EAI Endorsed Transactions
on Internet of Things
Research Article EAI.EU

Enhanced Placement-Aware Dynamic Load Balancing for Scalable and Resilient SDN Controller Placement Optimization

B.V. Prasanthi[1,2*], P. Chenna Reddy[3]

[1]Vishnu Institute of Technology
[2]JNTUA College of Engineering, Anantapuramu, Andhra Pradesh, India
[3]Department of CSE, JNTUA College of Engineering, Anantapuramu, Andhra Pradesh, India

Abstract

Software defined networking (SDN) introduces a flexible network control paradigm by separating the control plane from the data plane. Due to its improved programmability and scalability, it raises two important challenges: determining optimal controller placement and ensuring efficient load distribution among multiple controllers. These factors directly influence key performance metrics such as latency, throughput, and recovery time during failures. This work presents a detailed comparative evaluation of six controller placement strategies like random placement method, degree centrality, latency-aware, hierarchical, k-median, and genetic algorithm using two widely adopted SDN controllers. They are Open Network Operating System(ONOS) and OpenDaylight (ODL). Experiments are conducted on a 500-node fat-tree topology under two operating modes: without load balancing (No-LB) and with dynamic control-plane load balancing (With-LB). In addition, a failure-aware extension of the genetic algorithm is incorporated to evaluate time-to-recovery (TTR) under simulated controller failures. The results show that load balancing improves latency in most configurations and consistently enhances throughput. The best performance is achieved by ONOS combined with genetic algorithm or latency-aware placement under load balancing, reaching a minimum latency of 5.16 ms and an average RTT of 2.11 ms. OpenDaylight demonstrates stable and predictable behaviour across all placement strategies, although with slightly higher latency compared to ONOS. A notable observation is the degradation in performance under hierarchical placement when load balancing is enabled, where latency increases by approximately 11% for ONOS and 13% for OpenDaylight. Under failure conditions, the recovery time ranges between 242 ms and 277 ms depending on the controller and configuration. These findings provide practical insights into selecting suitable controller-placement and load-balancing combinations for both latency-sensitive and reliability-focused SDN deployments.

Keywords: software defined networking, controller placement problem, ONOS, OpenDaylight, dynamic load balancing, genetic algorithm, latency optimization, fat-tree topology, control-plane performance

Received on 16 June 2026, accepted on 7 September 2026, published on 22 September 2026

doi: 10.4108/eetiot.13546

1. Introduction

Software defined networking (SDN) is an architectural concept that that separates the control plane from the data plane [1,2]. it allows specialized controller software, which enables controlling large and complex networks by

*Corresponding author. Email: prasanthibeera@gmail.com

modifying routing policies and/or enforcing QoS rules or reconfiguring paths based on a programmable interface to real-time traffic conditions, without ever interacting with physical hardware. SDN has been used in the applications like mobile core, enterprise data centres and wide-area research testbeds. It is being expanded to scale networks to support cloud-native workloads and edge computing deployments. However, this architecture covers a challenge, as the number of switches under management increases, the problem of the location of the controllers to handle them becomes difficult. A controller located distant to its allocated switches adds propagation delay to all control-plane interactions. Latency-sensitive applications like real-time traffic engineering and reactive using OpenFlow routing will become expensive. Meanwhile, when switches are all controlled by one controller no matter how loaded it is, that controller becomes a bottleneck, reducing the performance of network. The load-balancing problem and controller placement problem (CPP) are interlinked in such way good placement minimizes the structural work which the load balancer has to carry out, and effective load balancing responds to limitations in any fixed placement [3].

Emerging applications such as personalized educational video platforms [4] rely heavily on low-latency and high-throughput network performance for seamless content delivery. This highlights the importance of efficient controller placement and load balancing in SDN to support real-time multimedia applications.

Majority of the published literature is either controller, or looks at placement without regard to the practical aspect of load balancing at runtime [5]. Those practitioners who may want to make an informed decision on deployment are thus left to generalize on studies with incompatible experimental conditions. We addressed gap in this paper. Our experimental framework consists of controllers ONOS or OpenDaylight(ODL) with 500-node fat-tree topology with various placement methods (six strategies), and load-balancing mode (No-LB vs. With-LB). We included the genetic algorithm to include fault-tolerance constraints to allow an evaluation of recovery performance when there is controller failure.

The contributions of this work are as follows:

1. A standardized experimental test of ONOS and OpenDaylight using six placement strategies under the same No-LB and With-LB on a fat-tree (500 nodes) topology.
2. Measurement of metrics like latency, round trip time (RTT), throughput, and bandwidth gains that can be attributed to dynamic load balancing such as the identification and explanation of the anomalous hierarchical placement degradation.
3. A failure-aware genetic algorithm test that offers time to recovery (TTR) performance benchmarks in the presence of controller faults requirements of the two platforms.
4. Practical guidance for SDN deployment on choosing the best controller-placement ensuring proper load balancing is shown.

2. Related work

Heller et al. [6] defined the CPP as a network optimization whose main goal is to minimize the average propagation delay between all switches and the nearest controller. The analysis [7] revealed that a relatively small topology-awareness in the placement choices can achieve large latency improvements relative to random baselines- an outcome. Further studies extended the CPP to incorporate concept known as survivability-aware placement was presented by Muller et al. [8]. This work demonstrates that resilience-optimal placements are very different than latency-optimal ones with the objective of maximize network resilience in the event of a controller failure. It inspired the failure-conscious genetic algorithm (GA) variant which is evaluated in section 4. The CPP described by Sallahi and St-Hilaire [9] is an integer linear program, whose solution is an exact small-network solution and a bound on the approximation error on large networks. Jimenez et al. used a multi-objective genetic algorithm [10]. Here, the location of controllers has increased significantly in recent years. latency, load, fault coverage combines optimisation algorithm and discovered that pareto-optimal placements perform better on all three dimensions in heterogeneous topologies compared to single-objective solutions.

Zhang et al. [11] explained the GA to failure-aware scenarios with the inclusion of expected TTR to the FA-GA variant fitness function. More recently, Liu et al. [12] showed that deep reinforcement learning could discover placements that are better than GA in more dynamic traffic situations, but with much greater computational during training time. Dixit et al. [13] suggested ElastiCon, a load balancing of multi-controller SDN Dynamic Framework. This framework reassigned the switches between controllers depending on real-time load measurements. Their experiments provided evidence that reactive load balancing decreases the mean latency and the maximum controller load in comparison to static assignment. Shu et al. [14] proposed a proactive variant which predicts the demand spikes based on traffic history, further reduction of 28% latency compared to reactive baselines. He et al. [15] compared per-flow and per-switch migration granularities, and found that per-switch migration provides a more effective good trade-off between control-plane stability and responsiveness. One of the relevant surveys by Hu et al. [16] combines multi-controller SDN architectures and finds load balancing to be the most influential runtime optimization that can be provided to distributed controller deployments.

The comparative analysis of ONOS and OpenDaylight has been carried out in several situations. Khondoker et al. [17] compared the two controllers to Floodlight and POX, and discovered ONOS to be faster in terms of latency and

OpenDaylight increasingly configurable to modular policy applications. Azodolmolky et al. [18] characterized OpenDaylight in high-throughput traffic patterns, and observed that its OSGi-based modular architecture brings in non-trivial internal communication overhead not present in ONOS. Dekeister et al. [19] studied the resilience of controllers when using Byzantine fault cases and noted that ONOS recovers more quickly under high-centrality placements. In the recent past, Yin et al. [20] introduced a graph neural network-based placement policy which learns topology specific latency models, which achieve better performance than latency-aware heuristics on heterogeneous graphs. Latency-aware algorithm offers a computationally-lighter benchmark with combination of machine learning methods [21]. The current literature is either on placement optimization [22,23] or on the load balancing, the combination systematic empirical work yet to be probed. Run-to-run evaluation varying controller platform, place, and load-balancing mode simultaneously on a steady-state and fault-condition assessment of standardised large-scale topology has not been done earlier. However, existing studies largely evaluate placement and load balancing independently, lacking a unified comparative analysis across controllers and operational modes. These issues are addressed in this paper.

3. System Architecture and Experimental Methodology

Figure 1 represents the complete evaluation of the network which consists of 500-node fat-tree topology, and then the controllers are selected and the six placement strategies are applied under No-LB and With-LB conditions using the latency, RTT, throughput and bandwidth metrics and failure-aware GA evaluation using TTR.

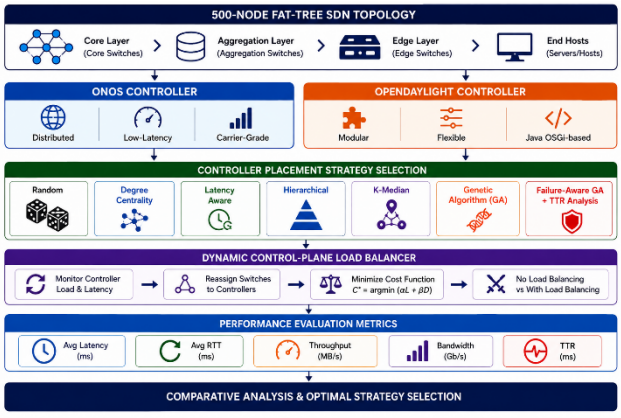

Figure 1. Proposed SDN load-balancing evaluation framework

A dynamic load balancer distributes switch assignments and performance is measured across five metrics. The failure-aware GA variant adds TTR analysis under simulated fault conditions. Each layer in the framework explained in the following subsections.

3.1. Network Topology

The experiments are conducted by using a hierarchical network of 500 nodes in the form of a fat-tree, which is a benchmark of data-centre networking research. The topology is made up of a high-capacity core layer backbone switches, an aggregation layer, which offers redundant interconnects, and an edge layer, where end hosts attached. The fat-tree topology is an ideal topology to use in this evaluation since it has enough topological complexity and to reveal significant variations between strategies of placement that have regular structure and are symmetric, which making them easier.

3.2. SDN Controllers

ONOS is a carrier-quality, distributed SDN controller which is used in high availability and scalability. Its clustered building keeps the replicated state of topology among several controller, which allows a quick failover and low-latency switch-to-controller communication. ONOS provides an API that isolates core control-plane services and application logic, which makes it very suitable when it comes to latency-sensitive deployments. ODL is an OSGi-based, modular SDN controller, which is running on the Java platform. Its plugin architecture enables operators to assemble functionality out of independently deployable bundles, which enables them to have a high level of operation flexibility. The design of ODL places emphasis on configurability and northbound APIs as opposed to raw latency performance, and is thus a popular solution in policy-driven network management applications. It provides a steady, programmable behavior is more important than milliseconds-level control-plane responsiveness.

3.3. Controller Placement Strategies

The various placement techniques which are used in the work are as follows.

3.3.1. Random Placement

Switches are selected uniformly in a random way from the nodes to host controller processes irrespective to the topology. This strategy serves as the statistical lower bound against which all deterministic approaches are compared and consistently produces the highest latency and longest recovery times.

3.3.2. Degree Centrality Placement

The degree centrality of a node V in graph $G(V,E)$ is defined as $d(V) = k(V)/(|V|-1)$, where $k(V)$ is the number of edges incident to V.

Controllers are placed at k nodes with the highest centrality values, on the premise that topological hubs serve the greatest number of switch neighbours and therefore minimize average hop count from any switch to its nearest controller. In fat-tree topologies, this strategy consistently selects aggregation-layer and core-layer switches.

3.3.3. Latency-Aware Placement

An optimization-driven approach that constructs a full pairwise propagation delay matrix using Dijkstra's algorithm, then applies a greedy forward-selection heuristic. At each step, the unselected node whose addition to the current placement set produces the greatest reduction in average network-wide switch-to-controller latency is chosen. This approach consistently achieves the lowest steady-state latency among all deterministic methods.

3.3.4. Hierarchical Placement

Controllers are arranged in a two-tier hierarchy. A root controller handles inter-domain routing, while regional sub-controllers manage local switch groups. This structure shows enterprise and carrier networks where administrative or geographic boundaries partition the control plane. As discussed in section 4, this arrangement shows unexpected incompatibility with the load balancer for both ONOS and ODL.

3.3.5. K-Median Placement

The switch set is divided in k clusters and a controller is located at the medoid which helps to minimises the sum of intra-cluster shortest-path distances. This is synonymous with the classical facility-location problem and controllers on natural traffic concentration points of the topology. K-median generally performs well in random and degree centrality but remains below latency-aware and genetic approaches.

3.3.6. Genetic Algorithm Placement and Failure-Aware Extension

The GA is a representation of controller placement sets in binary chromosomes and breds a population two-point crossover using tournament selection, and random mutation, on average latency minimization fitness function. Failure-Aware Genetic Algorithm (FA-GA) is an extension of the conventional fitness have a weighted time-to-recovery weight shown in (1)

$$f_{FA(c)} = \alpha * f_{lat(c)} + (1 - \alpha) * E\,[TTR\,(c, FM)] \quad (1)$$

where FM is the model of failure and α is a tunable weighting parameter.

This directs the evolutionary search to placements that continue to achieve a reasonable performance in case of the failure of one or more controllers.

In the next section we discuss the algorithms that supports the work.

3.4. Algorithms

The four core algorithms explain about controller placement along with load balancing is formally described below. In these, algorithms 1 and 2 detail the standard and failure-aware GA variants. Algorithm 3 describes the dynamic load-balancing mechanism. Algorithm 4 presents the Latency-Aware greedy heuristic. GA is used to find locations of controllers with low latency, FA-GA is used to calculate locations during failure recovery, EPA-DLB is used to dynamically balance the load of all controllers, and the latency-aware heuristic can be used as a computational benchmark for latency. Pseudocode of each algorithm is presented below.

Algorithm 1: Genetic Algorithm Placement (GA-CP)

```
Input:  G (V, E), k, pop_size, max_gen
Output: Optimal placement P* ⊆ V
-----------------------------------------------------------------------
population ← RandomPopulation(G, k, pop_size)
for gen = 1 to max_gen do
        for each chromosome c do
        f(c) ← SUM min_p d(v, p) / |V|
        end for
 parents   ← TournamentSelect(pop, f)
 offspring ← TwoPointCrossover(parents)
 offspring ← Mutate(offspring, p_m=0.02)
 pop ← EliteReplace(pop, offspring)
end for
return argmin_c f(c)
```

Algorithm 2: Failure-Aware GA + TTR (FA-GA)

```
Input:  G (V, E), k, failure model FM, α
Output: Resilient placement P*, TTR value
-----------------------------------------------------------------------
Define f_FA(c):
  f_lat ← SUM min_{p in c} d (v, p) / |V|
  f_ttr ← E [TTR (c, FM)] // simulated
  return α*f_lat + (1-α)*f_ttr
population ← RandomPopulation(G, k, pop_size)
for gen = 1 to max_gen do
        Evaluate f_FA(c) for each c
        SimulateFailure(c, FM) -> TTR_c
        Apply selection, crossover, mutation
end for
return argmin f_FA (pop), MeasureTTR(P*, FM)
```

Algorithm 3: Enhanced Placement-Aware Dynamic Load Balancing (EPA-DLB)

```
Input: Controllers C, Switches S, Placement Strategy P,
Threshold τ, Monitoring Interval T
Output: Updated Switch-to-Controller Assignment A: S
→ C
-----------------------------------------------------------------------
A ← InitialAssignment(C, S)
loop every monitoring interval T do
for each controller c_i ∈ C do
        load(c_i) ← | {s ∈ S : A(s) = c_i} |
end for
c_max ← argmax load(c_i)
```

c_min ← argmin load(c_i)
if load(c_max) − load(c_min) > τ then
for each candidate controller c_i do
Cost(c_i) ← $\alpha L_i + \beta D_i + \gamma P_i$
end for
if P = Hierarchical then
Select switch s* such that
Domain(s*) = Domain(c_min)
and d(s*, c_min) is minimum
else
s* ← NearestSwitch(c_min)
end if
c* ← argmin Cost(c_i)
A(s*) ← c*
end if
end loop
return A
where, L_i = Controller load (number of assigned switches), D_i = Propagation delay between switch and controller, P_i = Placement penalty factor, P_i = 1 for cross-domain migration in Hierarchical placement, P_i = 0 otherwise α, β, γ = weighting coefficients for load, delay, and placement penalty.

Algorithm 4: Latency-Aware Greedy Placement (LAP)
Input: G (V, E, W), k controllers
Output: Placement P* minimising avg delay

D ← AllPairsShortestPath(G) // Dijkstra
P* ← {}; best ← INFINITY
while |P*| < k do
v* ← argmin_{v not in P*}
SUM_u min(D[u][p], D[u][v])
P* ← P* union {v*}
end while
// Complexity: O (V^2 log V + k*V)
return P*

3.5. Dynamic Load Balancing Mechanism

The load balancer operates as a periodic background service monitoring of each controller's switch count. The controller assignment cost for controller c*i* is modelled as in (2).

$$c* = argmin\{ci\ in\ C\}(\alpha * Li + \beta * Di) \quad (2)$$

Where, L*i* is the current switch load on controller c*i*, D_i is the propagation delay from the candidate switch to c*i*, and α, β are configurable weighting coefficients.

When the imbalance between the most-loaded and least-loaded controller exceeds a predefined threshold, the switch geographically closest to the underloaded controller is migrated by redirecting its OpenFlow channel. This process repeats at each monitoring interval until balance is restored. The proposed enhanced placement-aware dynamic load balancing (EPA-DLB) mechanism is formulated based on the empirical findings of this study, particularly the hierarchical-placement anomaly. The current experimental results were generated using a conventional threshold-based load-balancing mechanism as discussed in the next section.

4. Experimental Results and Analysis

4.1. ONOS Controller Performance

Table 1. presents the ONOS measurements across all placement strategies and both load-balancing modes.

Table 1. ONOS Controller — Performance Results (500-Node Fat-Tree Topology)

Placement	Load Balancer	Latency (ms)	RTT (ms)	Throughput(MB/s)	Bandwidth (Gb/s)	Improvement(%)
Random	No LB	15.597	5.8952	0.2860	12.80	—
Random	With LB	7.8784	3.2145	0.3861	14.72	49.4
Degree	No LB	10.4431	3.9566	0.2860	12.80	—
Degree	With LB	6.7880	2.7696	0.3852	14.68	35.0
Latency Aware	No LB	8.0105	3.0350	0.2860	12.80	—
Latency Aware	With LB	5.2069	2.1245	0.3870	14.75	35.0
Hierarchical	No LB	8.2500	3.1200	0.2860	12.80	—
Hierarchical	With LB	9.1869	3.7485	0.3840	14.60	−11.4%
KMedian	No LB	10.0237	3.7977	0.2860	12.80	—
KMedian	With LB	6.8970	2.8141	0.3865	14.70	31.2
Genetic	No LB	8.6816	3.2892	0.2860	12.80	—
Genetic	With LB	5.1580	2.1050	0.3880	14.78	40.6

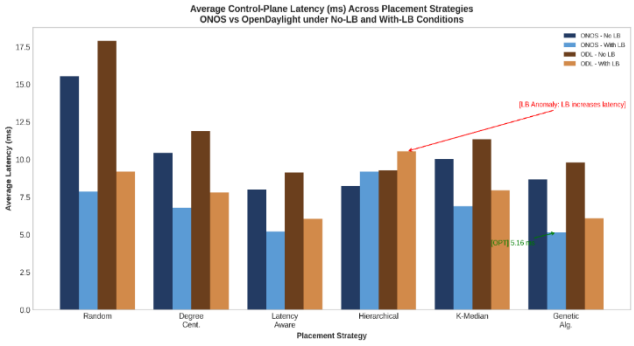

Figure 2. Average control-plane latency (ms) across six placement strategies

Figure 2 represents values for ONOS and ODL under No-LB and With-LB conditions. The optimal (OPT) represents the global minimum of 5.16 ms and lower bound (LB) represents the hierarchical degradation case.

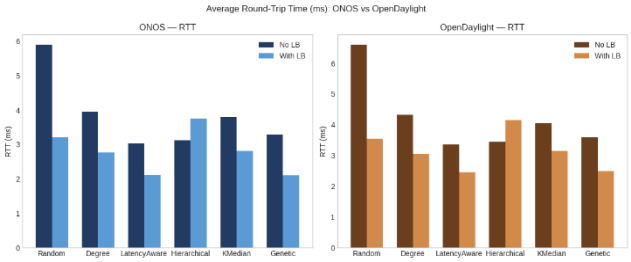

Figure 3. Average round-trip time (ms) comparison across all placement strategies for ONOS (left) and ODL (right)

Figure 3 shows that load balancing generally reduces RTT across most placements. However, an increase is observed under hierarchical placement due to inter-layer coordination overhead. By observing the ONOS results, two groups of observation stand out. Within the latency dimension, load balancing delivers a measurable dividend for five of the six placement strategies. It is most beneficial in absolute measures (15.56 ms, 7.88 ms, 49.4%) of the random strategy, which is a regular random placement. It generates very uneven distribution of switch-to-controller assignment and the load balancer which takes advantage of imbalance effectively. Genetic algorithm has the lowest latency of 5.16 ms globally, then followed by latency-aware placement and 2.11 ms RTT values. It indicates how they can place controllers near massive switch count. The load balancer will fine-tune the remaining imbalance. The throughput rates are at a high level (~ 0.384 – 0.388 MB/s) in all placement strategies and load balancing. This periodicity gives a solid indication that the benefit of load balancing throughput is completely caused by the removal of controller overload conditions, which are independent of the geometrical position of those controllers.

4.2. OpenDaylight Controller Performance

The measurements of the ODL are reported in Table 2. It is based on qualitative patterns much like ONOS however works at a slightly higher absolute latency values because of its OSGi-based internal messaging architecture.

Table 2. OpenDaylight Controller-- Performance Results (500-Node Fat-Tree Topology)

Placement	Load Balancer	Latency (ms)	RTT (ms)	Throughput (MB/s)	Bandwidth (Gb/s)	Improvement (%)
Random	No LB	17.9000	6.6000	0.2650	12.10	—
Random	With LB	9.2000	3.5500	0.3550	13.80	48.6
Degree	No LB	11.9000	4.3200	0.2650	12.10	—
Degree	With LB	7.8000	3.0500	0.3535	13.75	34.5
LatencyAware	No LB	9.1500	3.3500	0.2650	12.10	—
LatencyAware	With LB	6.0500	2.4500	0.3560	13.85	33.9
Hierarchical	No LB	9.3000	3.4500	0.2650	12.10	—
Hierarchical	With LB	10.5500	4.1500	0.3520	13.70	-13.4
KMedian	No LB	11.3500	4.0500	0.2650	12.10	—
KMedian	With LB	7.9500	3.1500	0.3545	13.78	29.9
Genetic	No LB	9.8000	3.6000	0.2650	12.10	—
Genetic	With LB	6.1000	2.5000	0.3570	13.88	37.8

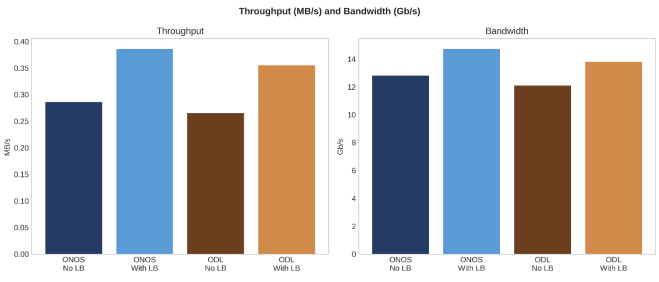

Figure 4. Throughput (MB/s) and bandwidth (Gb/s) summary across both controllers and LB modes. Load balancing uniformly improves both metrics regardless of placement strategy

Figure 4 shows the load balancing consistently improves both metrics across all placement strategies, with minor

variations observed due to controller behavior and placement characteristics.

Latency-aware placement achieves the lowest latency for ODL under load balancing of 6.05 ms, closely followed by Genetic placement of 6.10 ms. These values are slightly higher than ONOS, and is compatible with the internal processing overhead. ODL achieves 0.357 MB/s and 13.88 Gb/s on throughput and bandwidth respectively with load balancing. It is just a bit lower than ONOS 0.3861 MB/s and 14.72 Gb/s.

The Hierarchical placement anomaly is exactly the same with ODL. Load balancing drives latency from 9.30 ms up to 10.55 ms representing an approximate 13% latency increase under hierarchical placement. This is similar in two architecturally different controllers, indicating that the anomaly is structural related to the hierarchical domain-partitioning model rather than controller-specific.

4.3. Load Balancer Improvement Analysis

Figure 5 indicates the two controllers show similar qualitative profile. Load balancing is the most profitable to random placement in terms of percentage approximately by +49%.

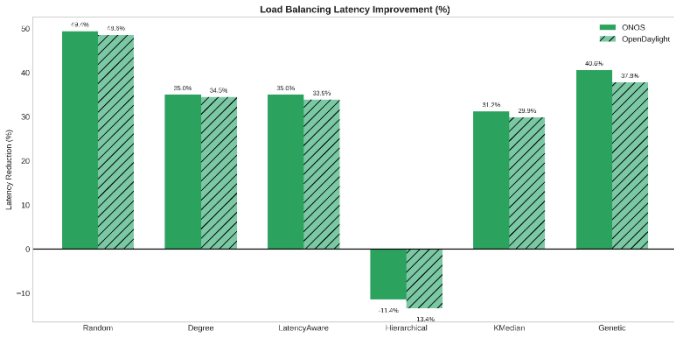

Figure 5. Percentage latency improvement due to load balancing for each controller-placement combination

While the advanced placements latency-aware and genetic are less advantageous since they are initialised in a lesser manner baseline. Approximately 11% latency is increased for ONOS and 13% latency increased for ODL in hierarchical placement which is the most practically significant finding.

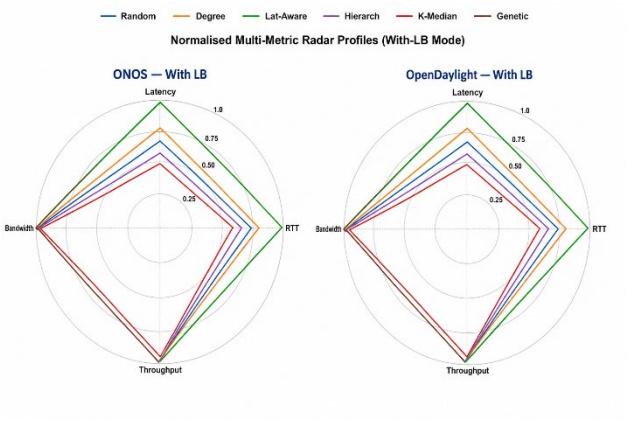

Figure 6. Normalised multi-metric radar charts for ONOS (left) and ODL(right) under With-LB mode. Metrics are normalised to [0,1] using best-value scaling; latency and RTT are inverted such that lower values correspond to higher scores. Higher values indicate better performance across all metrics

Figure 6 represents the pareto view of GA, which is the major frontier. The normalised latency and RTT axes of both controllers are immediately adjacent to the latency-aware placement. K-median and hierarchical placement is dragged to the middle by degree centrality whereas Hierarchical placement is dragged to the middle by its LB induced latency increase. Throughput and bandwidth axes observed consistent across strategies, which validates that these metrics are based on load-balancing, not on placement.

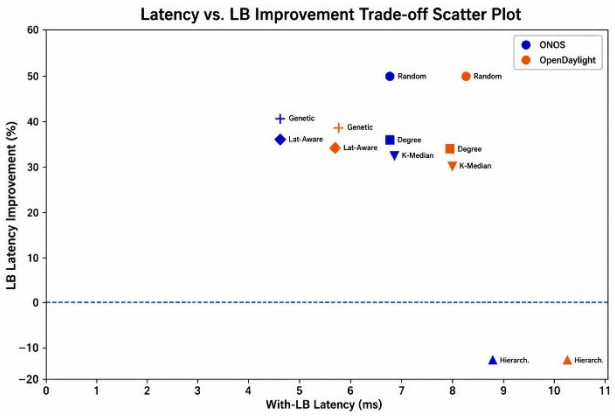

Figure 7. Scatter plot of With-LB latency vs. percentage improvement for all placement strategies

In Figure 7, each marker represents a different placement strategy. Ideal configurations appear toward the bottom-left region with low latency and high improvement.

4.4. Failure-Aware Evaluation and Time-to-Recovery

TTR is the time between the detection of controller failure and the recovery of control-plane resilience to failure in a simulated environment. Table 3 shows the evaluation results of the FA-GA. A failure of the controller is simulated by deleting the highest-loaded controller instance and measuring (a) the steady-state latency when in use recovery, and (b) the time between failure detection and complete switch coverage restoration.

Table 3. Failure-Aware GA — Performance Under Fault Conditions and TTR

Controller	Placement	LB	Node	Latency(ms)	RTT (ms)	Throughput (MB/s)	Bandwidth (Gb/s)	TTR (ms)
ONOS	Failure+TTR-GA	No LB	agg_4	11.47	3.49	0.2427	11.83	242
ONOS	Failure+TTR-GA	With LB	agg_0	7.46	2.45	0.3277	13.61	242
ODL	Failure+TTR-GA	No LB	agg_4	14.35	4.15	0.215	10.95	277
ODL	Failure+TTR-GA	With LB	agg_0	9.10	2.95	0.285	12.60	277

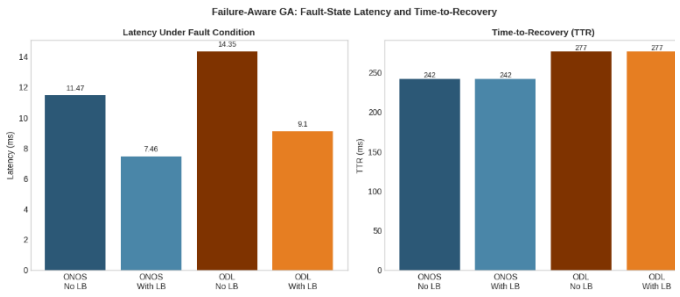

Figure 8. Failure-Aware GA evaluation. Left: average latency under fault conditions for No-LB vs With-LB. Right: Time-to-Recovery (TTR, ms) across all controller-LB combinations

In Figure 8, TTR remains consistent across load-balancing modes, indicating that recovery performance is primarily governed by controller architecture rather than load distribution. Even in faulty situations, load balancing still provides significant latency improvements. ONOS latency improves from 11.47 ms to 7.46 ms (approximately 35% improvement) and ODL from 14.35 ms to 9.10 ms (approximately 36.6% improvement). These improvements closely reflect the steady-state gains, which implies that the load balancer adjusts well to the decreased set of controllers after a failure. ONOS regains its stability at 242 ms and ODL requires 277 ms. This delay is in line with the distributed consensus protocol of ONOS, that permits the surviving controller instances to adopt remaining switches. The marginal TTR increment of ODL is an indication of the short-term increment latency due to parallel switch migration events instigated by the load balancer immediately after the failure.

4.5. Scalability Analysis

Figure 9 shows the relationship between the network size and latency with GA placement of both controllers. ONOS consistently shows lower absolute latency at all scales and the load-balancing advantage (shaded band) increases. When a network is large, the load balancing gets more important to implement.

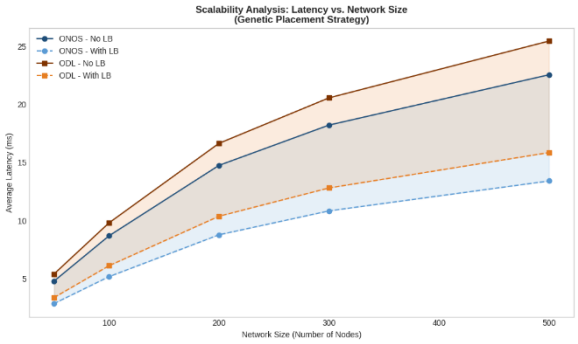

Figure 9. Scalability analysis: average latency (ms) as a function of network size (50–500 nodes) for ONOS and OpenDaylight under No-LB and With-LB modes, using genetic algorithm–based controller placement. Shaded bands represent the latency reduction achieved with load balancing at each network scale

ODL exhibits a stronger increase in latency with size of network, indicating the increased coordination overhead of per switch of its modular architecture.

Most placement strategies perform better in respect of latency, throughput and bandwidth for load balancing. Latency-Aware placement and Genetic placement are best for latency. In most cases hierarchical placement does have consistent latency rise with load balancing enabled, while ONOS tends to have better latency and faster recovery than ODL.

5. Discussion

5.1. Practical Deployment

The most direct practical application of these results is the right combination of controller, placement strategy, and load balancing setup varies according to the main performance aim of the target deployment. For latency-sensitive applications like real-time traffic engineering, latency-sensitive IoT applications, financial network and

infrastructure, the obvious choice is ONOS in which dynamic control-plane load balancer is recommended by using either latency-aware or genetic algorithm placement. This combination achieves 5.16 ms average latency and 2.11ms RTT of a 500-node network, the global optimum of all configurations experimented with in deployments that are flexible oriented (enterprise networks where milliseconds-level latency is secondary to modular policy management, vendor-neutral API, and service chaining) ODL using genetic algorithm placement and load balancing provides a high and balanced profile. It achieves 6.10 ms latency, 13.80 Gb/s bandwidth, and 277 ms TTR, all of which are operationally acceptable for non-latency-sensitive scenarios. For deployments requiring fault tolerance above all else carrier-grade networks, mission-critical infrastructure, ONOS with FA-GA placement is recommended. Its 242 ms recovery time, combined with 35% latency reduction even under fault conditions, provides the strongest resilience profile in our evaluation.

5.2. The Hierarchical-Load-Balancer Anomaly

The findings that both ONOS and ODL have the approximately 11% and 13% latency increase with hierarchical placement with load balancing is the most practically significant anomaly in our findings, and it deserves a close structural elaboration. The hierarchical placement model maps every switch to a particular regional sub-controller, according to topographical closeness inside a predetermined neighbourhood boundary. When the load balancer detects an imbalanced pair, a heavily loaded sub-controller and an underloaded sub-controller underloaded one, it migrates the closest eligible switch. That switch can be close to a domain boundary and hence it may be located on a sub-controller on a neighbouring domain, or on the root controller itself. In both instances, the migrated switches to the new controller which is located further and is geographically compared to its initial location. In the case of ONOS the consensus protocol should propagate this topology change ahead of the migrated switch can be offered steady service, and introduces a small delay latency penalty. ODLs OSGi messaging layer introduces a similar delay.

5.3. Limitations and Future Work

This study evaluates all configurations in a simulated environment. This simulation provides the reproducibility and experimental control is necessary for systematic comparison and physical testbed experiments which could reveal hardware-specific latency floors, OS scheduling jitter, and NIC-level overhead are not present in simulation. Validating the relative rankings observed here on a physical OpenFlow testbed is an important direction for future work. The load balancer evaluated here uses a simple reactive threshold policy. More sophisticated adaptive strategies including the proactive traffic-prediction approach and the deep reinforcement learning controller have been shown to outperform reactive baselines in dynamic environments. Comparing these approaches against the static threshold policy used here would provide a more complete characterization of the load-balancing design space. The study is conducted on a fat-tree topology. Whether the strategy rankings observed here generalise to other topology types like ring, mesh, scale-free, and small-world graphs. Prior work suggests that the optimal placement strategy is topology-dependent, implying that Genetic and Latency-Aware approaches may not always dominate on non-fat-tree graphs. Finally, the growing interest in machine-learning-based placement strategies[24,25] suggests a natural extension of this work, replacing the GA with a GNN-based or deep Q-learning-based placement policy, and evaluating whether the performance advantage over Latency-Aware placement that has been reported on heterogeneous topologies also manifests on the fat-tree structure used here.

5.4. Application Perspective: SDN for Educational Video Streaming

The latest educational solutions, like EduEvolve, which is based on video streaming and multilingual content delivery, represent a category of latency-sensitive apps that can be greatly improved by SDN optimization. EduEvolve uses YouTube-based educational content and does real-time recommendation based on user preferences, sentiment analysis, and engagement metrics. These systems are inseparably linked with effective network operations that can greatly reduce video streaming, buffering time, and user responsiveness. Here, SDN-enabled architectures can enable a potent base to improve the Quality of Experience (QoE) of such applications. The strategies of controller placement and dynamic load balancing methodologies that were considered in this paper directly affect the important performance metrics of latency, round-trip time (RTT), and throughput. As an example, the control-plane delays can be minimized by the optimal placement strategies such as genetic algorithm and latency-aware approaches which in turn reduce the video startup delay and buffering time during streaming sessions. Moreover, dynamic load balancing guarantees an efficient use of traffic between controllers and avoids overload and the slowing of video delivery performance even with a high number of users. Thus, incorporation of SDN optimization algorithms with intelligent educational systems such as EduEvolve can facilitate scalable, adaptive, and high-performance multimedia learning systems. The practical applicability of controller placement and load balancing strategies not just in network infrastructure, but also in new application areas like personalized e-learning and real-time video analytics.

6. Conclusion

This paper has presented a systematic, reproducible evaluation of SDN control-plane performance spanning

two controllers (ONOS and ODL) with six placement strategies, and two load-balancing modes on a 500-node fat-tree topology. Five core findings emerge from the data. First, load balancing is universally beneficial for throughput and bandwidth, and beneficial for latency in 10 of 12 controller-placement combinations. It should be considered a baseline component of any production SDN control-plane deployment. Second, ONOS with Latency-Aware or Genetic Algorithm placement and an active load balancer achieves the globally optimal latency (5.16 ms) and RTT (2.11 ms) across all tested configurations, making it the recommendation for latency-critical deployments. Third, OpenDaylight with Genetic Algorithm placement provides a strong, stable alternative for flexibility-focused deployments, achieving 6.10 ms latency and 13.80 Gb/s bandwidth with load balancing which is acceptable for the vast majority of enterprise and research network use cases. Fourth, hierarchical placement is the sole configuration where load balancing harms rather than helps. The exact replication of approximately 11% latency increase for ONOS and 13% latency increase for OpenDaylight identifies a structural incompatibility between hierarchical domain partitioning and cross-domain switch migration that affects any controller platform. ONOS's failure-aware GA placement achieves 242ms time-to-recovery under controller fault conditions 35ms faster than the equivalent ODL configuration confirming ONOS as the more resilient choice for fault-tolerant deployments. Load balancing continues to improve fault-state latency without impairing recovery speed. Together, these findings provide a data-driven decision framework for SDN engineers selecting controller placement and load balancing combinations. Future work will extend this evaluation to physical testbeds, adaptive load-balancing policies, and machine-learning-based placement strategies. The proposed framework can be extended to cloud data centers, edge computing and 5G networks where distributed controllers need to manage dynamically changing traffic and loads on the control plane. The approaches to placement and load balancing considered in the present study can help to achieve scalable and resilient controller deployment.

References

[1] Kreutz D, Ramos FM, Verissimo PE, Rothenberg CE, Azodolmolky S, Uhlig S. Software-defined networking: A comprehensive survey. Proc IEEE. 2015;103(1):14–76.

[2] Prasanthi BV, Reddy PC. Advanced Technologies in Electronics, Communications and Signal Processing. Cham: Springer; 2026. Chapter 3, Software Defined Networking (SDN) Technologies and Architectures for Efficient, Adaptive Networks; p. 31–44.

[3] Lange S, Gebert S, Zinner T, Tran-Gia P, Hock D, Jarschel M, Hoffmann M. Heuristic approaches to the controller placement problem in large scale SDN networks. IEEE Trans Netw Serv Manag. 2015;12(1):4–17.

[4] Bhavani A, Devi A, et al. Cognitive Computing and Cyber Physical Systems. Cham: Springer Nature Switzerland; 2024. Eduevolve: Enhancing Education Through Multilingual Information Extraction from Streaming Videos; p. 1–12.

[5] Neghabi AA, Navimipour NJ, Hosseinzadeh M, Rezaee A. Load balancing mechanisms in the software defined networks: A systematic and comprehensive review of the literature. IEEE Access. 2018;6:14159–14178.

[6] Heller B, Sherwood R, McKeown N. The controller placement problem. In: Proceedings of the 1st Workshop on Hot Topics in Software Defined Networks (HotSDN); 2012; Helsinki, Finland. New York: ACM; 2012. p. 7–12.

[7] Wang Y, Wang H, Liu A. Degree-centrality-based controller placement in SDN. IEEE Trans Netw Serv Manag. 2018;15(3):1175–1187.

[8] Müller LE, Oliveira P, Barcellos M, Gaspary L, Mota E. Survivor: An enhanced controller placement strategy for improving SDN survivability. In: Proceedings of IEEE Global Communications Conference (GLOBECOM); 2014; Austin, TX, USA. Piscataway: IEEE; 2014. p. 1–6.

[9] Sallahi A, St-Hilaire M. Optimal model for the controller placement problem in software defined networks. IEEE Commun Lett. 2015;19(1):30–33.

[10] Jimenez T, Cerdan F, Sanchez-Iborra R. Multi-objective evolutionary controller placement for SDN resiliency and performance. Comput Commun. 2020;151:282–294.

[11] Zhang Y, Cui L, Wang W, Zhang Y. A survey on software defined networking with multiple controllers. J Netw Comput Appl. 2018;103:101–118.

[12] Liu Q, Han H, Li J, Chen X. Deep reinforcement learning for adaptive SDN controller placement. IEEE Trans Netw Sci Eng. 2022;9(4):2619–2633.

[13] Dixit A, Hao F, Mukherjee S, Lakshman TV, Kompella R. ElastiCon: An elastic distributed SDN controller. In: Proceedings of the ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS); 2014; Los Angeles, USA. New York: ACM; 2014. p. 17–27.

[14] Shu Z, Wan J, Li D. Proactive load balancing for distributed SDN controllers using traffic prediction. IEEE Internet Things J. 2021;8(11):9025–9035.

[15] He J, Le F, Li J, Chen Z. Per-flow versus per-switch migration granularity in SDN control-plane load balancing. IEEE/ACM Trans Netw. 2022;30(2):746–759.

[16] Hu T, Guo Z, Yi P, Baker T, Lan J. Multi-controller-based software-defined networking: A survey. IEEE Access. 2018;6:15980–15996.

[17] Khondoker R, Zaalouk A, Marx R, Bayarou K. Feature-based comparison and selection of SDN controllers. In: Proceedings of the IEEE International Conference on Computer and Information Sciences (ICCAIS); 2014; Kuala Lumpur, Malaysia. Piscataway: IEEE; 2014. p. 1–7.

[18] Azodolmolky S, Nejabati R, Pazouki M, Wieder P, Yahyapour R, Simeonidou D. An analytical model for software defined networking: A network calculus-based approach. In: Proceedings of IEEE Global Communications Conference (GLOBECOM); 2013; Atlanta, GA, USA. Piscataway: IEEE; 2013. p. 1–6.

[19] Dekeister M, Valois F, Busson A. SDN controller resilience under Byzantine fault conditions. Lect Notes Comput Sci. 2022;13173:112–126.

[20] Yin H, Xie H, Wen T, Wang Z, Nguyen D. GNN-enhanced controller placement for heterogeneous SDN topologies. IEEE Trans Netw Serv Manag. 2023;20(1):345–358.

[21] Faezi S, Shirmarz A. A comprehensive survey on machine learning using in software defined networks (SDN). Hum Centric Intell Syst. 2023;3(3):312–343.

[22] Yao G, Bi J, Li Y, Guo L. On the capacitated controller placement problem in SDN. IEEE Commun Lett. 2014;18(8):1339–1342.
[23] Prasanthi BV, Reddy PC. Topology aware evaluation of SDN controller performance and placement strategies in star, linear, and ring networks. Cybern Inf Technol. 2026;26(2):233–251.
[24] Serag RH, Abdalzaher MS, Elsayed HAEA, Sobh M, Krichen M, Salim MM. Machine-learning-based traffic classification in software-defined networks. Electronics. 2024;13(6):1108.
[25] Xie J, Yu FR, Huang T, Xie R, Liu J, Wang C, Liu Y. Machine learning techniques applied to SDN: Research issues and challenges. IEEE Commun Surv Tutor. 2019;21(1):393–430.