Integrated Strategies for Bottleneck Mitigation in SDN: A Comprehensive Analysis of Caching, Queuing, and Network Coding

Souryendu Das[1*], Jacob Adamson[2*], Aaron Lee[2*], Vedant Vaideswar[2*], Stavros Kalafatis[3*]
[*]*Electrical and Computer Engineering*, *Texas A&M University*, College Station, USA
Email: souryendu@gmail.com, adamson1872@tamu.edu, aaron1145@tamu.edu,
vedant_vaideswar@tamu.edu, skalafatis-tamu@tamu.edu

***Abstract*—This study explores the application of Software Defined Networking (SDN) to optimize data center network performance by employing a series of innovative techniques tested within a Mininet-emulated dumbbell network topology. We precisely assess network behavior under different protocols and configurations, including IPerf3 in both Cubic and Bottleneck Bandwidth and Round-trip propagation time (BBR) modes, as well as User Datagram Protocol Speed Test (UDPST) mode. Results highlight IPerf3 in BBR mode as superior in stability and performance. Additionally, we investigate various caching architectures—singular, split, and multi-cache—with multi-cache configurations demonstrating the best performance in terms of capacity and access time. Furthermore, network coding strategies like XOR coding and Random Packet Spraying (RPS) are analyzed for their effectiveness in reducing packet loss and retransmissions, with RPS showing a notable decrease in retransmission rates. Collectively, these insights contribute significantly to SDN implementations aimed at enhancing the operational efficiency of data centers.**

***Index Terms*—Software Defined Networking, Data Center Networks, Mininet Emulation, Network Performance, Speed Testing, Caching Architectures, Network Coding, XOR Coding, Random Packet Spraying, Data Transmission.**

I. Introduction

Modern data centers face growing traffic volumes and stringent performance requirements. Traditional network infrastructures often struggle with congestion, latency, and packet loss under bottlenecks, cross-traffic, and resource contention [1], [2], [15]. Software Defined Networking (SDN) addresses these limitations by decoupling the control and data planes, enabling centralized control and flexible deployment of optimization mechanisms [3], [16].

Previous studies [12]–[14] investigated SDN-based subsystems such as *speed testing*, *caching architectures*, and *network coding* in Mininet dumbbell topologies. They showed, for example, that TCP BBR can outperform TCP Cubic and UDP-based methods, that multi-cache schemes improve latency and cache hit rate, and that XOR coding and Random Packet Spraying (RPS) enhance packet delivery reliability. However, these components were evaluated largely in isolation and on simplified topologies.

More recent SDN work has leveraged advanced queuing disciplines and caching to further improve performance. For instance, [17] studied priority queuing in SDN, while Cohen et al. [10] integrated adaptive network coding into SDN to improve throughput and reliability in high-demand scenarios.

In this work, we extend prior efforts by integrating multiple mechanisms into a single SDN-controlled testbed and explicitly studying their combined behavior under controlled bottleneck conditions:

- We propose an **integrated system** that combines *speed testing*, *advanced caching*, *network coding*, and *queuing disciplines* within one SDN framework.
- We introduce an **8-host topology** that simultaneously supports database-driven caching, packet delivery optimization, and queuing experiments, including cross-traffic and speed tests.
- We integrate queuing disciplines such as **FQ CoDel, TCPIE, RED, and BFIFO** to minimize packet loss and latency under congestion [18], [19].
- We combine **XOR Coding** with **RPS**, adding redundancy to improve resilience against packet loss [20].

Our system models a data-center-inspired, database-driven query–response workload hosted on SQLite servers. The workload captures key DCN characteristics (short exchanges, skewed popularity, and sharded datasets), though it is not trace-driven. By introducing probabilistic redundancy and randomized routing, we examine trade-offs between resource usage and reliability.

The contributions of this work are:

1) **Integrated SDN evaluation framework:** A unified SDN controller and Mininet testbed that jointly evaluates caching, queuing, and coding under a database-driven query–response workload. Prior studies analyze these mechanisms separately; here they are examined in combination.
2) **Database-backed 8-host topology with custom headers:** Packets carry structured question–answer payloads mapped to distributed SQLite databases. A custom TCP header enables flow steering and random server selection, creating realistic caching pressure and reproducible bottlenecks.
3) **Cross-layer performance insights and trade-offs:** We expose trade-offs that are not visible in isolated studies—for example, the strength of BFIFO under con-

gestion, the reliability–bandwidth trade-off of XOR+RPS, and latency gains from hierarchical caching around bottlenecks.

4) **Implementation-driven observations for practitioners:** We show that combining lightweight queueing, targeted redundancy, and hierarchical caching can reduce packet loss by 96.46% and improve completion time by ≈11% in SDN-controlled data center networks, providing actionable deployment guidelines.

The remainder of this paper is organized as follows: Section II reviews related work. Section III presents the system model and experimental setup. Section IV details the queuing methodology. Section V reports results and analysis. Section VI concludes with key insights and future work.

II. Related Work

SDN has emerged as a key enabler for addressing DCN challenges such as congestion, latency, and packet loss. The survey in [1] outlines performance issues and solutions in SDN-based data centers, while [2], [3] discuss load balancing and resource management in SDN-enabled environments.

Several works use SDN to improve congestion control and traffic management. For example, [6] proposed E-DCTCP, an SDN-assisted congestion control scheme that leverages global network visibility to adjust window sizes and enhance throughput. Intelligent load-balancing techniques within SDN frameworks are surveyed in [2], highlighting dynamic traffic distribution to mitigate bottlenecks.

Queuing disciplines also play a central role. Reticcioli et al. [7] used machine learning to control priority queues in SDN, improving bandwidth allocation and reducing packet loss. Montazerolghaem [3] examined advanced queuing disciplines such as FQ CoDel and TC-PIE, demonstrating their effectiveness in reducing bufferbloat and improving throughput.

In-network caching has been explored as a way to reduce latency and bandwidth usage. Chen et al. [8] proposed an integrated SDN-based framework for managing networking, caching, and computing resources. Addya et al. [9] introduced a hybrid queuing model for virtual machine placement in cloud data centers, while [4] examined predictive caching based on historical traffic patterns.

Network coding integrated into SDN has also been studied. Cohen et al. [10] considered controller-based adaptive network coding in heterogeneous networks, improving throughput and resilience. Janczukowicz [5] examined random linear coding in WebRTC contexts, showing improved reliability over conventional methods.

Hybrid optimization strategies have been proposed to combine caching, queuing, and coding. Li et al. [11] introduced FDRC, a flow-driven rule-caching algorithm that balances cache size constraints with flow predictability. Our earlier work [12]–[14] examined speed testing, caching, and network coding individually in SDN settings. Building on these foundations, the present work integrates queuing, caching, and coding within one SDN framework and evaluates them jointly in an 8-host database-driven topology.

III. System Model

This study investigates bottleneck mitigation strategies in a Mininet dumbbell topology where bottlenecks arise between Switch 1 and Switch 2 due to bandwidth limits, latency, and drops. We examine how speed tests, queuing mechanisms, caching strategies, and coding schemes interact under these conditions.

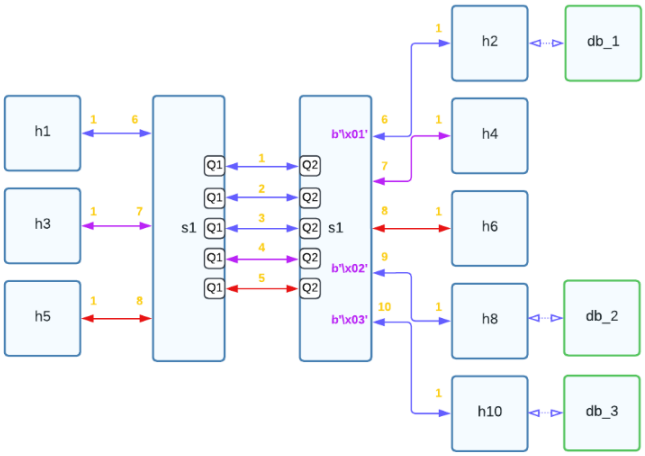

Figure 1: Mininet Dumbbell Network Topology.

A. Network Topology

The topology (Fig. 1) comprises 8 hosts with distinct roles:

- **Caching and coding path (blue)**: `h1` → `s1` → `s2` → {`h2`, `h8`, `h10`}.
- **Speed tests (pink)**: `h3` → `h4`.
- **Cross-traffic (red)**: `h5` → `h6`.

Packets from $h1$ carry questions as payloads. Servers $h2$, $h8$, and $h10$ host SQLite databases, each storing a disjoint subset of 100 question–answer pairs ($h2$: 1–33, $h8$: 34–66, $h10$: 67–100). The destination server is chosen using a random field in a custom TCP header, modeling database selection across shards. The sequence of questions is random but popularity-weighted, generating realistic cache pressure where caches store questions as keys and answers as values. The custom TCP header is illustrated in Fig. 2.

20 bytes	2 bytes	1 byte
Normal TCP Header	Kind: b'\x254'	Value: b'\x01' (maps to h2) b'\x02' (maps to h8) b'\x03' (maps to h10)

Figure 2: Custom Packet TCP Header.

B. Speed Testing Subsystem

We employ three speed-testing methods:

- **Iperf3** in TCP mode, with:
  - **Cubic**: the default Linux algorithm, adjusting rate based on RTT and loss.

  - **BBR**: a bandwidth- and delay-based algorithm maximizing throughput while limiting latency [21].
- **UDPST**: a UDP-based speed test that can generate high offered rates at the cost of higher potential loss.

C. Caching Subsystem

Caching uses an LRFU-like scheme:

- Each cache entry has a sticky counter incremented on hits and decremented on misses; entries with negative counters are evicted.
- Singular, split, and multi-cache configurations are evaluated to balance capacity and latency, following [12].

D. Network Coding Subsystem

To improve reliability and reduce retransmissions, we apply:

- **XOR Coding + RPS**: Packets are split into three fragments: two data fragments and one XOR parity. Multiple copies are sprayed randomly over available paths. Upon receipt of any two fragments, the packet is reconstructed and redundant fragments are discarded. A hash table tracks fragment tags and associated state.
- **Multiple paths**: Additional links increase path diversity, reducing congestion and improving resilience to link loss.

E. Queuing Subsystem

Queuing disciplines are applied to reduce packet loss:

- **Configuration**: Queues are configured on the bottleneck link between Switch 1 and Switch 2 in both directions.
- **Disciplines**: We test FQ CoDel, TC-PIE, RED, and BFIFO and compare their ability to mitigate loss under congestion.

This topology allows simultaneous evaluation of caching, coding, speed testing, and queuing over the same bottleneck, revealing cross-layer interactions.

F. Experimental Environment

Experiments use Mininet 2.3.0 on Ubuntu 20.04 LTS with an Intel Xeon 8-core CPU and 16 GB RAM. The SDN controller is implemented in Python using Ryu 4.34 with support for custom headers, caching, and coding.

Traffic generation uses IPerf3 3.13 for TCP/UDP and UDPST for additional UDP workloads. SQLite 3.39 runs on multiple hosts to emulate database-driven queries. Each experiment is repeated five times; we report averages to smooth transient fluctuations.

The bottleneck link between Switch 1 and Switch 2 is set to 10 Mbps and 10 ms delay, with all other links at 100 Mbps and 1 ms delay. Queue buffer sizes on the bottleneck are capped at 119,000 bytes. Randomization seeds are fixed across runs to ensure comparability while preserving stochastic behavior.

For brevity, we do not list every configuration parameter in tables. All queues use the Linux `tc` implementation on Ubuntu 20.04, with discipline-specific parameters left at distribution defaults unless otherwise noted, and a 119,000-byte bottleneck queue. IPerf3 and UDPST use consistent command-line options across runs (e.g., a 100 Mbps UDP offered load over the 10 Mbps bottleneck). The complete set of `tc` commands, Mininet scripts, and traffic-generation parameters will be released with the controller as an open-source artifact to support exact reproduction. Unlike prior SDN studies that isolate caching, queuing, or coding, our design runs all subsystems simultaneously over the same bottleneck link.

Although Mininet limits scale, the controller logic, custom headers, database flows, and coding pipeline are topology-agnostic and can be deployed on larger testbeds such as MaxiNet or CloudLab, which we identify as future scalability targets.

G. Scope and Design Rationale

Our goal is to study the joint behavior of caching, queuing, and coding under controlled bottleneck conditions. We deliberately employ a small but structured topology to fix bottleneck placement, generate repeatable database workloads, and maintain deterministic queuing behavior. While not large-scale, this setting serves as a methodological foundation for larger topologies. It is representative of database-backed services such as key–value stores or FAQ/query systems in data centers, where many short-lived queries share limited resources. We do not claim trace-level realism, but rather a controlled, DCN-inspired proxy.

IV. Queuing Methodology

A. Queuing System Introduction

We focus on four queuing disciplines:

- **RED (Random Early Detection)**: randomly drops packets before queues overflow.
- **TC-PIE (Proportional Integral controller Enhanced)**: adjusts drop probability to control queueing delay and mitigate bufferbloat.
- **FQ CoDel (Fair Queuing Controlled Delay)**: combines flow queuing with CoDel to control latency while maintaining high throughput.
- **BFIFO (Byte First In First Out)**: a simple byte-based FIFO dropping packets when the queue is full.

We compare their effectiveness at reducing loss under congestion.

B. Queuing System Details

The queuing experiments proceed as follows:

1) **Baseline**: IPerf3 in UDP mode sends at 100 Mbps while the bottleneck link is 10 Mbps, creating congestion.
2) **No-queuing case**: The queue is set to PFIFO with capacity of one packet, effectively disabling buffering. This provides a loss baseline.
3) **Queuing cases**: Each discipline (RED, TC-PIE, FQ CoDel, BFIFO) is applied with a 119,000-byte bottleneck buffer. Over 45 MB of data is sent in each test and loss is measured on the bottleneck link.

Loss percentages are computed and compared to the baseline. In this initial study, all disciplines use distribution-default parameters on the bottleneck link (subject to the 119,000-byte

cap). A full sensitivity analysis—including per-link tuning and parameter sweeps for RED thresholds, FQ CoDel target/interval, and TC-PIE controller settings—is left for future work.

V. Results and Analysis

All figures use descriptive axis labels and consistent scaling across queuing and integrated experiments to ease comparison.

A. Queuing System Results

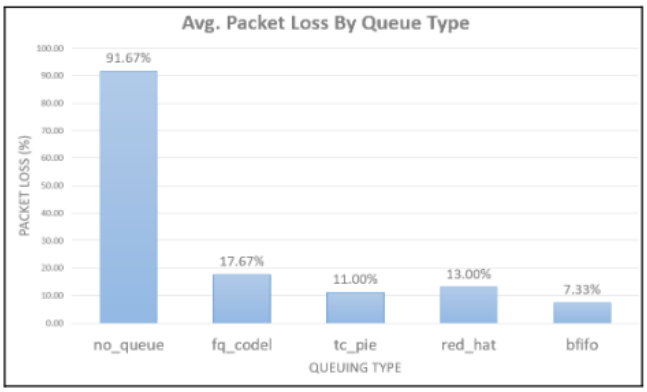

Figure 3: Average Packet Loss by Queuing Type.

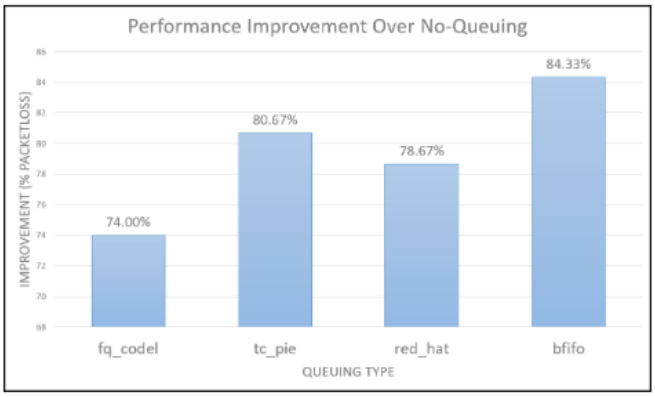

Figure 4: Average Performance Increase over No-Queuing.

With no queuing, packet loss is **91.67%**, as expected given a 100 Mbps offered rate over a 10 Mbps link. All four disciplines substantially reduce loss:

- **FQ CoDel**: loss reduced to **17%**.
- **BFIFO**: best performance with **7.33%** loss on average.
- **TC-PIE**: close to BFIFO, with strong loss reduction.
- **RED**: moderate improvement relative to the baseline.

BFIFO and TC-PIE yield the largest improvement over no-queuing, with FQ CoDel and RED also providing substantial gains. BFIFO's strong performance is notable given its simplicity: in our bursty UDP workload, predictable byte-ordered dropping avoids head-of-line blocking and stabilizes loss rates. TC-PIE provides comparable loss reduction while explicitly regulating delay through its PI controller.

B. Integrated System Results

We now compare the integrated controller to a base controller. The base controller simply forwards packets to their destinations without caching, XOR coding, or RPS. The integrated controller enables caching, XOR+RPS, and queuing. For each controller, we send 50,000 custom packets and measure packet loss, completion time, data transmitted, and cache hits.

The performance of the integrated controller arises from interactions among subsystems: caching reduces load on the bottleneck, coding reduces retransmissions under loss, and queuing stabilizes congestion. Together, these effects yield significantly lower loss despite redundancy from XOR+RPS.

1) Packet Loss Reduction: As shown in Fig. 5 and Tables I–II, with a configured drop rate of 5% the integrated system loses only **194.2** packets on average (**0.388%**), compared to **5480.4** packets (**10.96%**) in the base controller. This reduction highlights the effectiveness of XOR+RPS and caching in enhancing reliability.

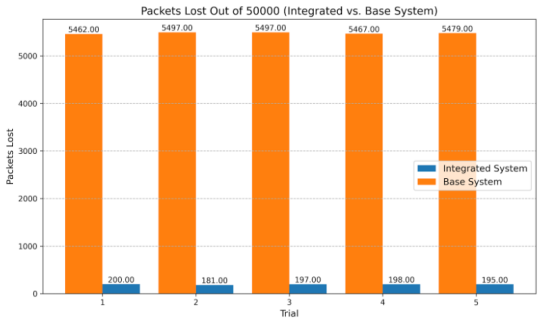

Figure 5: Packets Lost Out of 50,000 (Integrated vs. Base System).

Trial #	Test Time (s)	Packets Sent	Packets Received	Packets Lost
1	2672	55462	50000	5462
2	2711	55497	50000	5497
3	2965	55497	50000	5497
4	3031	55467	50000	5467
5	2767	55479	50000	5479
Average	**2829.2**	**55480.4**	**50000**	**5480.4 (10.96%)**

Table I: Base Controller Summarized Results.

Trial #	Test Time (s)	Packets Sent	Packets Received	Packets Lost
1	2630	50200	50000	200
2	2533	50181	50000	181
3	2477	50197	50000	197
4	2430	50198	50000	198
5	2481	50195	50000	195
Average	**2510.2**	**50194.2**	**50000**	**194.2 (0.388%)**

Table II: Integrated Controller Summarized Results.

2) Speed Improvements: The integrated system also reduces completion time by about **11%**, as shown in Fig. 6. Fewer retransmissions and cache hits near the bottleneck and databases both contribute to this improvement: M1 cache hits avoid crossing the bottleneck, while deeper cache hits avoid database reads [12].

3) Caching Efficiency: Fig. 7 shows M1 and M3 cache hits out of 50,000 packets. M1 cache hits reduce traffic over the bottleneck, and M3 hits reduce database workloads. The

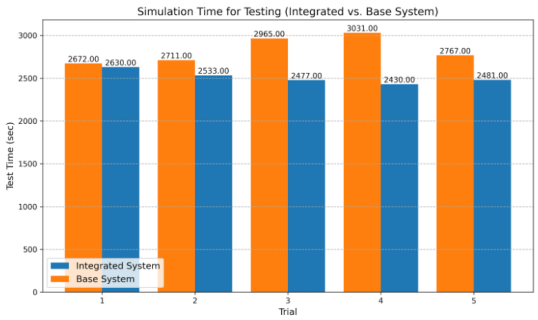

Figure 6: Simulation Time (Integrated vs. Base System).

hierarchical caching design thus improves both throughput and latency.

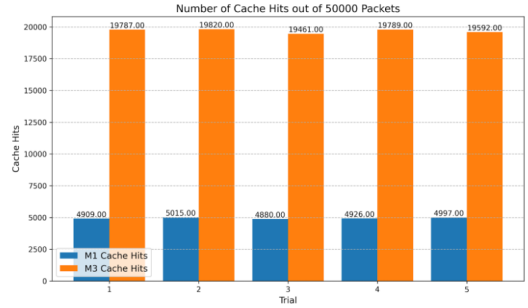

Figure 7: Cache Hits Out of 50,000 Packets.

4) Data Transmission Overhead: Fig. 8 shows that the integrated system transmits roughly twice as much data as the base system across the bottleneck, due to redundancy from XOR+RPS. In our prototype, this redundancy level is fixed (two data fragments plus one XOR fragment sprayed across multiple paths). Exploring tunable redundancy levels and adaptive spraying policies—for example, scaling parity with observed loss—is left as future work to more systematically map the reliability–bandwidth trade-off.

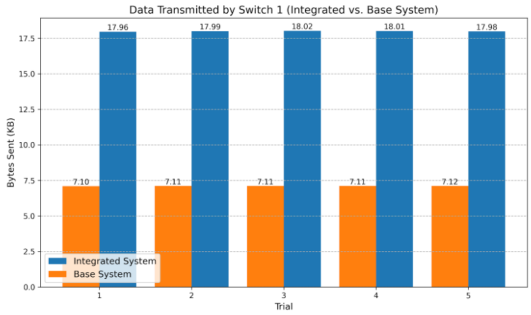

Figure 8: Data Transmitted by Switch 1 (Integrated vs. Base System).

Overall, the integrated system substantially improves packet loss and completion time relative to the base controller. Caching reduces retransmissions and database queries, while XOR+RPS improves delivery reliability. The cost is increased bandwidth usage, which future work will seek to optimize via redundancy tuning and further caching enhancements.

C. Discussion on Combined Results

The results above reflect the combined effect of caching, queuing, XOR coding, and RPS; we do not isolate each subsystem's contribution. While this integrated view highlights the benefits of combining techniques, it limits visibility into the relative impact of each mechanism. A full ablation study—starting from the base controller and progressively enabling caching, queuing, and coding individually and in combination—is beyond the scope of this paper due to space and runtime constraints. Designing and reporting such a study is part of our planned future work.

D. Reproducibility Considerations

Our experiments use synthetic workloads generated via randomized SQLite queries and custom packet payloads. While this setup is DCN-inspired, incorporating public, trace-driven workloads from operational data centers would further strengthen realism and external validity. All experiments were repeated five times with fixed random seeds, and we report averages; including standard deviations or confidence intervals in an extended version would improve statistical rigor. Future work will therefore (i) integrate trace-driven workloads, (ii) report more detailed statistics, and (iii) release the Mininet controller and scripts as an open-source artifact.

VI. Conclusion

We presented an integrated SDN-enabled system that jointly applies caching, queuing, and network coding to mitigate bottlenecks in a database-driven Mininet environment. Rather than proposing a single new primitive, our contribution is to unify established techniques within a common 8-host testbed and empirically characterize their trade-offs.

Our experiments yield several observations:

- **Reliability vs. bandwidth**: Combining XOR coding with RPS significantly improves reliability. For a 5% configured drop rate, packet loss is reduced from 10.96% in the base controller to 0.388% in the integrated system, at the cost of approximately doubling bottleneck traffic. XOR+RPS is thus attractive for loss-sensitive flows where bandwidth is less constrained. Our current results correspond to a single fixed redundancy setting; future work will sweep coding rates and spraying probabilities to fully map the reliability–bandwidth trade-off.
- **Caching vs. latency and database load**: Hierarchical caching around the bottleneck and database servers reduces retransmissions and shortens paths for popular queries. M1 caches prevent traffic from crossing the bottleneck, while deeper caches avoid database lookups. Together with coding, this yields an $\approx$11% speedup in completion time, indicating that strategic cache placement can provide tangible latency and load benefits even in the presence of coding redundancy.
- **Queue management vs. complexity**: Under UDP stress tests, all queuing disciplines (RED, FQ CoDel, TC-PIE, BFIFO) significantly reduce loss relative to the no-queuing baseline. BFIFO—a simple byte-based FIFO—achieves

the lowest observed loss (7.33%), with TC-PIE close behind, while FQ CoDel and RED still provide strong benefits. This suggests that a carefully sized BFIFO queue can capture much of the benefit of more complex AQMs in some SDN settings, while TC-PIE may be preferable where tighter delay control and bufferbloat mitigation are critical.

For SDN-controlled data center networks with database-backed applications, our findings suggest a practical recipe: (i) configure BFIFO or TC-PIE on known bottleneck links, (ii) deploy a hierarchical caching layer around bottlenecks and database servers, and (iii) selectively enable XOR+RPS on loss-prone flows or paths. This combination can reduce packet loss and completion times at the cost of additional bandwidth, which can be managed via tunable redundancy.

This work has several limitations. We focus on a small Mininet topology, synthetic query–response workloads, and metrics centered on loss and completion time; we do not yet examine throughput distributions, latency distributions, or controller overhead in depth. We also defer a full ablation study and queue-parameter sensitivity analysis. Future work will extend the framework to larger and more heterogeneous topologies (e.g., MaxiNet, CloudLab), incorporate trace-driven traffic, and release the code and scripts as an open-source artifact.

Overall, the originality of this paper lies in **integrating and jointly analyzing caching, queuing, and coding within an SDN-enabled, database-driven testbed** and in empirically characterizing their interactions under controlled bottleneck conditions. We hope this framework and its insights serve both as a template for future studies and as a practical guide for SDN practitioners engineering more resilient and efficient bottleneck links in data center networks.

REFERENCES

[1] Shirmarz, A., and Ghaffari, A., "Performance issues and solutions in SDN-based data center: a survey", *The Journal of Supercomputing*, 76(10), 7545-7593, 2020

[2] Saxena, M.C., Sabharwal, M. and Bajaj, P., "Review of SDN-based load-balancing methods, issues, challenges, and roadmap.", *International journal of electrical and computer engineering systems*, 14(9), pp.1031-1049, 2023

[3] Montazerolghaem, A., "Software-defined load-balanced data center: design, implementation and performance analysis.", *Cluster Computing*, 24(2), pp.591-610, 2021

[4] Doyle, R., "Testing the Effectiveness of AQMs at Improving Network Performance", 2015.

[5] Janczukowicz, EC, "QoS management for WebRTC: loose coupling strategies", *(Doctoral dissertation, Ecole nationale supérieure Mines-Télécom Atlantique)*, 2017

[6] Xu, J., Pan, W., Tan, H. and Cheng, L., "A TCP Congestion Control Optimization Method for SDN-Based Data Center Networks." *In 2024 9th International Conference on Computer and Communication Systems (ICCCS)* (pp. 468-473). IEEE, 2024

[7] Reticcioli, E., Di Girolamo, G.D., Smarra, F., Torzi, A., Graziosi, F. and D'innocenzo, A., "Modeling and Control of Priority Queueing in Software Defined Networks via Machine Learning." *IEEE Access*, 10, pp.91481-91496, 2022

[8] Chen, Q., Yu, F.R., Huang, T., Xie, R., Liu, J. and Liu, Y., "Integrated Resource Management in Software Defined Networking, Caching and Computing." *arXiv preprint arXiv:1611.05122*, 2016.

[9] Addya, S.K., Turuk, A.K., Sahoo, B. and Sarkar, M., "A hybrid queuing model for virtual machine placement in cloud data center." *In 2015 IEEE international conference on advanced networks and telecommuncations systems (ANTS)* (pp. 1-3). IEEE, 2015

[10] Cohen, A., Esfahanizadeh, H., Sousa, B., Vilela, J.P., Luís, M., Raposo, D., Michel, F., Sargento, S. and Médard, M., "Bringing network coding into SDN: a case-study for highly meshed heterogeneous communications." *arXiv preprint arXiv:2010.00343*, 2020

[11] Li, H., Guo, S., Wu, C. and Li, J., "FDRC: Flow-driven rule caching optimization in software defined networking.", *In 2015 IEEE International Conference on Communications (ICC)* (pp. 5777-5782). IEEE, 2015

[12] S. Das, J. Adamson, A. Lee, V. Vaideswar and S. Kalafatis, "Optimizing Data Center Network Performance: A Comprehensive Analysis of Speed Testing, Caching, and Network Coding in Software Defined Networking," *2024 20th International Conference on Network and Service Management (CNSM), Prague, Czech Republic, 2024*, pp. 1-9.

[13] S. Das and S. Kalafatis, "Speed Testing for Measuring Network Traffic in a Smart Network Switch," *2024 International Conference on Computing, Networking and Communications (ICNC), Big Island, HI, USA, 2024*, pp. 446-450.

[14] S. Das and S. Kalafatis, "Network Coding to Reduce Congestion and Improve Memory Buffer in Smart Switch," *2023 International Symposium on Networks, Computers and Communications (ISNCC), Doha, Qatar*, 2023, pp. 1-6.

[15] Shirmarz, A. and Ghaffari, A., "An adaptive greedy flow routing algorithm for performance improvement in software-defined network", *International Journal of Numerical Modelling: Electronic Networks, Devices and Fields*, 33(1), p.e2676, 2020

[16] Montazerolghaem, A., "Software-defined internet of multimedia things: energy-efficient and load-balanced resource management", *IEEE Internet of Things Journal*, 9(3), pp.2432-2442, 2021.

[17] Shen, G., Li, Q., Shi, W., Jiang, Y., Zhang, P., Gu, L. and Xu, M., "Modeling and optimization of the data plane in the SDN-based DCN by queuing theory", *Journal of Network and Computer Applications*, 207, p.103481, 2022

[18] https://www.man7.org/linux/man-pages/man8/tc-fq_codel.8.html

[19] Chen, X.R., Liu, X.P. and Yu, A.L., "An Integrated Queuing Network Model for Optimizing Multi-Level AVS/RS Performance in Multi-Floor Manufacturing Environments", *IEEE Access*, 2024.

[20] Huang, J., Wang, S., Li, S., Zou, S., Hu, J. and Wang, J., "HTPC: heterogeneous traffic-aware partition coding for random packet spraying in data center networks", *J Cloud Comp* 10, 31 (2021).

[21] Cardwell, N., Cheng, Y., Gunn, S.C., Yeganeh, S., and Jacobson, V., "BBR: Congestion-based congestion control", *Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pp.523-537, 2017