronous ones. However, system designers commonly consider assuming upper time bounded interactions to be undesirable because (1) it is likely that jitter in processing or communication delays would break bounds which are set too low and (2) setting high bounds to encompass delays would lead to inefficiencies as processes end up waiting longer than necessary, impacting latency and availability. This led to the widespread adoption of the asynchronous model where packets are assumed to take an arbitrary time to travel between nodes. Asynchronous protocols guarantee safety and are suitable for several use-cases but present some fundamental impossibilities namely in the presence of crash failures [19] as these cannot be detected under asynchrony. In fact, asynchronous systems (including FDs) overcome limitations by assuming synchrony only eventually and only for some properties, setting prohibitively large timeouts on top of commodity systems without providing any guarantees. The distributed systems community has long regarded these costs as the unavoidable "price to pay", justified by the unreliability and poor performance of past networks and endhost systems.

Advances in networking and computing technologies challenge this common belief, exemplified by the real-time and cyber-physical systems communities which have been building synchronous systems for decades. We believe this time has come also for modern datacenters. In fact, recent work [20] shares some of our observations about the overly conservative asynchronous assumptions on modern systems, going as far as proposing a design for a fully synchronous datacenter. However, the current technological stack does not seem to be suited to adopt synchrony at such scale in terms of efficiency, due to the disruption to existing asynchronous services, and reliability, due to the uncertainty of best-effort network and processing layers (particularly software).

Instead, we propose an alternative: a novel system better suited to the current technological ecosystem, leveraging programmable network hardware within a subset of the datacenter, which we abstract as the *digital cluster circuit (DCC)*. Our system can be thought of as a practical instance of the "timely computing base" proposed by seminal work [21]. In their line of work, authors provide algorithmic foundations to reason about a model which can rely on varying degrees of timeliness provided by a specific subsystem, showcasing how it can be used to build dependable and reliable applications. By grounding these theoretical concepts in concrete implementations, DCC demonstrates its potential to enhance system performance and reliability in real-world datacenter environments. Through this practical adaptation, our initial work aims to bridge the gap between theoretical hybrid timing models and the operational demands of modern infrastructure.

B. System design

Figure 1 depicts a preliminary high-level design of a DCC. Modules are replicated units inside programmable network devices, which can be switches or smartNICs. Similarly to a classical digital circuit, every module receives a periodic clock signal which triggers a state transition based on custom logic and application inputs. In our system, inputs can also come from the rich clock (RCLK) signal, term which we use to denote low-jitter periodic packets going into every node. Modules contain a decoder module to parse the RCLK, information which is fed into custom logic based on the application (e.g. failure detection). The complexity of DCC's custom logic is limited by the processing capability and processing time constraints of network devices, which need to be able to operate at line speed. Nonetheless, DCC modules are well

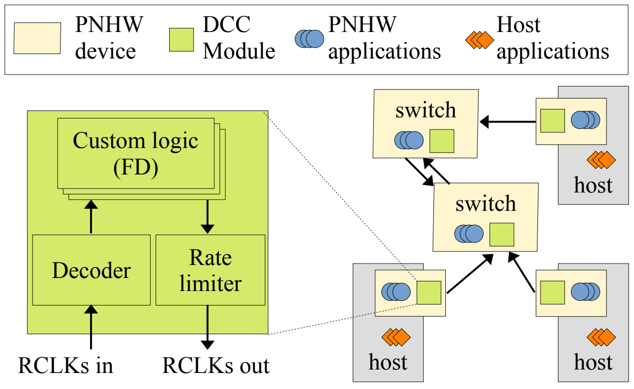

Fig. 1. Instance of the digital cluster circuit (DCC) model. Modules are deployed on programmable network hardware (PNHW) devices and exchange periodic rich clock (RCLK) packets. Interactions are logic dependent (e.g. FD) and may be asymmetrical.

suited for distributed synchronous coordination protocols and other short latency-centric tasks while applications can focus on the heavy lifting computation. Note that the logic block may vary between modules and may be replicated to handle multiple RCLKs signals. The custom logic output goes into a rate limiter: a fundamental module which controls the amount of egress traffic to avoid network congestion. The limiter follows a precise cluster schedule to ensure timely delivery for RCLK packets. DCC interactions can be asymmetrical, as shown in Figure 1 where the two switches aggregate multiple RCLK signals. Modules are dynamically deployed to programmable network devices following specific software and/or hardware-offloaded application processes.

C. Synchronous interactions with modern network hardware

DCC relies on the timely delivery of RCLK packets which needs to be ensured even upon heavy networking and processing load to avoid breaking the model's guarantees. Below we summarize the key emerging factors that substantiate the feasibility of our design:

- **Single administrative domain.** Datacenters have end-to-end interactions under administrative control. This gives operators and users have the ability of setting custom traffic policies and resource reservation in selected clusters.
- **Programmable network hardware.** Commodity datacenters are increasingly adopting hybrid workload-driven architectures [22] Technologies such as FPGAs, smartNICs, allow to offload small processing routines directly onto the wire, offering unprecedented programmability and accuracy.
- **Predictable processing times.** With network becoming increasingly faster, the overhead introduced by network stacks is no longer negligible. While state-of-the-art software processing pipelines on top of heavily multi-tasked commodity OSs are still unpredictable and hard to configure (e.g. RDMA [23]), programmable network devices offer a more stable alternative. Thanks to physically isolated processing units and the absence of software layers below them, hardware-offloaded routines promise to reduce jitter down a handful of clock cycles.
- **Bounded network latency.** Industrial network consortiums are actively working towards network reliability for time-sensitive traffic. Notably, IEEE time-sensitive networking (TSN) [24] and the Deterministic Networking IETF Task Force (DetNet) provide sets of standards specifically for bounded low latency and strong reliability in Ethernet and even wireless networks, with manufacturers providing growing support. These mechanisms, already adopted in real-time and cyber-physical systems, ensure that time-critical traffic meets hard deadlines, proving promising for datacenter applications. Furthermore, some datacenter-specific systems already show how to use software-defined networks (SDNs) which provide separated control planes to dynamically reserve network resources ensuring that packets never experience delay even under heavy traffic and congestion, even claiming bounded communication latency [17], [25], [26].
- **Optimal traffic scheduling.** TSN 802.1Qbv time-aware shaping [27] provides a mechanism that schedules traffic based on time using precision-time protocol (PTP) to ensure ns-range clock synchronization among network devices. These technologies also provide support to efficiently interleave time-sensitive and best-effort traffic, standing out among others for their clear potential to offer ultra-reliable rate limiting functionality to DCC modules while maximizing network utilization. An alternative (possibly to be used in combination) is the established priority traffic shaping through the use of switch multi-queues largely adopted by datacenter protocols.
- **Easier network programming.** Low-level data plane and hardware development (e.g. P4 [28]) frameworks for automatic hardware offloading [29], and platforms designed for large-scale, streamlined programming and deployment of network processing routines [30].

III. Building Dependable Applications with DCC

A. Reliable failure detection

The DCC can be used to build a typical heartbeat crash FD by trivially considering the absence of a RCLK packet as a failure indication. However, the presence of network failures, i.e. switch or link faults, might harm the accuracy of the failure detection. This is because in the occurrence of transient or permanent network partitions, a network failure might be falsely interpreted as failure of an end node by a subset of the processes, leading to inconsistent detection across the cluster. Below we propose methods to achieve reliability in DCCs by explicitly addressing network failures.

- **Exploiting redundancy.** Over-provisioned datacenters offer redundant physical connections among nodes (e.g. variations of the typical fat-tree topology [31]). Moreover, highly available core services are usually placed in a way to guarantee performance in practice, increasing the probability of having multiple alternative network paths between distributed replicas or placing them close to each other to make failures “atomic” (all/nothing). DCC could leverage the physical redundancy by sending parallel RCLK signals to reduce the probability of a false failure suspicion at the cost of increased network utilization.

- **Recovery networks.** Seminal work on resilient datacenter networks [31] proposes re-engineered fat-tree topologies and in-network failure detection to quickly overcome failures by dynamic re-routing. By integrating similar mechanisms within the stable communication protocols, DCC can safely overcome some network failures at the cost of slightly higher time bounds. This strategy can be complemented with safety backstops when a critical number of network failures is reached.
- **Monitoring neighbors.** Another option to avoid misinterpretation of network failures is to deploy DCCs so that every node sends a RCLK signal only to its neighbors (no multi-hop). The monitoring nodes take charge of propagating the failure notification which is therefore unequivocally detected by the entire cluster. This approach needs to carefully consider multiple network failures in rapid succession.

Combination of these mechanisms and DCC stable interactions offer concrete solutions to dramatically lower the probability of a false positive, providing an ultra-reliable backbone for datacenter services.

B. DCC-application interaction

DCCs are the building block of a novel practical model for developing hybrid services, where distributed applications running in a best-effort environment interact with a subsystem substrate ensuring timely and reliable periodic interactions. We anticipate that solutions will necessitate meticulous design focused on a clear interface which can be conceptualized as two bi-directional shared buffers between best-effort and reliable “domains”. In practice, buffers are implemented via high-bandwidth memory-mapped regions shared between end-host and network device or in-between hardware modules. Composition of reliable protocols in this model (only partially explored by the theory community [21]) presents opportunities but also challenges since designers need to take into account unbounded access times to buffers. However, asynchronous programs built on top of the DCC do not necessarily have to directly meet tight timing constraints imposed by the RCLK packet schedule and fast processing times of hardware-offloaded programs. Instead, they interact with the properties of an abstraction built within the DCC (i.e. reliability guarantees), making composition no more complex than when using purely asynchronous building blocks. One relevant example is application failure detection which extends basic heartbeat FD to detect custom failures. This is usually achieved by systems which carefully monitor entire systems through probes and complex notification schemes, ensuring broad coverage [32] including gray failure detectors (GFDs) [33], which leverage similar techniques to identify subtle malfunctions (e.g., deadlocks). While DCC cannot address complex software failures by default, it can be used to enhance the reliability and performance of GFDs by ensuring low, reliable crash failure detection down to the network level and also timely propagation of gray failure notifications by piggybacking them onto RCLK packets. Through a reliable FD or simply by pushing reliable periodic interactions to the network, DCCs offer an out-of-the-box solution to improve reliability and efficiency of both network-offloaded and software applications, enabling a new hybrid stack to rethink about practical distributed systems.

C. Discussion

We discuss some implications of integrating DCCs into a real-world system below.

- **Reliability breaches.** Clearly, there will always be a probability that some unforeseen event will breach DCC synchronous interactions hence hampering reliability, as it is the case for protocols widely regarded as reliable (e.g. TCP has the probability of 1 corrupt packet every 10 billion to go undetected at best [34]). Our goal is to reduce this probability as much as possible through concrete system design to provide stronger guarantees for services which cannot circumvent reliable timeouts and a low-risk alternative for more efficient distributed coordination based on synchrony. Model extensions such as the assumption of a correctly communicating quorum as shown by XFT [35] or the inclusion of omission failures [36] can be employed to further increase the robustness of the model.
- **Efficiency and reliability.** DCC requires some network resource reservation (e.g. priority queues) to ensure reliable time bounds. Contrarily to the common belief that synchrony requires inconveniently large safety margins, DCC relies on the predictability of hardware processing and reliable network standards hence resulting in convenient utilization of the reserved resources. Furthermore, the network can be shared with application traffic in the usual best-effort manner: programmers can size DCCs according to the application need, offering a trade-off between reliability and efficiency to highly available applications. Moreover, the benefits of synchronous interactions can outweigh reservation costs since they can reduce complexity of costly asynchronous coordination protocols (e.g. consensus) at the core of many distributed services as well providing higher fault-tolerance.
- **Scalability.** DCC design can accommodate multiple services which can either share a RCLK or leverage multiple modules coexisting on the same network device since modern network hardware such as FPGA-smartNICs come with sufficient capacity for hundreds of small modules. Designers can deploy custom scheduling algorithms to coordinate simultaneous services with different priorities, timeliness and reliability guarantees.

IV. Conclusion

We present the digital cluster circuit (DCC): a practical system model/design which exploits programmable network devices to establish reliable interactions in a subset of the datacenter. Like in classical digital circuits, DCC nodes (i.e. smartNICs, programmable switches) receive a periodic clock signal in the form of ultra-low jitter packets. This allows for robust timeouts, enabling a new class of reliable and efficient services spanning best-effort applications and DCC modules.

References

[1] I. Calciu, M. T. Imran, I. Puddu, S. Kashyap, H. A. Maruf, O. Mutlu, and A. Kolli, "Rethinking software runtimes for disaggregated memory," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '21, (New York, NY, USA), p. 79–92, Association for Computing Machinery, 2021.

[2] Forbes Technology Council, "The true cost of downtime and how to avoid it," 2024. Accessed: 2025-02-06.

[3] L. Barroso, M. Marty, D. Patterson, and P. Ranganathan, "Attack of the killer microseconds," *Commun. ACM*, vol. 60, p. 48–54, Mar. 2017.

[4] M. K. Aguilera, N. Ben-David, R. Guerraoui, V. J. Marathe, A. Xygkis, and I. Zablotchi, "Microsecond consensus for microsecond applications," Nov. 2020.

[5] C. Kachris and C. Z. Patrikakis, "The rise of accelerator-based data centers: Opportunities and challenges," *IT Professional*, vol. 26, no. 6, pp. 4–9, 2024.

[6] Apache Software Foundation, "Apache Kafka." https://kafka.apache.org/, 2011. Accessed: 2025-02-04.

[7] M. Alimadadi, H. Mai, S. Cho, M. Ferdman, P. Milder, and S. Mu, "Waverunner: An elegant approach to hardware acceleration of state machine replication," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, (Boston, MA), pp. 357–374, USENIX Association, Apr. 2023.

[8] R. Guerraoui, A. Murat, J. Picorel, A. Xygkis, H. Yan, and P. Zuo, "uKharon: A membership service for microsecond applications," in *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, pp. 101–120, July 2022.

[9] K. Lim, J. Chang, T. Mudge, P. Ranganathan, S. K. Reinhardt, and T. F. Wenisch, "Disaggregated memory for expansion and sharing in blade servers," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, p. 267–278, 2009.

[10] M. Ewais and P. Chow, "Disaggregated memory in the datacenter: A survey," *IEEE Access*, vol. 11, pp. 20688–20712, 2023.

[11] Y. Lee, H. A. Maruf, M. Chowdhury, A. Cidon, and K. G. Shin, "Hydra : Resilient and highly available remote memory," in *20th USENIX Conference on File and Storage Technologies (FAST 22)*, pp. 181–198, Feb. 2022.

[12] R. Schlichting and F. Schneider, "Fail-stop processors: An approach to designing fault-tolerant computing systems," *ACM Transactions on Computer Systems (TOCS)*, vol. 1, no. 3, pp. 222–238, 1983.

[13] Z. Hu, P. Zuo, Y. Chen, C. Wang, J. Hu, and M.-C. Yang, "Aceso: Achieving efficient fault tolerance in memory-disaggregated key-value stores," in *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, SOSP '24, p. 127–143, 2024.

[14] N. Zilberman, M. Grosvenor, D. A. Popescu, N. Manihatty-Bojan, G. Antichi, M. Wójcik, and A. W. Moore, "Where has my time gone?," in *Passive and Active Measurement: 18th International Conference, PAM 2017, Sydney, NSW, Australia, March 30-31, 2017, Proceedings 18*, pp. 201–214, Springer, 2017.

[15] I. Zhang, A. Raybuck, P. Patel, K. Olynyk, J. Nelson, O. S. N. Leija, A. Martinez, J. Liu, A. K. Simpson, S. Jayakar, P. H. Penna, M. Demoulin, P. Choudhury, and A. Badam, "The demikernel datapath os architecture for microsecond-scale datacenter systems," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, SOSP '21, (New York, NY, USA), p. 195–211, Association for Computing Machinery, 2021.

[16] A. Kalia, M. Kaminsky, and D. Andersen, "Datacenter RPCs can be general and fast," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, (Boston, MA), pp. 1–16, USENIX Association, Feb. 2019.

[17] P. Jahnke, V. Riesop, P.-L. Roman, P. Chuprikov, and P. Eugster, "Live in the express lane," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pp. 581–597, July 2021.

[18] F. Reghenzani, G. Massari, and W. Fornaciari, "The real-time linux kernel: A survey on preempt_rt," *ACM Comput. Surv.*, vol. 52, feb 2019.

[19] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *J. ACM*, vol. 32, p. 374–382, apr 1985.

[20] T. Yang, R. Gifford, A. Haeberlen, and L. T. X. Phan, "The synchronous data center," in *HotOS '19: Workshop on Hot Topics in Operating Systems*, ACM, 5 2019.

[21] P. Verissimo and A. Casimiro, "The timely computing base model and architecture," *IEEE Transactions on Computers*, vol. 51, no. 8, pp. 916–930, 2002.

[22] K. A, V. Kumar, P. V. Prasanth, R. Dutta, B. Roy, and P. Chakraborty, "Fpgas as hardware accelerators in data centers: A survey from the data centric perspective," in *2024 2nd International Conference on Device Intelligence, Computing and Communication Technologies (DICCT)*, pp. 1–6, 2024.

[23] T. Ziegler, J. Nelson-Slivon, V. Leis, and C. Binnig, "Design guidelines for correct, efficient, and scalable synchronization using one-sided rdma," *Proceedings of the ACM on Management of Data*, vol. 1, pp. 131:1–131:26, June 2023.

[24] "Time-Sensitive Netoworking (TSN) Task Group." Online; accessed 10-Jan-2025.

[25] M. P. Grosvenor, M. Schwarzkopf, I. Gog, R. N. M. Watson, A. W. Moore, S. Hand, and J. Crowcroft, "Queues Don't matter when you can JUMP them!," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, (Oakland, CA), pp. 1–14, USENIX Association, May 2015.

[26] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: a centralized "zero-queue" datacenter network," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, (New York, NY, USA), p. 307–318, Association for Computing Machinery, 2014.

[27] IEEE, "IEEE Standard for Local and metropolitan area networks–Frame Replication and Elimination for Reliability," *IEEE Std 802.1CB-2017*, pp. 1–102, 2017.

[28] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," in *Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*, pp. 87–95, 2014.

[29] M. S. Brunella, G. Belocchi, M. Bonola, S. Pontarelli, G. Siracusano, G. Bianchi, A. Cammarano, A. Palumbo, L. Petrucci, and R. Bifulco, "hxdp," *Communications of the ACM*, vol. 65, pp. 92–100, 8 2022.

[30] W. Xu, Z. Zhang, Y. Feng, H. Song, Z. Chen, W. Wu, G. Liu, Y. Zhang, S. Liu, Z. Tian, and B. Liu, "Clickinc: In-network computing as a service in heterogeneous programmable data-center networks," in *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, (New York, NY, USA), pp. 798–815, Association for Computing Machinery, September 2023.

[31] V. Liu, D. Halperin, A. Krishnamurthy, and T. Anderson, "F10: A Fault-Tolerant engineered network," in *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, (Lombard, IL), pp. 399–412, USENIX Association, Apr. 2013.

[32] J. B. Leners, H. Wu, W.-L. Hung, M. K. Aguilera, and M. Walfish, "Detecting failures in distributed systems with the falcon spy network," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, p. 279–294, 2011.

[33] P. Huang, C. Guo, L. Zhou, J. R. Lorch, Y. Dang, M. Chintalapati, and R. Yao, "Gray failure: The achilles' heel of cloud-scale systems," in *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*, HotOS '17, (New York, NY, USA), p. 150–155, Association for Computing Machinery, 2017.

[34] J. Stone and C. Partridge, "When the crc and tcp checksum disagree," *SIGCOMM Comput. Commun. Rev.*, vol. 30, p. 309–319, aug 2000.

[35] S. Liu, P. Viotti, C. Cachin, V. Quéma, and M. Vukolic, "XFT: practical fault tolerance beyond crashes," in *12th USENIX Symposium on Operating Systems Design and Implementation, (OSDI '16')*, pp. 485–500, 2016.

[36] M. Raynal, "Consensus in synchronous systems: a concise guided tour," in *2002 Pacific Rim International Symposium on Dependable Computing, 2002. Proceedings.*, pp. 221–228, December 2002.