Optimizing Data Center Network Performance: A Comprehensive Analysis of Speed Testing, Caching, and Network Coding in Software Defined Networking

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 UDPST (User Datagram Protocol Speed Test). 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

As data-intensive applications and services increase in today's network landscape, traditional network infrastructures are increasingly challenged by high data volumes and the dynamic nature of modern computing environments [21], [24]. This scenario is particularly evident in data centers, where efficient data routing and load balancing are critical to avoiding performance bottlenecks and managing operational costs effectively [19], [22]. Software Defined Networking (SDN) has emerged as a transformative approach in this context, decoupling the control plane from the data plane to offer a more flexible and centralized management system [25]. This paradigm shift facilitates rapid adjustments to network configurations and seamless integration of services, effectively addressing the limitations of traditional networks [23], [26].

Our research leverages the Mininet emulator to investigate how SDN can enhance data center operations through innovative caching and network coding strategies, testing these strategies in various cross-traffic scenarios to evaluate different network speed testing methodologies (TCP Cubic, TCP BBR, and OB-UDPST) under multiple bottleneck conditions [11], [18]. This approach aims to improve overall network performance by optimizing data transmissions between hosts within a controlled, emulated environment [2], [9].

Despite the advantages of SDN, efficiently managing high-speed data traffic remains a significant challenge [30]. This research not only explores these challenges but also proposes solutions tailored for data center network optimizations [7], [8]. We focus on the design, implementation, and evaluation of an SDN solution crafted explicitly for a dumbbell network topology, a model that effectively represents common network scenarios with bottleneck issues [3], [4]. The study develops custom caching architectures for packet transmissions between network switches and implements diverse network coding schemes to reduce latency and enhance throughput [10], [12].

Moreover, SDN controllers play a crucial role in managing network resources efficiently, where advanced algorithms help distribute traffic dynamically, enhancing the robustness and responsiveness of network infrastructures [13]–[16]. Techniques such as load balancing, congestion control, and data flow prioritization are critical in achieving these goals, especially in environments characterized by heavy data exchange and real time processing requirements [17], [20].

Network tools and switches, although increasingly sophisticated, still face challenges related to hardware limitations and the need for integration with legacy systems. Studies have shown that strategic placement of SDN controllers and the adoption of intelligent routing protocols can significantly mitigate these issues, providing more stable and efficient network operations [1], [5], [6], [27].

This work is intended to contribute valuable insights into network performance optimizations, which are particularly useful for further research and practical applications. While our solutions are designed for a dumbbell network topology, they offer foundational strategies that could be adapted for broader network configurations [28]. The findings from this project will provide a significant reference point for future researchers seeking to optimize network configurations effectively.

However, the use of Mininet as an emulation tool comes with inherent limitations that must be considered:

- **Scalability of Emulated Network:** Mininet runs on a single host machine, which restricts the size and complexity of the network that can be emulated. Consequently, larger topologies may suffer from reduced performance due to resource constraints [27].
- **Physical Network Conditions:** Mininet does not simulate physical characteristics and environmental factors, such as interference and signal degradation, which can significantly impact real-world network performance [19], [20].

II. System Model

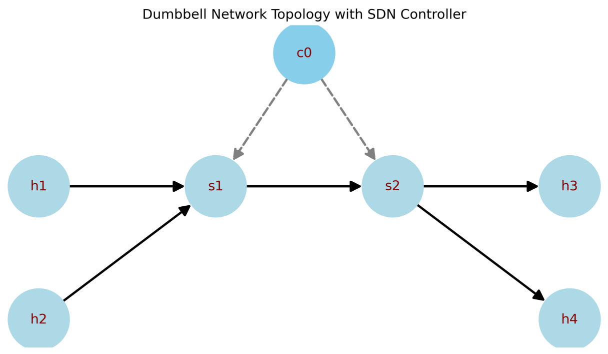


Figure 1: Dumbbell Network Topology with SDN Controller.

We employ a Mininet-based dumbbell network topology managed by an SDN controller (c0), connecting four hosts (h1, h2, h3, h4) through two switches (s1, s2), as shown in Figure 1. This topology facilitates precise control and monitoring of traffic flows and is essential for testing network coding, caching strategies, and speed testing under various conditions. Our research aims to address specifically network traffic that is congested within a bottleneck; that is, we are using the dumbbell topology due to its inherent design, which provides this bottleneck while not introducing other network complexities. The goal is to explore how traffic congestion within a bottleneck scenario can be optimized using different strategies, including Caching and Custom Network Coding. By running tests using this potential form of optimization through the bottleneck, the worst-case scenario for network traffic can be navigated. The dumbbell topology is used instead of others, such as FatTree and BCube because we are explicitly analyzing the effects of caching and custom network coding under bottleneck and highly congested scenarios. By investigating the effects of caching and custom network coding under such conditions, we can see how well these techniques perform under high-stress situations. This will allow us to see potential insights into how similar techniques would perform within other topologies.

A. Overview

Our architecture comprises three components addressing specific performance aspects within the dumbbell topology, which includes speed testing, caching architectures, and network coding strategies.

B. Justification for Using Dumbbell Topology

Dumbbell topologies, characterized by their single bottleneck between two sets of hosts, effectively represent scenarios with bandwidth constraints, making them ideal for studying network congestion and throughput dynamics. Studies such as Xu et al. [3] and Jain and Routhier [4] underscore the utility of this topology for network research, validating our choice for experimental studies.

C. Cross-Traffic Testing

We explore the impact of cross-traffic, which consists of background data flows that do not interact with the primary traffic but may influence network performance due to congestion. Various scenarios, including different traffic types and network conditions, are tested to evaluate performance comprehensively.

1) Testing Scenarios: Initial tests are conducted without cross-traffic to establish baseline performance metrics for each component. Subsequent tests introduce cross-traffic in various forms—single bursts, periodic bursts, and continuous large data transfers—to assess the resilience and adaptability of the network setups.

D. Details

1) Network Speed Test Metrics: We examine the efficacy of Iperf3 in Cubic and BBR modes and UDPST in measuring network performance, focusing on their response to cross-traffic. TCP-based tests (Cubic and BBR) provide robustness against packet loss, while UDPST offers faster data transfer at the risk of increased packet loss.

2) Network Coding and Custom Packet Configurations: Advanced network coding techniques like XOR coding and Random Packet Spraying (RPS) are evaluated for their ability to enhance data transmission reliability and efficiency. We also explore the use of multiple links and packet pipelining to mitigate congestion and improve throughput.

3) Packet Caching Architectures: The caching component uses LRFU caching strategies to optimize data retrieval. This is critical for reducing latency and managing data efficiently across the network. Various caching architectures—singular, split, and multi-cache—are tested to determine the optimal configuration that balances cache capacity and access time.

E. Experiment Automation and Traffic Simulation

Our experiments are automated using Bash and Python scripts to generate synthetic traffic patterns, simulating real-world traffic scenarios. This automation ensures reproducibility and reliability in our experimental outcomes.

This model allows us to thoroughly assess the performance implications of different network configurations and strategies, providing a comprehensive analysis of potential improvements in data center network operations.

III. Speed Testing Component

A. *Speed Testing Introduction*

This component evaluates IPerf3 in both Cubic and BBR modes and UDPST to compare their performance under various network conditions. IPerf3 is a Transmission Control Protocol (TCP)-based tool that is standard for network speed tests, where Cubic is the default congestion control mode, and BBR is used to optimize bandwidth and minimize latency. UDPST, utilizing UDP, offers faster data transmission with a risk of increased packet loss. We assess the effects of different bandwidths, latencies, and packet drop rates, along with cross-traffic impacts on these technologies. The purpose of using IPerf3 in cubic and BBR mode is because IPerf3 is a widely accepted standard for speed testing. The newer BBR mode allows for faster transmission of data and is more reliable than the Cubic mode [33]. The decision to use both BBR and Cubic will enable us to compare the effects of switching between the two IPerf3 modes. The introduction of UDPST into speed testing is done to test how a UDP-based architecture could compare under different scenarios of speed testing done with varying types of cross-traffic. IPerf3 uses TCP to transmit, while UDPST uses UDP to communicate. The reason for doing this study is to find out if it is possible to replace TCP with UDP for speed testing due to UDP's several advantages, such as low overhead and speed compared to TCP. Furthermore, the current HTTP 3.0 is based on QUIC, which is based on UDP.

B. *Speed Testing Details*

Speed tests between hosts 1 and 3 in our dumbbell topology involve IPerf3 and UDPST operations, with an IPerf3 server on host three and clients on host 1. Cross-traffic from host 2 to 4 introduces potential bottlenecks, as depicted in Fig. 2.

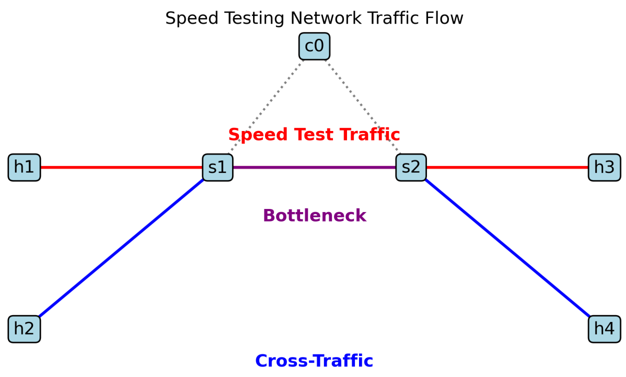


Figure 2: Speed Testing Network Traffic Flow

1,008 tests were conducted, each varying in bandwidth, latency, and packet drop, to establish a performance baseline and examine the influence of cross-traffic. Test conditions included no cross-traffic, single and multiple packet bursts, and file transfer scenarios, with each test lasting 30 seconds.

The cross-traffic scenarios tested were:

- **Single-Packet Periodic:** A 256-byte packet sent periodically.
- **Multi-Packet Periodic:** Multiple 256-byte packets sent periodically.
- **Burst-Packet:** Randomly sized packets sent periodically.
- **Single-File Periodic:** A 256-kb file sent periodically.
- **Multi-File Periodic:** Several 256-kb files sent periodically.
- **Burst-File:** A single 500 MB file sent mid-test.

Data from each test was captured and analyzed to determine the instantaneous bitrate, enhancing our understanding of each speed test framework's performance under simulated network stress.

C. *Speed Testing Results*

Results across various test scenarios demonstrated that BBR mode in IPerf3 consistently outperforms others, particularly under high latency and packet drop conditions. UDPST, while offering higher throughput, showed reduced stability with increased packet loss.

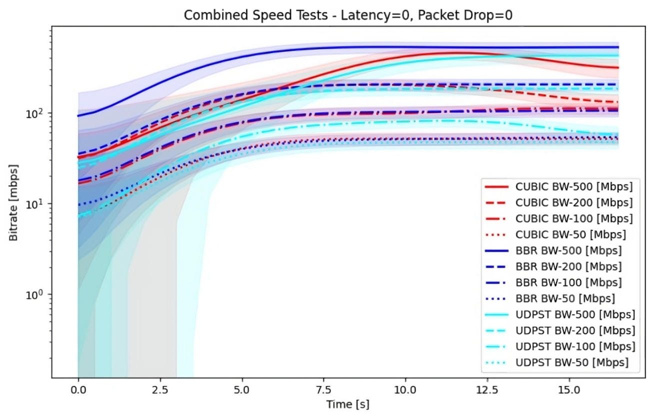


Figure 3: Speed Tests without Latency and Packet Drop

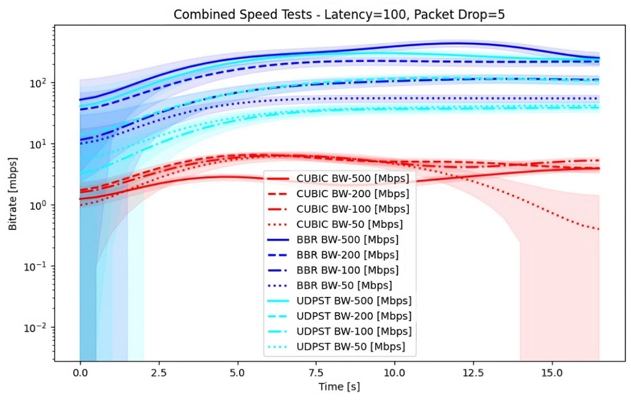


Figure 4: Speed Tests with Latency and Packet Drop

Further testing with cross-traffic scenarios revealed significant findings:

Overall, IPerf3's BBR mode proved to be the most stable across all tested scenarios, highlighting its suitability for high-demand network environments. UDPST showed resilience in less congested scenarios but struggled with stability during intensive cross-traffic tests. This evaluation confirms the robustness of TCP BBR for ensuring reliable network performance

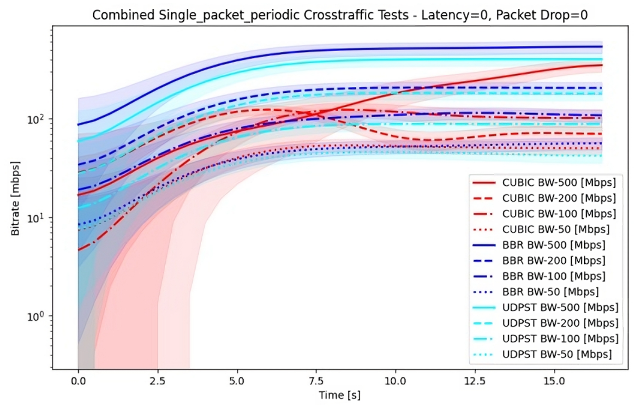


Figure 5: Single-Packet Periodic Cross-Traffic Tests

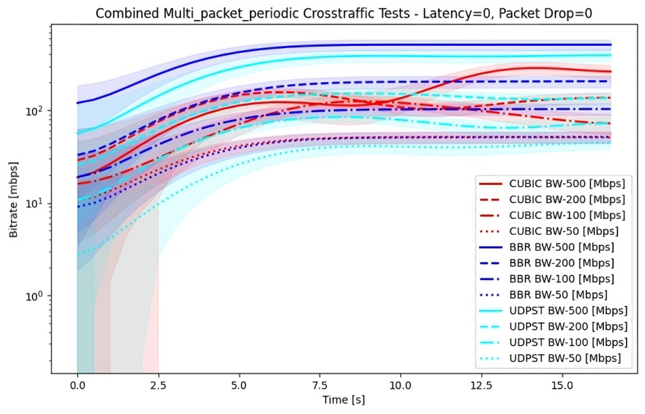


Figure 7: Multi-Packet Periodic Cross-Traffic Tests

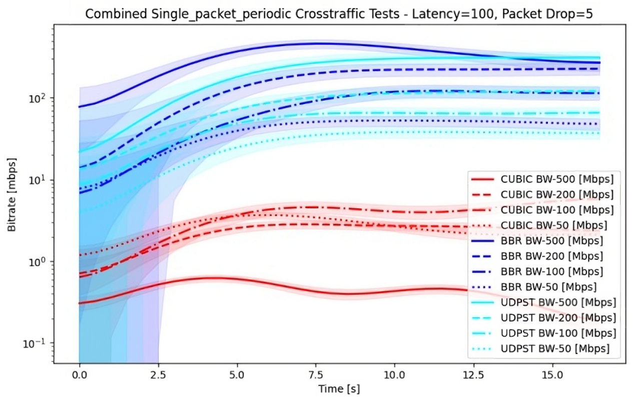


Figure 6: Single-Packet Periodic Cross-Traffic with Latency and Packet Drop

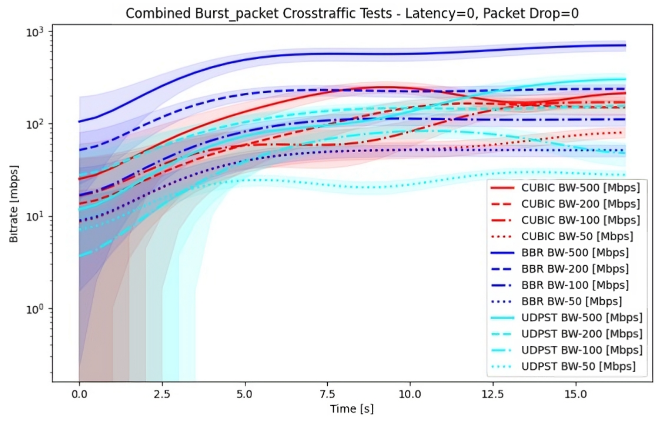


Figure 8: Burst-Packet Periodic Cross-Traffic Tests

and underscores the tradeoffs between TCP and UDP protocols in terms of throughput and reliability.

IV. Caching Component

A. Caching Introduction

This component evaluates various caching strategies within a dumbbell network topology to optimize data retrieval, reduce latency, and improve overall performance.

B. Caching Architectures: Considerations and Tradeoffs

Caching significantly speeds up data retrieval and reduces latency but introduces challenges such as increased cost, complexity, and resource demands. Advanced caching systems like multi-cache architectures incur higher costs and management complexity but offer better scalability compared to simpler single-cache setups. However, scalability comes with its challenges, such as the risk of under or over-provisioning, which can negatively impact performance. Additionally, caching can lead to data staleness, particularly in dynamic environments, potentially degrading the data's relevance and accuracy.

C. Caching Strategies

The component was tested on a primary path (h1 $\rightarrow$ s1 $\rightarrow$ s2 $\rightarrow$ h3) in a dumbbell topology. Traffic consisted of queries from h1 to h3, with responses including data from an SQLite database. This setup allowed us to analyze the effectiveness of caching strategies in handling realistic network traffic.

a) LRFU Caching Scheme: We implemented an LRFU caching scheme, which integrates the principles of LRU and LFU. This scheme adjusts to changing access patterns by dynamically balancing between recency and frequency of access:

1) **Cache Loading:** New entries are added until the cache fills, each initialized with a count of 1.
2) **Cache Hit:** Hits increase the count of the entry by a predefined factor D, reinforcing the entry's priority.
3) **Cache Miss and Eviction:** Misses decrement the LRU entry's count. Entries with a negative count upon cache saturation are replaced.

Factors like initial count, increment count (D), capacity, and operational delay are critical to balancing performance and resource utilization. The increment count (D) was used to prioritize frequently accessed entries by dynamically increasing their ”stickiness,” thus reducing the likelihood of their eviction from the cache. The experiment aimed to test the effectiveness of different values of D in optimizing cache performance under the Mininet dumbbell topology. The specific goal was to find a balance between cache efficiency and the system's ability to manage frequent cache hits without overcrowding the cache memory. By adjusting D, we evaluated how increasing

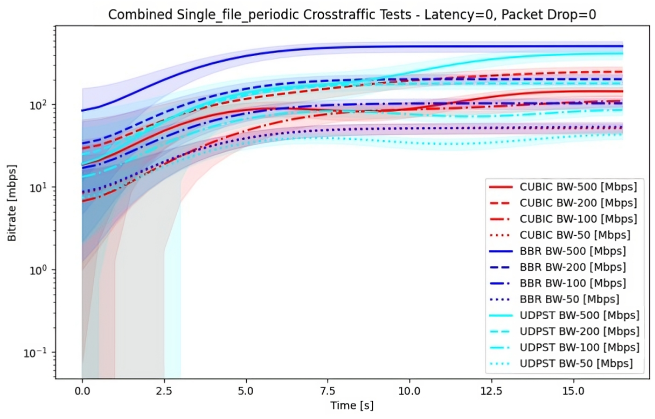


Figure 9: Single-File Periodic Cross-Traffic Tests

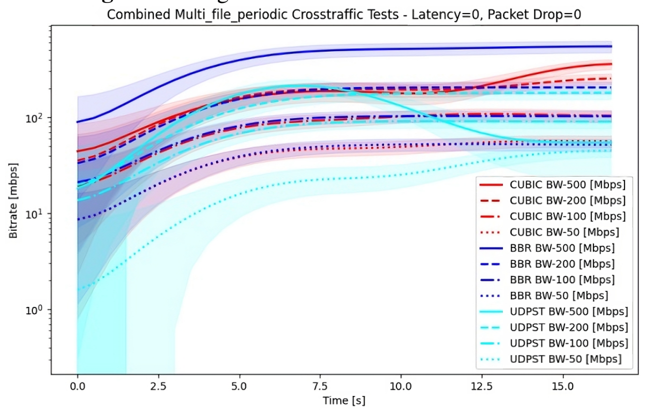


Figure 10: Multi-File Periodic Cross-Traffic Tests

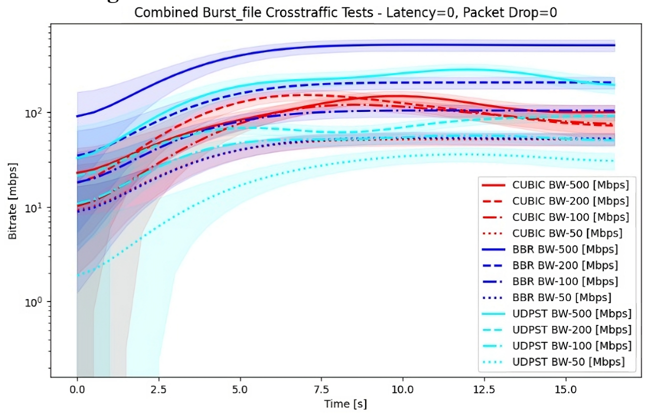


Figure 11: Burst-File Periodic Cross-Traffic Tests

the ”stickiness” of cached entries impacts overall latency and retrieval time.

b) Cache Optimization: The capacities of the M1, M2, and M3 caches were selected, and their specifications were optimized through simulations. The optimization tables reveal the relative performance of each caching strategy.

These results shown in Fig. 12 provide insights into the performance and efficiency of the different caching strategies. By carefully selecting the initial count, increment count, capacity, and delay, optimal cache configurations can be achieved that best suit the requirements of the network.

Table I: M1 Optimization

Cache Size	Initial	D	Hits
5000	1	2	749
	1	3	776
	1	4	935
	1	5	1088
	1	6	1289
	1	7	1201
	1	8	1256
	1	9	1243
	2	2	815
	2	3	811
	2	4	1202
	2	5	1112
	2	6	1253
	2	7	1224
	2	8	1129
	3	2	779
	3	3	948
	3	4	1141
	3	5	1213
	3	6	1129
	3	7	1242
	3	8	1126
	4	2	849
	4	3	957
	4	4	1207
	4	5	1149
	4	6	1245
	4	7	1241
	4	8	1002
	5	2	815
	5	3	1020
	5	4	1146
	5	5	1104
	5	6	1327
	5	7	1310
	5	8	1290
	10	2	847
	10	3	1044
	10	4	1106
	10	5	1141
	10	6	1222
	10	7	1023
10000	1	6	2669
	2	6	2633
	3	7	2622
	4	6	2625
	5	6	2707

Table II: M2 Optimization

Cache Size	Initial	D	Hits
5000	1	2	2425
	1	3	2532
	1	4	2391
	2	2	2365
	2	3	2470
	2	4	2462
	3	2	2393
	3	3	2516
	3	4	2467
	4	2	2428
	4	3	2481
	4	4	2511
10000	1	2	4894
	1	3	5171
	3	3	5164

Table III: M3 Optimization

Cache Size	Initial	D	Hits
5000	1	2	3971
	1	3	3962
	1	4	3996
	1	5	4003
	1	6	3942
	2	2	3965
	2	3	3940
	2	4	3934
	2	5	3981
	2	6	3864
	3	2	3987
	3	3	3896
	3	4	3952
	3	5	3936
	3	6	3892
	4	2	3872
	4	3	3922
	4	4	3937
	4	5	3911
	4	6	3950
[illegible]	1	2	8042
	1	3	7985
	1	5	8022

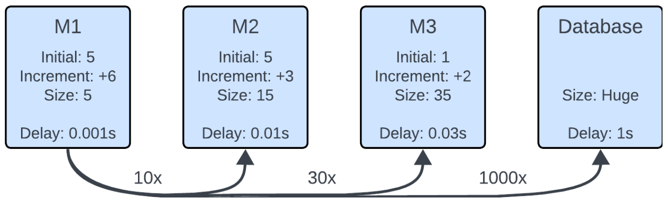


Figure 12: Custom Cache Specifications

D. Caching Architectures

Our evaluation of caching architectures within the dumbbell topology reveals significant differences in performance across various setups. Figure 13a illustrates the baseline scenario without any caching mechanisms, which predictably results in the slowest data retrieval times due to the absence of any cache at the h3 database.

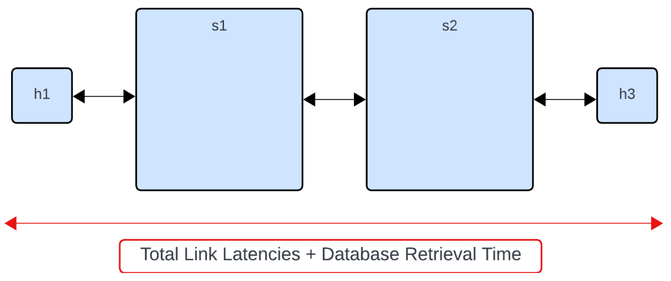
The initial caching architecture, a single M2 LRFU cache located at s1, as shown in Figure 13b, strikes a balance between capacity and access time, suitable for environments with limited caching capabilities.

We then explored a split cache configuration, deploying two M2 caches to address hardware capacity limitations. These caches, depicted in Figure 13c, operate in tandem yet are physically separated, sharing cache space and the exact LRU mechanism, effectively behaving as a unified cache system.

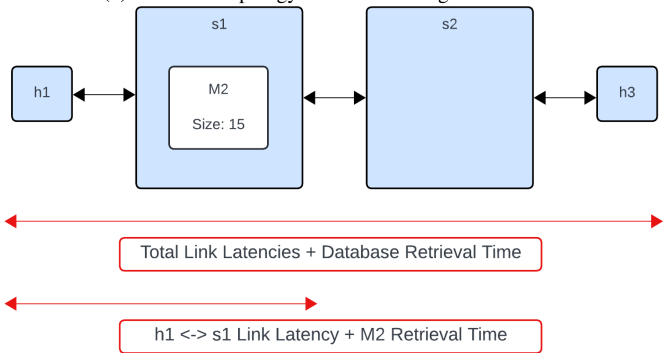
The most complex setup involved a multi-cache infrastructure, shown in Figure 13d, where each cache operates independently with its own set of rules and policies tailored to the specific needs of the network segments they serve. This configuration provided the best performance due to its high degree of optimization.

E. Architecture Comparison Results

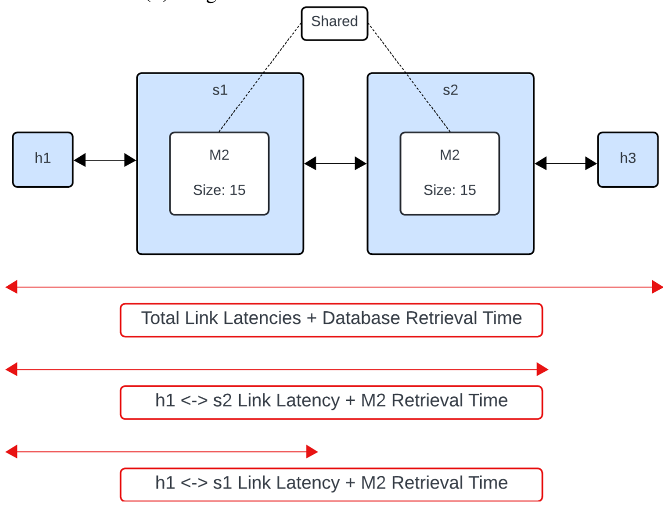
Simulation results from running 5000 packet tests on each caching infrastructure are displayed in Figure 14, detailing the accumulated delay throughout the simulation. As hypothesized,



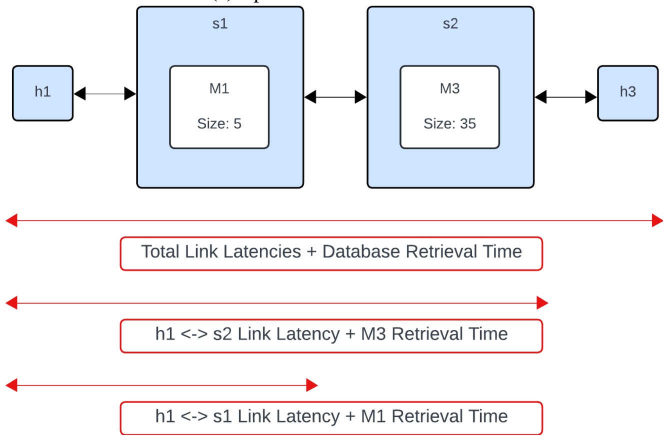
(a) Network Topology without Caching Mechanisms


s1
s2
h1
M2
Size: 15
h3
Total Link Latencies + Database Retrieval Time
h1 <-> s1 Link Latency + M2 Retrieval Time

(b) Singular M2 LRFU Cache Architecture


Shared
s1
s2
h1
M2
Size: 15
M2
Size: 15
h3
Total Link Latencies + Database Retrieval Time
h1 <-> s2 Link Latency + M2 Retrieval Time
h1 <-> s1 Link Latency + M2 Retrieval Time

(c) Split Cache Architecture


s1
s2
h1
M1
Size: 5
M3
Size: 35
h3
Total Link Latencies + Database Retrieval Time
h1 <-> s2 Link Latency + M3 Retrieval Time
h1 <-> s1 Link Latency + M1 Retrieval Time

(d) Multi-Cache Architecture

Figure 13: Different Caching Architectures

the multi-cache infrastructure outperformed all others, with a cacheless setup showing the least efficiency.

Cache Type	Cache Hits
No Cache	0
Single Cache	2532
Split Cache	3581
Multi-Cache	4187

Table IV: Cache Architecture Hits Comparison

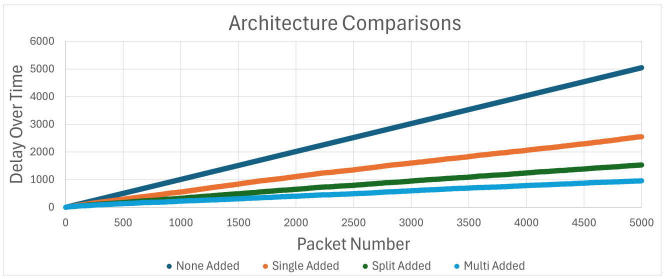


Figure 14: Simulation Times for Each Caching Infrastructure

F. Performance Comparison

The comparison between single-cache and multi-cache setups shows a significant improvement of approximately 65.4%, highlighting the benefits of employing a more sophisticated caching strategy in high-demand network environments.

G. Final Remarks

In this research, three different cache sizes are set between M1, M2, and M3 caches to portray the tradeoffs of increasing capacity for faster simulation time. Although not explicitly mentioned, another crucial aspect of the caching architectures implemented in the studies is to demonstrate a balance in the costs of integrating different caches. The singular cache architecture used an M2 cache, which balanced both the speed of M1 and the capacity of M3. The split cache architecture utilized two M2 caches, and the results showed that the increased capacity offered by a split cache offered better performance but would double the cost. And finally, the most optimized performance was expressed through a multi-cache architecture that focused on speed in s1 and capacity in s2. This architecture would theoretically cost the most because of the expensive technology needed for M1's speed and the drastically bigger capacity of M3. As for LRFU factors, two different variables were tested in each cache (M1, M2, M3). These variables were the initial value, which is the count that the cached entry would start with, and D, the value at which the entry is incremented when there is a cache hit. The values of D used in the experiment were chosen based on a series of performance simulations. These simulations involved varying D and analyzing its effect on the number of cache hits, the efficiency of retrieval, and the overall system latency. After changing these factors under 5000 and 10000 packets, the caches were set with the most optimized initial count and D value for each.

V. Network Coding Component

A. Introduction

This component employs XOR coding and RPS within a dumbbell topology to enhance data transmission reliability and efficiency, especially over the bottleneck link shown in Figure 15. By introducing multiple parallel links and network coding strategies, we aim to improve data resilience against packet losses significantly.

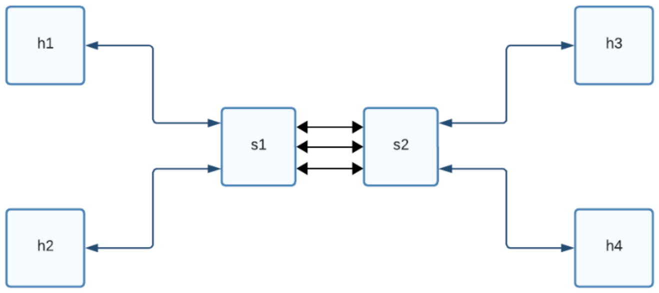


Figure 15: Network Topology with Multiple Links

B. Coding Strategies

1) XOR Coding: XOR coding at switch s1 involves splitting incoming packet payloads into two fragments and creating a third fragment by XORing the two pieces for redundancy. This setup enables data recovery at switch s2, where packet fragments are reassembled or reconstructed using XOR operations via custom headers, as shown in Figure 16 [29].

2 Bytes	2 Bytes	2 Bytes	4 Bytes
Fragment No.	No. Fragments	Orig. IPv4 Proto.	Tag

Figure 16: Custom IPv4 header

2) Random Packet Spraying (RPS): RPS aims to enhance packet delivery success rates by introducing redundancy. Once identified, large packets are either duplicated or forwarded generally based on a probabilistic decision, thereby increasing their chances of successful transmission without unduly burdening the network.

C. Validation

Testing with a 7% packet loss scenario revealed distinct transmission behaviors. The XOR coding strategy resulted in more packets being transmitted due to its redundancy mechanism, whereas RPS demonstrated lower retransmission rates, emphasizing its efficiency (Figures 17, 18, and 19).

D. Conclusion

The component's effectiveness in reducing packet drops and retransmissions confirms the utility of network coding strategies in enhancing network reliability. Future work will focus on refining and testing these techniques on more robust platforms to overcome current processing power limitations and ensure the accuracy of the experimental results.

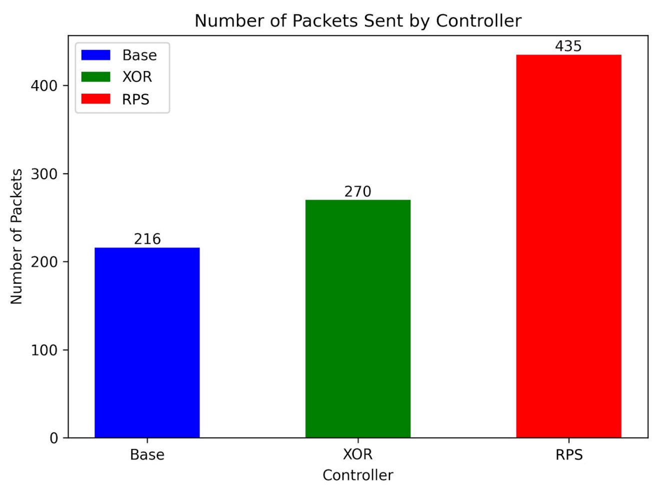


Figure 17: Transmission Patterns Across Different Controllers

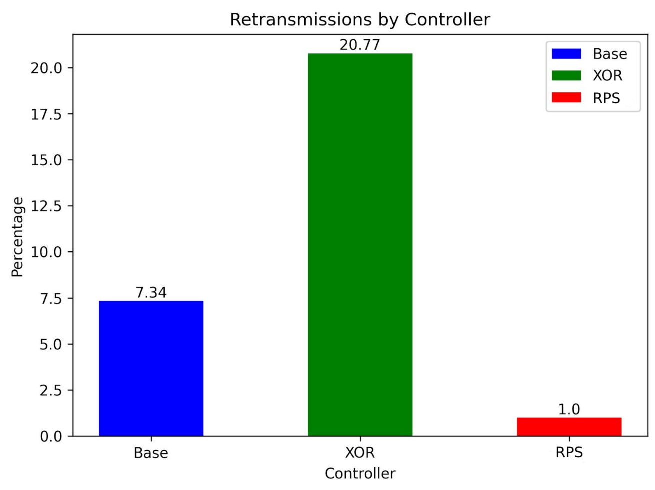


Figure 18: Retransmission Rate Comparison

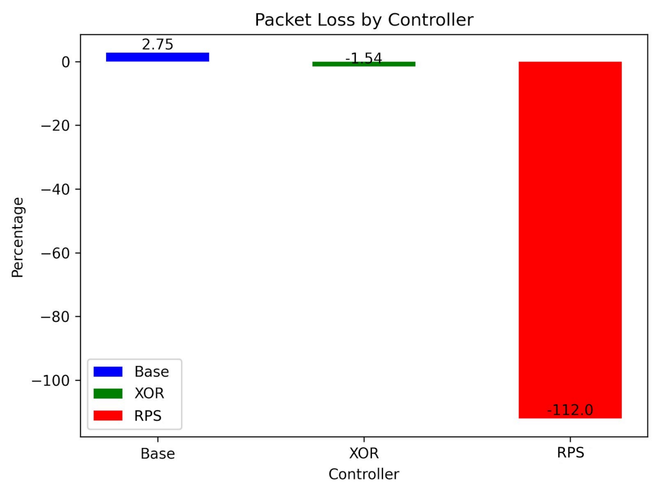


Figure 19: Packet Loss Comparison

VI. Discussion

This research has demonstrated the potential of integrating speed testing, caching architectures, and network coding strategies to mitigate key network performance parameters such as packet loss, latency, and retransmissions over a bottleneck. Each component addresses specific challenges within a data center environment, and their combined application can significantly enhance overall network efficiency.

The speed testing component, utilizing IPerf3 in both Cubic and BBR modes, along with UDPST, provides a robust framework for evaluating network performance under various conditions. The results indicate that BBR mode consistently outperforms other configurations, particularly in high-latency and packet-drop scenarios. This finding is crucial for data centers where maintaining high throughput and low latency is essential.

Network coding, precisely the XOR coding strategy, has shown promise in reducing packet loss for data center queries. However, its implementation is more expensive due to the use of three links. To address this, caching can be introduced to reduce duplicate queries from hosts, which reduces the load on the switches. The combination of these strategies can enhance data transmission reliability, especially in scenarios with significant packet loss.

Caching can play a vital role in optimizing data retrieval and reducing latency, particularly for redundant queries common in data centers. The study evaluated various caching architectures, including singular, split, and multi-cache configurations. The multi-cache architecture emerged as the most effective, balancing cache capacity and access time. This setup is particularly beneficial in environments with high query redundancy, as it reduces the load on the database and improves response times.

The integration of these components is particularly relevant for data centers, where many users generate redundant queries, leading to potential bottlenecks. By combining speed testing, caching, and network coding strategies, data centers can achieve significant reductions in time, cost, and power consumption. This holistic approach ensures that each component complements the others, providing a comprehensive solution to network performance challenges.

VII. Conclusion

This study utilized SDN within a Mininet environment to enhance data center network performance, focusing on speed testing, caching architectures, and network coding. While Mininet may not fully capture the complexities of large-scale networks, it provides a controlled environment to explore fundamental strategies like caching and network coding [31], [32].

Our evaluation highlighted BBR's effectiveness in handling high-latency and packet-loss scenarios, outperforming other protocols. Multi-cache architectures significantly optimized latency and cache hit ratios, and network coding techniques like Random Packet Spraying (RPS) minimized retransmissions and improved reliability.

These findings demonstrate the potential of customized SDN strategies in data centers. Future work will expand these strategies across more complex network topologies, explore additional queuing mechanisms like tc-pie and fq_codel, and replace artificial cross-traffic with traffic generated by our caching and network coding mechanisms.

We plan to further integrate network coding and caching strategies with various cross-traffic simulations to assess their performance over bottleneck scenarios. Testing with different queuing strategies and real-world network traces will also be conducted to ensure our solutions are robust and adaptable to real-world conditions.

References

[1] C. BasuMallick , "Switch vs. Router: Understanding 15 Key Comparisons," 2022.

[2] 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.

[3] Ahmad, M., Ahmad, U., Ngadi, M. A., Habib, M. A., Khalid, S., and Ashraf, R. "loss based congestion control module for health centers deployed by using advanced IoT based SDN communication networks", *International Journal of Parallel Programming*, Vol. 48, pp. 213–243, 2020

[4] Verma, L. P., and Kumar, M. "An IoT based congestion control algorithm", *Internet of Things*, Vol. 9, pp. 100157, 2020.

[5] Zhuang, R., Han, J., Xue, K., Li, J., Sun, Q., and Lu, J., "ProactMP: A Proactive Multipath Transport Protocol for Low-Latency Datacenters", *IEEE Transactions on Network and Service Management*, 2024

[6] Huang, J., Odiathevar, M., Valera, A., Sahni, J., Frean, M., and Seah, W. K., "Realtime BGP Anomaly Detection Using Graph Centrality Features", *International Conference on Advanced Information Networking and Applications*, pp. 222–233, 2024

[7] Alhilali, A. H., and Montazerolghaem, A., "Artificial intelligence based load balancing in SDN: A comprehensive survey", *Internet of Things*, 100814, 2023

[8] Iqbal, M. S., and Chen, C., "P4-MLFQ: A P4 implementation of Multi-level Feedback Queue Scheduling Using A Coarse-Grained Timer for Data Center Networks", *IEEE 12th International Conference on Cloud Networking (CloudNet)*, pp. 120–125, 2023

[9] Song, C. H., Khooi, X. Z., Joshi, R., Choi, I., Li, J., and Chan, M. C., "Network Load Balancing with In-network Reordering Support for RDMA", *Proceedings of the ACM SIGCOMM 2023 Conference*, pp. 816–831, 2023

[10] Zhang, X., Wan, K., Sun, H., Ji, M., and Caire, G., "A Novel Scheme for Cache-Aided Multiuser Private Information Retrieval with User-to-User Privacy", *57th Asilomar Conference on Signals, Systems, and Computers*, pp. 717–723, 2023

[11] Bhardwaj, S., and Girdhar, A., "Network Traffic Analysis in Software-Defined Networking Using RYU Controller", *Wireless Personal Communications*, 132(3), 1797-1818, 2023.

[12] Yan, D., Liu, Y., Zhang, S., Fang, B., Zhao, F., and Yang, Z., "PCNP: A RoCEv2 congestion control using precise CNP", *Computer Networks*, 110453, 2024

[13] Sedaghat, S., and Jahangir, A. H., "FRT-SDN: an effective firm real time routing for SDN by early removal of late packets", *Telecommunication Systems*, 80(3), 359-382, 2022

[14] Lu, Y., Ma, X., and Cui, C., "DCCS: A dual congestion control signals based TCP for datacenter networks", *Computer Networks*, 110457, 2024

[15] Li, W., Ren, M., Liu, Y., Li, C., Qian, H., and Zhang, Z., "Congestion Control Mechanism Based on Backpressure Feedback in Data Center Networks", *Future Internet*, 16(4), 131, 2024

[16] Zhang, X., Li, Q., Han, F., and Jiang, Y., "A receiver-driven transport protocol using differentiated algorithms for differential congestion in datacenters", *Computer Networks*, 245, 110357, 2024

[17] Zhuang, R., Han, J., Xue, K., Li, J., Sun, Q., and Lu, J., "ProactMP: A Proactive Multipath Transport Protocol for Low-Latency Datacenters", *IEEE Transactions on Network and Service Management*, 2024

[18] Hagargund, A. G., Shet, N. S. V., and Kulkarni, M., "DTPF algorithm based open-source Time-Sensitive Network leveraging SDN architecture", *IEEE Access*, 2023

[19] 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), 1031-1049, 2023

[20] Li, Z., Huang, J., Li, Y., and Wang, J., "Traffic-aware rate control for mix-flow in datacenter", *IET Communications*, 17(18), 2132-2139, 2023

[21] 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

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

[23] Nougnanke, B., Labit, Y., Bruyere, M., Aivodji, U., and Ferlin, S., "ML-based performance modeling in SDN-enabled data center networks", *IEEE Transactions on Network and Service Management*, 20(1), 815-829, 2022

[24] Shuja, J., Madani, S. A., Bilal, K., Hayat, K., Khan, S. U., and Sarwar, S., "Energy-efficient data centers", *Computing*, 94(12), 973-994, 2012

[25] Zhu, L., Karim, M. M., Sharif, K., Xu, C., Li, F., Du, X., and Guizani, M., "SDN controllers: A comprehensive analysis and performance evaluation study", *ACM Computing Surveys (CSUR)*, 53(6), 1-40, 2020

[26] Isong, B., Molose, R. R. S., Abu-Mahfouz, A. M., and Dladlu, N., "Comprehensive review of SDN controller placement strategies", *IEEE Access*, 8, 170070-170092, 2020

[27] Paliwal, M., Shrimankar, D., and Tembhurne, O., "Controllers in SDN: A review report", *IEEE access*, 6, 36256-36270, 2018

[28] Xiao, Z., Song, W., and Chen, Q., "Dynamic resource allocation using virtual machines for cloud computing environment", *IEEE transactions on parallel and distributed systems*, 24(6), 1107-1117, 2012

[29] Ahlswede, R., Cai, N., Li, S. Y., & Yeung, R. W., "Network information flow", *IEEE Transactions on information theory*, 46(4), 1204-1216, 2000.

[30] Ma, H., Luo, X., and Xu, D., "Intelligent queue management of open vSwitch in multi-tenant data center", *Future Generation Computer Systems*, 144, 50-62, 2023

[31] Handigol, N., Heller, B., Jeyakumar, V., Lantz, B., and McKeown, N., "Reproducible network experiments using container-based emulation", *In Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pp. 253-264, 2012.

[32] Varga, A., and Hornig, R., "An overview of the OMNeT++ simulation environment", *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, 2010

[33] Cardwell, N., Cheng, Y., Gunn, C. S., Yeganeh, S. H., and Jacobson, V., "BBR: Congestion-based congestion control", *Communications of the ACM* Vol. 60 No. 2, pp. 58-66, 2017

[34] Dong, M., Li, Q., Zarchy, D., Godfrey, P. B., and Schapira, M., "PCC: Re-architecting congestion control for consistent high performance", *In 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pp. 395-408, 2015.