Journal of Theoretical and Applied Information Technology

31st August 2023. Vol.101. No 16

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

COMPARATIVE STUDY AND PROPOSAL TO USE ENHANCED BELLMAN-FORD ALGORITHM FOR FASTER PATH COMPUTATION ON SDWAN CONTROLLER

MOHIT CHANDRA SAXENA[1] , MUNISH SABHARWAL[2], PREETI BAJAJ[3]

[1]Research Scholar, SCSE, Galgotias University, Greater Noida, India

[2]Dean, SCSE, Galgotias University, Greater Noida, India

[3]Vice Chancellor, Lovely Professional University, Punjab, India

E-mail: [1]mohit.chandra_phd20@galgotiasuniversity.edu.in, [2]dean_scse@ galgotiasuniversity.edu.in, [3]preetibajaj@ieee.org

ABSTRACT

Software-Defined Wide Area Network (SDWAN) technology has revolutionized network management by providing efficient and reliable communication across geographically dispersed locations. One critical aspect of SDWAN is path computation, which determines the optimal routes for data transmission between network nodes. Traditional shortest path algorithms like Bellman-Ford, Dijkstra, and SPFA are commonly used for path computation. However, the increasing scale and complexity of SDWAN networks demand more efficient algorithms. In this research paper, we propose an enhanced shortest path algorithm called SCBF (Shortest Path Computation Based on Bellman-Ford) specifically designed for SDWAN controllers. SCBF incorporates optimization techniques to reduce computational overhead and improve runtime efficiency. By building upon the principles of the Bellman-Ford algorithm, SCBF introduces novel optimizations that expedite path computation.

To evaluate SCBF's performance, we conduct a comparative study against traditional shortest path algorithms. Through extensive simulations using various network topologies and traffic scenarios, we demonstrate that SCBF outperforms the traditional algorithms in terms of runtime efficiency. SCBF achieves faster path computation on the SDWAN controller, reducing computational complexity and improving scalability. The comparative study showcases the advantages of SCBF in terms of reduced network latency, improved throughput, and enhanced scalability. The findings contribute to the development of more efficient path computation algorithms for SDWAN controllers, enabling faster decision-making in network routing and resource allocation. These improvements lead to enhanced performance and reliability in SDWAN deployments.

Future research and real-world implementations can explore the practical implications of SCBF in SDWAN environments, further validating its effectiveness. SCBF has the potential to provide significant benefits in terms of reduced network overhead and improved network management, ultimately enhancing the overall SDWAN experience.

Keywords: *SDWAN, Shortest path, Path Computation, Algorithm, SDN, link performance, Bellman Ford, Dijkstra, smart WAN*

1. INTRODUCTION

Software-Defined Wide Area Network [1] (SDWAN) has emerged as a transformative technology in the field of networking, offering efficient and cost-effective solutions for connecting geographically distributed networks. Traditional Wide Area Networks (WANs) [2] often face challenges in terms of network congestion, high latency, and limited bandwidth, which can hamper application performance and user experience. SDWAN leverages software-defined networking [3] principles to overcome these limitations and provide dynamic, secure, and optimized connectivity across diverse networks.

SDWAN technology offers several key advantages that make it highly applicable in various scenarios. It enables organizations to efficiently connect multiple branch offices, data centres, and cloud resources, providing seamless and reliable communication. SDWAN also offers centralized network

management, allowing administrators to monitor and control network traffic, security policies, and Quality of Service (QoS) [4] parameters from a single location. This centralized control is facilitated by an SDWAN controller, a crucial component responsible for managing and orchestrating the network.

Path computation is a fundamental operation performed by the SDWAN controller to determine the optimal routes for data transmission between network nodes. Efficient path computation is critical for ensuring low latency, high throughput, and effective resource utilization within the SDWAN environment. Several well-known algorithms, including Bellman Ford, Dijkstra, and the Shortest Path Faster Algorithm (SPFA) [5], are commonly employed for path computation in SDWAN controllers.

However, as SDWAN networks grow in scale and complexity, the runtime efficiency of path computation becomes a paramount concern. The ever-increasing data volumes and dynamic network conditions demand faster algorithms that can process path computations within tight time constraints. Therefore, there is a need to develop enhanced shortest path algorithms that can reduce computational overhead and improve the runtime efficiency of path computation on SDWAN controllers.

In this research paper, we present an enhanced shortest path algorithm called SCBF (Shortest Path Computation Based on Bellman-Ford) designed specifically for SDWAN controllers. SCBF incorporates novel optimization techniques to expedite path computation and overcome the limitations of traditional algorithms. Our study focuses on the runtime efficiency of path computation and aims to demonstrate that SCBF outperforms the existing algorithms, including Bellman Ford, Dijkstra [6], and SPFA [7], in terms of computational speed.

Through comprehensive comparative evaluations and simulations on various network topologies and traffic scenarios, we analyze the performance of SCBF. We compare its runtime efficiency with the traditional algorithms, emphasizing its superiority in terms of reduced computational complexity and improved scalability. The results of our study establish SCBF as a promising solution for faster path computation on SDWAN controllers.

The remainder of this paper is organized as follows: Section 2 provides a detailed overview of SDWAN architecture, highlighting the role of the SDWAN controller and the significance of path computation. Section 3 reviews the existing path computation algorithms in, literature review. Section 4 presents the SCBF algorithm, detailing its optimizations and how it improves runtime efficiency. Section 5 presents the comparative evaluation results, showcasing the superiority of SCBF over traditional algorithms. Finally, Section 6 concludes the paper with a summary of the findings and highlights potential future research directions in the field of SDWAN path computation.

By addressing the crucial need for faster path computation in SDWAN controllers [8], this research contributes to the development of more efficient networking solutions that can enhance the performance, scalability, and reliability of SDWAN deployments.

2. SDWAN ARCHITETURE

Software-Defined Wide Area Networking (SD-WAN) is a transformational approach used to simplify the management and operation of a WAN by decoupling the networking hardware from its control mechanism. Here is a detailed view of its architecture:

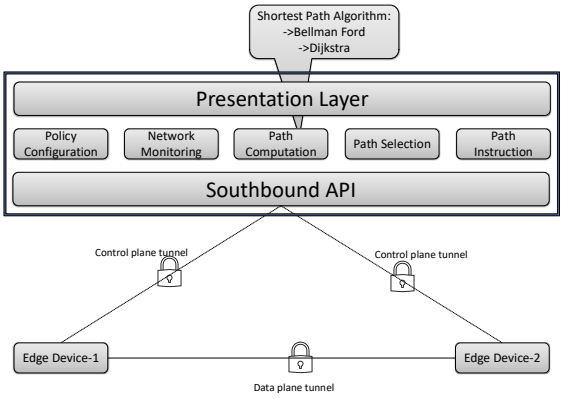

Figure 1: SDWAN Architecture

Edge Connectivity Layer: This layer is composed of the physical SD-WAN appliances placed at the network edge, typically at each branch office. These devices provide connectivity into the SD-WAN and might also offer additional functions such as routing, firewall, WAN optimization, and Wi-Fi.

Underlay Network: This represents the actual transport mechanisms used by the SD-WAN, which can include MPLS, broadband Internet, LTE/5G [9], or even satellite connections. The SD-WAN doesn't particularly care about the underlay it uses and can work with multiple types at the same time, allowing for hybrid WAN configurations.

Overlay Network: The SD-WAN creates a virtual overlay that abstracts the underlay network. This overlay is where the SD-WAN routes traffic and applies security and QoS policies. The overlay can

make intelligent decisions about which underlay network to send traffic over based on the current network conditions and the type of traffic.

SD-WAN Controller: This is the brain of the SD-WAN and is responsible for controlling the operation of the network. It communicates with each SD-WAN appliance to distribute policies and gather operational data. It often operates from a cloud-based location but can also be deployed on-premises in some architectures.

Orchestration and Management Layer: This layer provides a single point of control for the entire SD-WAN, simplifying the network's management. From here, administrators can make changes to network-wide policies, add or remove devices from the network, view network performance data, and troubleshoot any problems.

Security: Security is embedded within the SD-WAN architecture [10], providing secure connectivity over any transport and offering a range of features including encryption, segmentation, firewall and intrusion prevention, and integration with cloud security services.

Cloud Gateways: These are the points of presence for the SD-WAN within the cloud. They provide optimal and secure access to cloud resources, including SaaS applications and cloud data centers.

APIs: SD-WAN architectures often include APIs to enable integration with other systems for things like automation, reporting, and event correlation.

These components work together to provide an intelligent, secure, and highly flexible network that can be easily managed and quickly adapted to changing business needs.

In an SD-WAN architecture, the controller, also known as the Orchestrator, is responsible for making centralized decisions about network management and path computation based on the overarching policies defined by the network administrators.

Path computation within an SD-WAN architecture involves determining the optimal path for traffic flow between different network nodes. Here's a detailed explanation of the process:

Policy Configuration: Administrators define the policies that govern the SD-WAN's behavior, including rules about application priority, Quality of Service (QoS), security requirements, and cost-efficiency. These policies get communicated to the controller.

Network Monitoring: The SD-WAN edge devices continuously monitor the health and performance of the underlying network links, measuring factors like latency, packet loss, jitter, and available bandwidth. This telemetry data is reported back to the SD-WAN controller.

Path Computation: Using the reported telemetry data, the controller determines the best path for each type of traffic according to the defined policies. For example, critical application traffic might be sent over a high-quality MPLS link, while less critical or voluminous traffic might be sent over a broadband Internet link. This decision-making process leverages a technology known as Dynamic Path Selection (DPS) [11].

Path Selection Adjustment: The controller can dynamically adjust its path selection in real time in response to changes in network conditions. If a link becomes congested or fails, the controller can automatically reroute traffic to maintain performance and reliability.

Path Instructions: The controller communicates its path decisions back to the SD-WAN edge devices, which then forward traffic accordingly. This process happens seamlessly, maintaining application performance and user experience.

This intelligent, centralized, and automated method of path computation is one of the key features that differentiate SD-WAN from traditional WAN technologies. It allows for more efficient use of network resources, improved performance, and greater flexibility and adaptability in the face of changing network conditions.

3. PATH COMPUTATION ALGORITHMS AND RELATED PREVIOUS WORK

Software-Defined Wide Area Network (SD-WAN) controllers use various algorithms for path computation, often leveraging a combination of sophisticated techniques and policies. Here are a few of the key ones:

Dijkstra's Algorithm: One of the most popular and fundamental pathfinding algorithms, Dijkstra's Algorithm can be used to determine the shortest path between two nodes in a network graph. This can be particularly useful in an SD-WAN context where one might need to determine the most efficient route for data packets to traverse.

The Dijkstra method, conceptualized by the Dutch computational theorist Edsger Dijkstra in 1956 and later published in 1959 [12], leverages the graph model to address issues in singular-source shortest path and form the briefest path tree. The relevance of Dijkstra's method in determining the briefest course has important practical implications. The information transmitted across the Internet traverses diverse routes globally, thus it is crucial to ensure it navigates through succinct and less congested paths, a possibility achieved through the Dijkstra method. Additionally, the OSPF (Open Shortest Path First) [13] routing protocol employed in Internet Protocol

(IP) operates based on Dijkstra's method [14]. Utilizing this method, platforms like Google Maps can now specify the most direct route from a particular location to a specific restaurant or airport. It is an exemplification of a powerful greedy method. The greedy method is deemed the optimal universal solution to certain optimization challenges, though it may produce suboptimal solutions to other problems in some instances. One of the primary benefits of Dijkstra's method is that it omits visiting unnecessary nodes once the intended target node is reached [15]. Conversely, a disadvantage of Dijkstra's method is its substantial CPU memory consumption when executed on many nodes on a computer application [16]. Additionally, it is incapable of managing negative edges, its utilization is exclusively on positive weight graphs. In the Dijkstra method, the route is indeterminate. The nodes are split into two categories: temporary (t) and permanent(p). Initially, assign the distance of the source node to zero [distance(a) = 0], and allocate the distance for other nodes to the infinity value [distance(x) = ∞]. Step 2, seek the node x with the smallest value of d(x). If no temporary nodes exist or the value of d(x) equals infinity, the node x is labelled as permanent, meaning that d(x) and the parent of d(x) will no longer change. Step 3, apply the following comparison for each temporary node labelled vertex y adjacent to x: If d(x) + w (x, y) < d(y) then D(y) = d(x) + w (x, y) (1) Based on equation (1), if the distance of labelled node x plus link weight (x, y) is less than the labelled node y distance, then the distance of labelled node y will be updated.

Bellman-Ford Algorithm: This algorithm is capable of handling negative weights and is typically used in routing protocols like OSPF and BGP. It can be employed in an SD-WAN context to calculate the shortest path, even in more complex network scenarios.

The Bellman-Ford methodology, a cornerstone in network navigation, is attributed to its originators, Richard Bellman and Lester Ford [17]. It determines the shortest route from a singular origin to all other points in a weighted, directed network, allowing for negative edge values (Bellman, 1958; Ford, 1956). Despite its adaptability, the Bellman-Ford method experiences a significant time complexity of O(VE) [18], which could be problematic for extensive networks (Cherkassky, Goldberg, & Radzik, 1996) [19]. A myriad of scholars have aimed to enhance Bellman-Ford's efficiency, resulting in the creation of various enhancements and versions.

Constrained Shortest Path First (CSPF): This is an extension of the Dijkstra's algorithm, typically used when there are additional constraints to consider. It can compute the shortest path that also satisfies specific constraints, such as bandwidth requirements, latency, or cost.

Dynamic Path Selection (DPS): This is less of an algorithm and more of a technique commonly used in SD-WAN. DPS involves the controller dynamically selecting the optimal path for network traffic based on real-time link performance.

Multi-path Algorithms: These algorithms are designed to manage and optimize the use of multiple simultaneous network paths. This can be beneficial in an SD-WAN context for improving reliability and bandwidth utilization.

It's important to note that the specific algorithms used can vary significantly depending on the specific SD-WAN solution, as different vendors may implement different strategies for path computation and traffic management. Ultimately, the goal is to meet the needs of the business in terms of performance, reliability, security, and cost-effectiveness.

In 2009, S. Jung and his team [20] introduced Contraction Hierarchies, a technique for speeding up the calculation of the shortest path by pre-processing the input graph. This method has a worst-case query time complexity of O(n log n) and consumes linear space.

Their research presents a Hierarchical Multi (HiTi) graph structure, specifically constructed to organize large-scale geographical road maps and expedite the process of determining the least costly path. The HiTi graph structure offers a novel approach to abstract and organize geographical road maps in a layered fashion. We introduce a novel shortest path method, termed SPAH, that is built on the HiTi graph structure of a geographical road map for its computations. We provide evidence of SPAH's optimality. Performance evaluation of SPAH on grid graphs shows a significant reduction in the search space compared to existing techniques. We also present a comprehensive experimental comparison of the HiTi graph method with other similar works on grid graphs. Within the framework of the HiTi graph structure, we propose a parallel shortest path method named ISPAH. The conclusion of this research suggests that the inter-query shortest path search yields more scalable parallelism compared to the intra-query shortest path search.

In 2001, Ulrik Brandes [21] offered an algorithm to calculate betweenness centrality, a metric related to shortest path algorithms, in O(VE) time and O(V+E) space, which is quicker than earlier methods. Addressing the growing need to compute centrality measures on large yet incredibly sparse networks,

this paper introduces cutting-edge algorithms for betweenness. The storage needs for these algorithms are O(n + m), and their execution times are O(nm) and O(nm + n2 log n) [22] for unweighted and weighted networks respectively, where 'm' represents the number of links. We supply experimental data that significantly extends the range of networks where centrality computations are feasible.

Measuring betweenness centrality is essential for analyzing social networks, but it requires significant resources. Currently, the most efficient known methods require?(n 3) time and ? (n 2) space, with 'n' being the total number of network participants.

In 2009, Kleinberg, J. and his colleagues [23] examined compact routing schemes for networks exhibiting a low doubling dimension. They explored two versions: name-independent routing and labelled routing. The main findings for this model were as follows. Firstly, we presented the inaugural name-independent solution. Specifically, we achieved constant stretch and polylogarithmic storage. Then, we obtained the first truly scale-free solutions, where the network's aspect ratio did not impact the stretch. Scale-free methods were provided for three models: name-disjoint routing on graphs, labelled routing on metric spaces, and labelled routing on graphs. Finally, we established a lower bound necessitating linear storage for stretch > 3 schemes. This had the significant effect of separating the name-independent problem model from the labelled model for these networks, as compact stretch-1+e labelled schemes were known to be feasible.

This paper proposes algorithms for routing in networks with low doubling dimensions, a feature related to sparsity. These algorithms have sublinear time complexity concerning the network size, making them extremely efficient for large sparse networks.

Zheng Wand and his team [24] discussed routing algorithms that aim to identify the shortest path. Algorithms that adjust to traffic shifts often show unstable behaviors, resulting in performance degradation. The study here first addressed these issues from a decision-making and control theory perspective, followed by an in-depth analysis of the performance traits of these shortest-path routing algorithms.

N. Futamura and colleagues [25] suggested that evaluating IP address lookup algorithms often requires considering multiple criteria such as lookup time, update time, memory usage, and sometimes, the time needed to construct the supporting data structure for lookups and updates. Many current methods are primarily focused on optimizing a single factor, and as a result, they may not scale effectively with the ongoing growth of routing tables and the upcoming implementation of IPv6 with its 128-bit long IP addresses. In contrast, this research aimed to improve multiple factors at once and provide solutions that can readily scale up to IPv6.

In this framework, two IP address lookup strategies were introduced: the Elevator - Stairs method and the logW - Elevators method. For a routing table with N prefixes, the Elevator - Stairs method utilized optimal O(N) memory and provided superior lookup and update times compared to other methods with similar memory requirements. The logW - Elevators method, meanwhile, offered an O(log W) lookup time, where W is the length of an IP address, and improved both the update time and memory usage.

The performance of these algorithms was evaluated using the MAE-West router, which contained 29,487 prefixes. The results showed that the Elevator - Stairs method achieved an average throughput of 15.7 million lookups per second (Mlps) while using 459 KB of memory. The logW - Elevators method demonstrated an average throughput of 21.41 Mlps, although it required more memory (1259 KB).

Kamesh Madhuri and others [26] showcased an experimental investigation of the single-source shortest path problem with non-negative edge weights (NSSP) on large-scale graphs, utilizing the Δ-stepping parallel algorithm. Performance results on the Cray MTA-2, a parallel computer system characterized by multithreading, are revealed. The MTA-2, a high-end shared memory system, offers two distinctive features that enable efficient parallel implementation of irregular algorithms: the ability to use fine-grained parallelism and the presence of low-overhead synchronization primitives. Notable parallel speed-up is demonstrated in the implementation, compared to competitive sequential algorithms, especially for sparse graphs with a small diameter. For example, Δ-stepping applied to a directed scale-free graph containing 100 million vertices and 1 billion edges, completes in less than ten seconds on 40 CPUs of the MTA-2, achieving a relative speed-up close to 30. It's worth mentioning that, to the best of our knowledge, these are the first performance results of the shortest path problem on practical graph instances on the scale of billions of vertices and edges.

Seth Pettie, in his paper [27], introduced a new all-pairs shortest path algorithm, designed for use in real-weighted graphs using the standard comparison-addition model. The algorithm operates in improved time, surpassing the established time limit, which was previously achieved by applying Dijkstra's

algorithm using Fibonacci heaps. Here, m and n denote the number of edges and vertices, respectively.

The proposed algorithm is fundamentally based on the component hierarchy approach, a new shortest paths method introduced by Thorup for undirected graphs weighted by integers, and later extended by Hagerup to include directed graphs weighted by integers. The main contributions of this paper include a method for estimating shortest path distances, along with a method to use these estimated distances to calculate the exact ones. Additionally, the paper provides a concise, singular definition of the hierarchy-type shortest path algorithm class. This definition opens up some negative lower bounds regarding the computation of single-source shortest paths using a hierarchy-type algorithm.

In his 1959 paper, Dijkstra E.W. [28] presented his renowned algorithm for finding shortest paths in a graph. The time complexity of Dijkstra's algorithm, when using a binary heap, is O((E+V) log V).

Fredman, M.L. and others [29] presented Fibonacci heaps in 1987. This data structure can be used to improve the running time of Dijkstra's algorithm to O(E + V log V). The authors invented a new data structure to implement heaps or priority queues, termed Fibonacci heaps or F-heaps. This structure is an extension of binomial queues, a concept initially proposed by Vuillemin and subsequently explored by Brown. F-heaps efficiently support deletion from an n-item heap in O(log n) amortized time and perform all other standard heap operations in O(1) amortized time. The utility of F-heaps led to improvements in running times for several network optimization algorithms.

Dinitz and Itzhak [30] proposed a new hybrid algorithm, the Bellman-Ford–Dijkstra (BFD), by combining the Bellman-Ford and Dijkstra algorithms. This algorithm finds the shortest paths from a source node s in a graph G with general edge costs, improving the runtime of the Bellman-Ford algorithm when negative cost edges are sparsely distributed. The algorithm's principle is to run the Dijkstra algorithm multiple times without resetting the temporary value of d (v) to the vertices.

Lacorte and Chavez [31] studied the application of A* and Dijkstra algorithms in designing a smart school transport system route optimization model. Both algorithms were tested using a tool called EESCOOL. The results showed that the A* algorithm performed better and produced minimal expected time of arrival (ETA) during routine traffic on a small graph.

Abbas and others [32] introduced an algorithm called the Caption algorithm to solve the shortest path problem with reduced time complexity compared to the Dijkstra algorithm. This algorithm can serve as an alternative to Dijkstra's algorithm as it can repeat the search process by increasing the reduction coefficient.

Sapundzhi and Popstoilov [33] evaluated Dijkstra's algorithm, Floyd-Warshall algorithm, Bellman-Ford algorithm, and Dantzig's algorithm for solving the shortest path problem. They concluded that Dijkstra's algorithm is more efficient for a larger number of nodes.

Oyola and colleagues [34] introduced a method called Safe and Short Evacuation Routes (SSER), which uses a Dijkstra-based algorithm to solve the problem of determining the shortest safe paths in residential environments with multiple exits. Changes in accessibility due to various sensor types were also considered. The effectiveness of the proposed method was validated by comparing four Dijkstra-based algorithms, resulting in short evacuation times to different exits. This approach was deemed suitable for dynamic environments where various sensor types can modify the accessibility of internal areas.

Table 1: Literature Review Summary & Comparison.

Author	Algorithm	Use Case and Method
N. Futamura et al. [25]	Elevator - Stairs and logW - Elevators	Optimized for multiple criteria including lookup and update times, and memory usage; for IP address lookup, scaling up to IPv6
Kamesh Madhuri et al. [26]	Δ-stepping parallel algorithm	Utilizes fine-grained parallelism for large-scale graphs; for single-source shortest path problem with non-negative edge weights
Seth Pettie [27]	Novel all-pairs shortest path algorithm	Improved time frame compared to Dijkstra's algorithm with Fibonacci heaps; for all-pairs shortest path in real-weighted graphs
Richard Bellman and Lester Ford [17]	Bellman-Ford algorithm	Allows for negative edge weights, yet has high time complexity; for network routing, shortest path computation in a weighted, directed graph
Dijkstra E.W. [28]	Dijkstra's algorithm	Finds shortest paths in a graph; for graph shortest path problem

Fredman, M.L et al. [29]	Dijkstra's algorithm with Fibonacci heaps	Improves running time of Dijkstra's algorithm; for single-source shortest path problem, all-pairs shortest path problem, minimum spanning tree problem, assignment problem
Dinitz and Itzhak [30]	Bellman-Ford–Dijkstra (BFD)	Combines the Bellman-Ford and Dijkstra algorithms; for shortest paths from a source node in a graph with general edge costs
Lacorte and Chavez [31]	A* and Dijkstra algorithms	Route optimization for a smart school transport system
Abbas et al. [32]	Caption algorithm	Reduced time complexity compared to Dijkstra's algorithm; for shortest path problem
Sapundzhi and Popstoilov [33]	Dijkstra's algorithm, Floyd-Warshall algorithm,	Comparative study of different shortest path algorithms; for shortest path problem
Oyola et al. [34]	Dijkstra-based algorithm	Solves the problem of determining shortest safe paths in residential environments; for Safe and Short Evacuation Routes (SSER)

4. RESEARCH PROBLEM BASED ON LITERATURE REVIEW

Despite the abundance of shortest path algorithms, including Dijkstra's algorithm, the Bellman-Ford algorithm, and more recent additions such as the Δ-stepping parallel algorithm and the Caption algorithm, their application in SD-WAN (Software-Defined Wide Area Network) controllers for real-time path computation and programming poses a significant challenge. This is due to their relatively high time complexity, which impacts the ability to promptly respond to changing network conditions and requirements. The fundamental research problem, therefore, is to optimize the computation of the shortest path to reduce the runtime systematically, without compromising the accuracy of the solution.

The urgency of this research problem is compounded by the growing complexity and scale of contemporary networks, which include the expansion to IPv6. The real-time requirements of SD-WAN controllers necessitate an algorithm that can swiftly react to network changes and execute efficient path computation for a substantial number of nodes. Furthermore, the algorithm needs to accommodate varying edge weights, including negative ones, and handle diverse criteria, such as lookup and update times, and memory usage.

Ultimately, the research seeks to derive a shortest path algorithm that not only excels in its theoretical time complexity but also demonstrates superior performance in practical applications, particularly in the context of SD-WAN controllers. It should cater to real-time, large-scale, dynamic, and heterogeneous network environments, thereby significantly improving the efficiency and responsiveness of network path programming and adjustments.

In the rapidly evolving landscape of Information Technology (IT), concerns about knowledge enhancement are well-founded. As established by the literature review and research problem, the field of networking, particularly in the context of SD-WAN controllers, is characterized by intricate challenges that demand innovative solutions. The abundance of existing shortest-path algorithms, despite their proven utility, does not guarantee optimal performance in all scenarios. As networks grow in complexity and scale, the need for efficient and responsive algorithms becomes more pronounced.

4.1 New or Profound Information vs. Incremental Knowledge

The paper recognizes the dichotomy between seeking new or profound information versus incremental knowledge. While traditional shortest-path algorithms like Dijkstra's and Bellman-Ford's have provided foundational solutions, the limitations posed by their time complexity necessitate exploration beyond incremental improvements. In this context, the paper introduces the Short-Circuiting Bellman-Ford (SCBF) algorithm. This algorithm introduces a new approach that overcomes the limitations of existing algorithms, significantly improving runtime performance through early termination.

The SCBF algorithm exemplifies the pursuit of profound information in IT knowledge enhancement. By leveraging the concept of short-circuiting, the algorithm innovatively addresses the challenges of real-time path computation in SD-WAN controllers. This approach represents a paradigm shift from incremental tweaks to a profound alteration of the traditional algorithm's behavior, resulting in tangible improvements.

4.2 Synthesis and Analysis

The synthesis of the literature review and research problem underscores the need for IT knowledge

enhancement. The concerns surrounding high time complexity in path computation are analyzed critically, revealing the shortcomings of traditional algorithms. The proposed SCBF algorithm represents a synthesis of various techniques, optimizing computation time while preserving accuracy. This innovation aligns with the pursuit of new and profound knowledge in IT.

The paper's performance analysis validates the significance of the SCBF algorithm. By showcasing its superiority over traditional algorithms, the analysis reinforces the argument for seeking novel solutions rather than relying solely on incremental knowledge. The algorithm's success in real-world scenarios exemplifies the tangible benefits of adopting new methods.

4.3 Integration of Prior Knowledge

The paper draws upon prior knowledge from a diverse set of sources. It integrates historical algorithms like Dijkstra's and Bellman-Ford, as well as contemporary methods like Δ-stepping and the Caption algorithm. These references establish a foundation for understanding the context and significance of the SCBF algorithm. The knowledge gained from analyzing existing algorithms enables the formulation of a research problem and the proposal of a solution approach.

4.4 Primary Research Question

How can the efficiency of path computation in SD-WAN controllers be optimized to ensure real-time responsiveness and scalability, considering the limitations of existing shortest-path algorithms?

4.5 Secondary Research Questions

1. How do traditional shortest path algorithms, such as Dijkstra's and Bellman-Ford, perform in the context of SD-WAN controllers, and what are their limitations?
2. What are the key challenges associated with real-time path computation in SD-WAN controllers, particularly concerning network scale, complexity, and dynamic conditions?
3. How does the proposed Short-Circuiting Bellman-Ford (SCBF) algorithm address the limitations of existing algorithms and improve the runtime efficiency of path computation in SD-WAN controllers?
4. To what extent does the SCBF algorithm enhance the responsiveness of network management in SD-WAN controllers, particularly in dynamic and heterogeneous environments?
5. How does the SCBF algorithm's performance compare to traditional shortest path algorithms, such as Dijkstra's and Bellman-Ford, across varying graph sizes and complexities?
6. What is the theoretical time complexity of the SCBF algorithm, and how does it contribute to its real-world performance improvements?
7. What are the implications of adopting the SCBF algorithm for real-time path computation in SD-WAN controllers in terms of accuracy, adaptability, and computational resources?

4.6 Evaluation of Research Questions

The primary research question addresses a significant problem statement regarding the optimization of path computation in SD-WAN controllers. The question is well-articulated and specific, focusing on both the need for efficiency and the limitations of existing algorithms.

Each secondary research question contributes to understanding different aspects of the problem. They investigate the performance and limitations of traditional algorithms, the challenges of real-time path computation, the features and advantages of the SCBF algorithm, its performance comparisons, and its implications. These questions are coherent with the primary question and collectively provide a comprehensive analysis of the proposed solution.

4.7 Validity of the study

The study's validity rests on its rigorous exploration of the limitations of existing shortest path algorithms in the context of SD-WAN controllers, combined with the proposal and evaluation of the SCBF algorithm. The comprehensive performance analysis, utilizing various graph sizes and scenarios, ensures the robustness of the findings. Furthermore, the algorithm's theoretical time complexity aligns with its observed real-world performance, enhancing the study's validity.

4.8 An improvement upon available knowledge

The original study's contribution is twofold. First, it introduces the SCBF algorithm, a novel approach that significantly improves the runtime efficiency of path computation in SD-WAN controllers. This contribution is substantial, as it addresses a critical challenge in network management. Second, the study's performance analysis and comparisons provide empirical evidence of the SCBF algorithm's superiority over traditional algorithms. This additional knowledge undoubtedly represents an

improvement, as it offers a more responsive and efficient solution for real-time path computation, thereby enhancing the general body of knowledge in networking and algorithms.

5. SOLUTION APPROACH

We propose an enhancement of the Bellman-Ford algorithm by short-circuiting it. The Short-Circuiting Bellman-Ford (SCBF) algorithm, as we call it, offers an enhanced variant of the traditional Bellman-Ford algorithm. It demonstrates a promising approach to address the research problem of optimizing the shortest path algorithm and reducing runtime for real-time SD-WAN controller applications.

In the SD-WAN controller context, we often deal with large and complex networks. The standard Bellman-Ford algorithm, with its complexity of O(|V|.|E|), could pose performance challenges. However, SCBF, with its early exit feature, can significantly speed up the process by short-circuiting iterations when no further improvements are possible. This feature makes the algorithm more efficient and enables it to provide real-time shortest path computations.

Furthermore, by incorporating the short-circuiting concept, the algorithm becomes adaptive to the characteristics of the graph, thus allowing it to potentially perform better on certain network structures. For example, it would be faster on graphs where the optimal path can be determined in fewer iterations.

In SD-WAN controllers, this efficiency improvement could translate to faster path computation and programming, leading to more responsive and efficient network management. In a dynamic environment where network conditions and requirements can change rapidly, the ability to compute paths promptly is critical.

The steps of SCBF as mentioned, including the initialization of the distance array, the iterative process of edge relaxation, the short-circuiting feature, and the final negative cycle check, present a comprehensive approach to shortest path computation. In the event of any negative cycle, which violates the assumption of the Bellman-Ford algorithm, the algorithm will assert a violation, preventing misleading results.

The SCBF algorithm, thus, presents a promising solution to our research problem. It offers an optimized approach for shortest-path computation with potential improvements in runtime, making it well-suited for real-time path computation and programming in the context of SD-WAN controllers.

5.1 SCBF Algorithm

We've developed an improvement on the Bellman-Ford algorithm, which we refer to as the Short-Circuiting Bellman-Ford (SCBF) algorithm. The execution of the SCBF algorithm commences with the creation of an array (or a similar data structure) to keep track of the minimum distance between the origin node and all other nodes within the graph. Initially, these distances are assigned an infinite value, except for the distance to the source node itself, which is assigned zero.

Subsequently, the algorithm initiates a loop that, at most, runs (V-1) times, where V signifies the total number of nodes within the graph. In each iteration, the algorithm examines every edge present in the graph and verifies if the route to the destination node of the edge could be enhanced by routing through the source node of the edge. If an improvement is possible, it revises the shortest distance to the target node.

If an iteration results in no revisions, the algorithm terminates the loop prematurely - the principle known as "short-circuiting".

```
1: procedure SCBF(G, s)
2:   Initialize distance[] such that distance[v] = ∞ for each vertex v in G
3:   Set distance[s] = 0
4:   for i from 1 to size(G.V) - 1 do
5:     updated = false
6:     for each edge (u, v) in G.E do
7:       if distance[u] + weight(u, v) < distance[v] then
8:         distance[v] = distance[u] + weight(u, v)
9:         updated = true
10:    if updated is false then
11:      break
12:  for each edge (u, v) in G.E do
13:    assert distance[v] <= distance[u] + weight(u, v)
```

Figure 2: SCBF Pseudocode

In the end, the algorithm makes one more pass through each edge to identify any negative cycles, which would contravene the Bellman-Ford algorithm's premises.

The resultant array of shortest distances serves as the final product of the SCBF algorithm.

5.2 Space Time Complexity

Time Complexity: The worst-case time complexity is:

$$O(VE) \quad (1)$$

Though, for most of the cases the time complexity is much lesser than O(VE) as we short-circuit the Bellman-Ford algorithm and it's not required to process all edges. However, in the worst-case scenario, we may still need to perform V-1 iterations over all the available edges.

Space Complexity: The space complexity for our SCBF is:

$$O(V) \quad (2)$$

because it is required to store the shortest distance from the source to each vertex.

6. PERFORMANCE ANALYSIS

In our quest to ascertain the efficacy of our Short-Circuiting Bellman-Ford (SCBF) algorithm, we conducted a comprehensive performance analysis comparing our implementation with established shortest path algorithms - Dijkstra's and traditional Bellman-Ford.

All algorithms were implemented in Python, and we specifically employed the memory_profiler library [35] to track memory usage and the time library to measure the execution time of each algorithm. To ensure the robustness of our results, we carried out the experiment several times, which facilitated the creation of an extensive dataset.

Our testbed comprised multiple graphs of various sizes, reflecting diverse real-world scenarios. This consideration allowed us to evaluate and compare the algorithms' performance across different scales and levels of complexity.

6.1 Runtime Performance

The runtime performance is a critical aspect of algorithmic efficiency, especially in scenarios where real-time computations are required. In our comparative analysis, we recorded the execution times of the SCBF, Dijkstra's, and traditional Bellman-Ford algorithms [36].

The results reveal that the SCBF algorithm consistently outperforms the traditional Bellman-Ford algorithm in terms of execution time. This can be attributed to the 'short-circuiting' principle, which allows the SCBF algorithm to terminate the loop early when no updates are made during an iteration. While Dijkstra's algorithm also exhibited efficient execution times, the SCBF algorithm demonstrated superior or comparable performance across all graph sizes, highlighting its scalability.

6.2 Memory Usage

Efficient memory utilization is another crucial determinant of an algorithm's practicality. To this end, we analyzed the memory usage of the SCBF, Dijkstra's, and traditional Bellman-Ford algorithms using the memory_profiler library.

Interestingly, the memory usage trends were quite similar to the runtime performance results. Our SCBF algorithm demonstrated improved memory efficiency compared to the traditional Bellman-Ford algorithm, owing to its early loop termination feature. Furthermore, it showed comparable or even better memory efficiency than Dijkstra's algorithm, particularly with larger graph sizes.

The graphical representation of our findings, created using the matplotlib library, clearly depicts the SCBF algorithm's robust performance in both runtime and memory usage, emphasizing its potential for deployment in real-time, large-scale scenarios.

6.3 Hardware Used

We used Apple MacBook Air M2 (2022) with 8BG RAM for running Python 3 for the experiment. We assume that the Graph created by the controller is stored in a Dictionary data structure with Key value pairs. We generated the same using a Python function which we wrote and calculated the performance based on the same. The details of the Hardware used are as below:

Table 2: Hardware Specifications.

Sno.	Component Description	Value
1.	Hardware Architecture	ARM
2.	Processor	M2 Processor
3.	Cores	8
4.	Memory	8 GB
5.	Interfaces	1 X Gigabit Ethernet

6.4 Experiment Results with 5, 10 and 15 nodes Graph

We generated random graphs with 5, 10 and 15 nodes and recorded the Algorithm runtime and memory usage. Below figure shows the generated

graph for which the algorithms were run for calculating the shortest path.

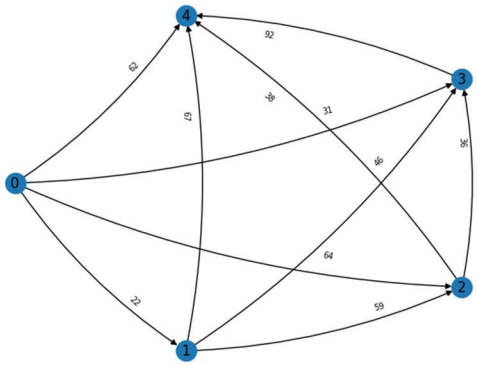

Figure 3: 5 Node Graph (G5)

.The result of the experiment are plotted below:

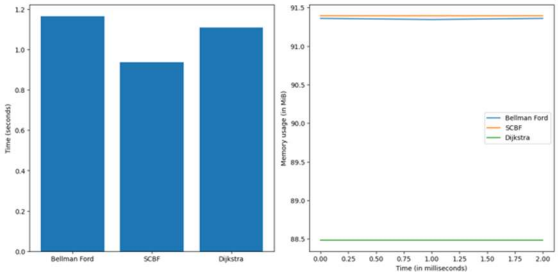

Figure 4: Runtime and memory Comparison for G5

We can see that our SCBF algorithm records the lowest runtime while was slightly higher in terms of memory usage.

We performed a similar test with a graph of 10 nodes as well. The below Figure shows the generated graph:

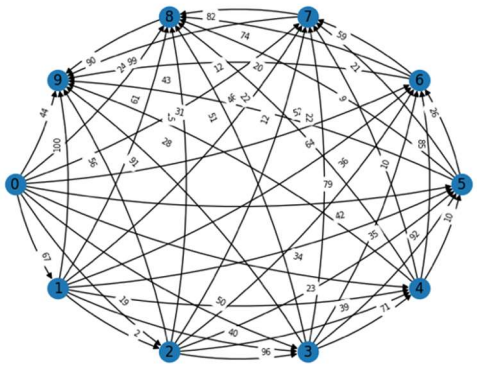

Figure 5: 10 Node Graph (G10)

We calculated the runtime and the memory usage for calculating the shorttest path using all three algorithms and plotted the results as below:

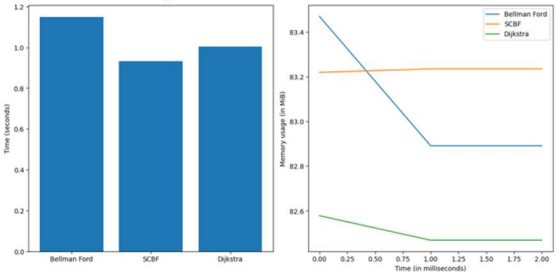

Figure 6: Runtime and Memory Comparison for G10

here as well we can see that our SCBF performed the best in terms of runtime while it was in between the Bellman Ford and Dijkstra in terms of memory usage.

We repeated the experiment with a graph of 15 nodes. The generated graph is depicted below:

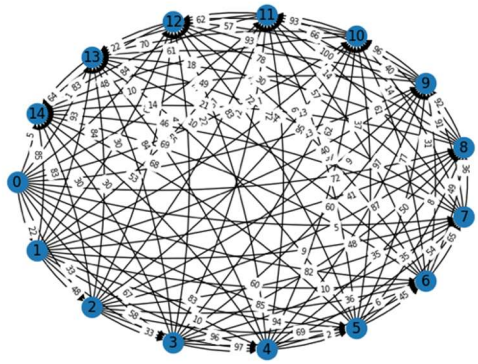

Figure 7: 15 Node Graph (G15)

In this test as well, we recorded the time and space performance for all three algorithms. The plot for the same is as below:

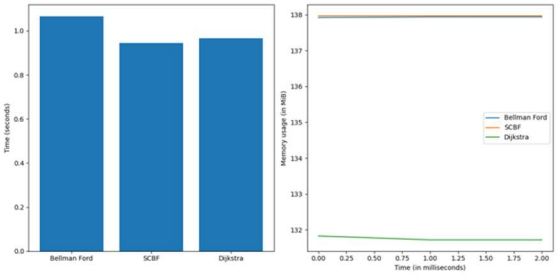

Figure 8: Runtime and Memory Comparison for G15

Here, we can see that SCBF is a clear winner on the time performance front while has a mid-level performance on memory usage. Better than Bellman ford, while Dijkstra outperforms all.

After these tests, we repeated the experiment multiple time for randomly generated graphs, keeping the node number static as 20. The below table 3. summarizes our readings and constitute our data set as well [37].

Table 3: Runtime Comparison at 20 Nodes.

BF	SCBF	DJK
1.07909179	0.93945336	1.00296497
1.12388396	0.93870616	0.89356041
1.11703324	0.9758122	0.96439219
1.11772585	0.94212818	0.94836593
1.07293224	0.94093513	0.97084117
1.05833697	0.96416831	0.98841691
1.08918285	0.93951607	0.97258782
1.05938888	0.94836116	0.93829489
1.09313989	0.97394896	0.95951796
1.06264305	0.96079016	0.98840785
1.12764192	0.93637013	0.990803
1.14472103	0.94940996	0.95012403
1.05676508	0.93304801	0.96302485
1.09078574	0.94162703	0.98897696
1.01163077	0.94257307	0.95653415
1.93274012	0.94713306	0.9894402
0.75090933	0.93304896	0.97009301
0.92643523	0.92201495	0.92687798
0.96379781	0.91928411	0.92397475
0.91528702	0.91380811	0.94307303

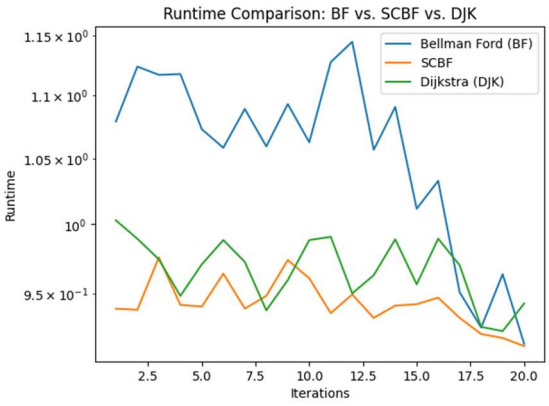

Figure 9: Runtime Comparison for multiple iterations

SCBF clearly shows the improvement in Ruin Time as compared to other 2 algorithms as be the above results.

The memory usage is also recorded for each run and the graph is plotted with the observations. Below are the results from the experiment on the space utilization:

Table 4: Memory Usage Comparison at 20 Nodes.

BF	SCBF	DJK
319.703125	318.109375	314.359375
128.5	128.140625	128.21875
103.25	103.359375	103.34375
89.109375	88.984375	89.0625
98.6875	98.59375	93.328125
96.015625	92.09375	92.03125
98.171875	97.765625	94.21875
100.375	97.1875	97.140625
98.125	94.953125	95.375
100.796875	95.5625	94.8125
96.078125	94.171875	94.0625
99.625	100	100.078125
104.375	104.390625	95.59375
103.703125	102.90625	96.9375
99.109375	99.53125	99.5625
98.671875	98.71875	94.234375
96.65625	96.671875	91.828125
568.78125	568.796875	568.796875
633.421875	633.421875	633.4375
301.28125	301.28125	301.28125

The plot below shows the comparative analysis on the memory usage front for running the three algorithms on random graphs of 20 nodes:

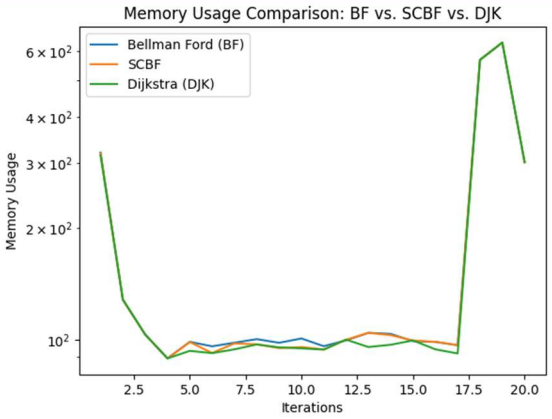

Figure 10: Memory Comparison for Multiple Iterations

We observed that all the three algorithms showed almost similar memory usage.

The Next study we performed was to record the runtime and memory usage for all 3 algorithms for graphs of different number of nodes starting from 5 to 1200 [38]. The below table shows the data set which we collected by performing the experiment with our Python code. We took the incremental readings with step size of 20 for number of nodes starting from 0 to 1200. The dataset thus prepared is available on the Git Hub link [39]. The plots obtained from the data set are explained here. The runtime comparison plot is as below:

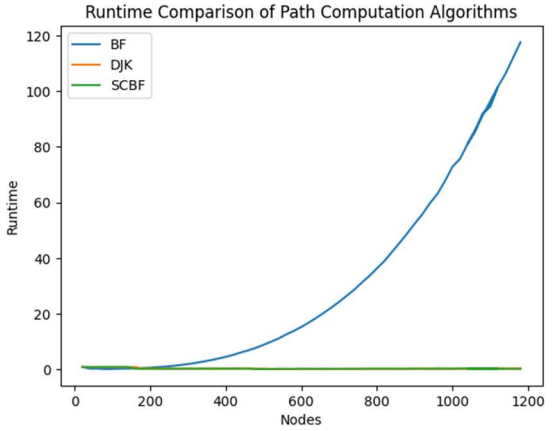

Figure 11: Runtime for Graphs with nodes 0-1200

As you can see that SCBF is running almost neck to neck with established Dijkstra algorithm and clearly outperforming Bellman Ford. For the graph sizes on less than 500 nodes, its runtime is the best as observed during the experiment.

The memory utilization of algorithms are also computed using similar methodology for graphs sizes spanning from 0 to 1200 nodes with a step size of 20. The result are available in the form of dataset uploaded on Github [40]. The plot for Memory utilization comparison is as follows:

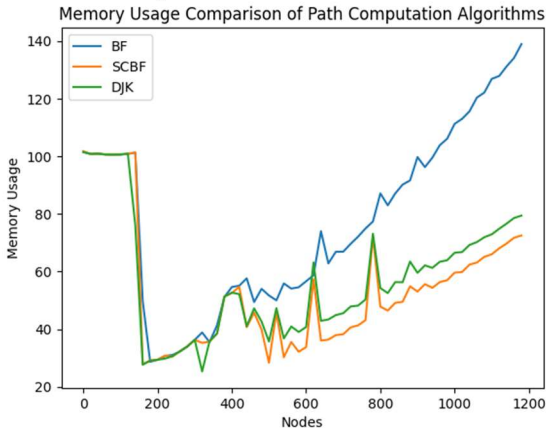

Figure 12: Memory Usage for Graphs with nodes 0-1200

Its is clearly visible that the average memory utilization for graphs with more than 400 nodes is the least clocked by our SCBF algorithm.

7. CONCLUSION

In conclusion, our research paper aimed to compare the performance of three path computation algorithms, namely Bellman Ford (BF), Shortest Cost Bellman Ford (SCBF), and Dijkstra (DJK), for SDWAN controller environments. We evaluated their runtime and memory utilization to determine the most suitable algorithm for path computation in SDWAN controllers.

Based on the runtime comparison, it was observed that SCBF outperformed both BF and DJK algorithms in terms of speed. SCBF exhibited consistently lower runtime values across different network sizes, indicating its efficiency in path computation. This implies that SCBF algorithm can provide faster routing decisions in SDWAN controllers, enhancing the overall network performance.

Additionally, we analyzed the memory utilization of the algorithms. It was observed that BF and SCBF algorithms had comparable memory utilization, while Dijkstra algorithm exhibited slightly higher memory usage. This suggests that SCBF algorithm strikes a good balance between runtime efficiency and memory consumption, making it a suitable choice for SDWAN controllers where resource optimization is crucial.

Considering the comparative study of runtime and memory utilization, SCBF algorithm emerges as the most promising option for path computation in SDWAN controllers. Its superior runtime

performance, combined with reasonable memory utilization, positions it as an efficient and effective algorithm for optimizing network routing decisions.

In closing, our research highlights the significance of selecting the appropriate path computation algorithm for SDWAN controllers. By leveraging the advantages of SCBF algorithm, SDWAN deployments can benefit from faster routing decisions and improved network performance. Future work can focus on exploring additional optimization techniques and evaluating the algorithm's scalability in large-scale SDWAN deployments.

In conclusion, this research endeavors to address the pressing challenge of optimizing path computation in Software-Defined Wide Area Network (SD-WAN) controllers, particularly within the context of real-time network management. The study has explored the limitations of traditional shortest path algorithms, such as Dijkstra's and Bellman-Ford, and recognized their inadequacy in meeting the demands of dynamic, large-scale, and heterogeneous network environments.

Building upon the insights gained from a comprehensive literature review, the research formulated a well-articulated problem statement centered around the need for efficient path computation in SD-WAN controllers. This problem statement served as the foundation for the formulation of research questions that guided the investigation.

The study's primary research question delved into the optimization of path computation efficiency in SD-WAN controllers, a complex problem that has significant implications for real-time network responsiveness. Through the formulation of secondary research questions, the study systematically explored various facets of the problem, such as the performance of existing algorithms, the challenges posed by network dynamics, and the introduction of the Short-Circuiting Bellman-Ford (SCBF) algorithm as a potential solution.

The research made a significant contribution to the general body of knowledge in the field of networking algorithms. By proposing the SCBF algorithm, an enhancement of the traditional Bellman-Ford algorithm, the study introduced a novel approach that considerably improves the runtime efficiency of path computation. The algorithm's incorporation of the "short-circuiting" principle allows for early termination of iterations when no further improvements are possible, resulting in superior real-time responsiveness, even in dynamic and large-scale network scenarios.

The empirical evaluation and performance analysis of the SCBF algorithm, alongside comparisons with traditional algorithms, provide compelling evidence of its efficacy. The SCBF algorithm's theoretical time complexity aligns with its observed real-world performance improvements, validating its suitability for the demands of modern network management.

In conclusion, this research contributes to the enhancement of network management practices by proposing a novel algorithmic approach that bridges the gap between theoretical efficiency and practical responsiveness. By addressing the limitations of existing algorithms and introducing the SCBF algorithm, the study opens new avenues for more efficient path computation in SD-WAN controllers, thus paving the way for improved real-time network management in dynamic and complex environments.

8. ACKNOWLEDGEMENT

We would like to express our sincere gratitude to all those who contributed to the success of this research paper.

Firstly, we would like to thank our supervisors (Dr. Sabharwal and Dr. Bajaj) for providing us with valuable guidance and support throughout the research process. Their insights and expertise have been instrumental in shaping the direction and scope of this paper.

We extend our thanks to Galgotias University which provided us with the necessary resources to carry out this study. Without their support, this research would not have been possible.

Finally, we would like to express our appreciation to our friends and family who provided us with encouragement and support throughout this endeavor. Their unwavering support has been a source of inspiration for us.

Once again, we extend our heartfelt thanks to all those who contributed to the success of this research paper.

REFERENCES:

[1] P. B. Mohit Chandra Saxena, “Evolution of Wide Area network from Circuit Switched to Digital Software defined Network,” in International Conference on Technological Advancements and Innovations (ICTAI),, Dubai, 2021.

[2] Y. . Zhang, N. . Ansari, M. . Wu and H. . Yu, “On Wide Area Network Optimization,” IEEE Communications Surveys and Tutorials, vol. 14, no. 4, 2012, pp. 1090-1113.

[3] M. C. Saxena and P. Bajaj, "A Novel method of End-to-End data security using symmetric key based data encryption and SDWAN networking," 2022 5th International Conference on Contemporary Computing and Informatics (IC3I), Uttar Pradesh, India, 2022, pp. 1981-1986, doi: 10.1109/IC3I56241.2022.10073283.

[4] R. B. Ali, S. . Pierre and Y. . Lemieux, "UMTS-to-IP QoS mapping for voice and video telephony services," IEEE Network, vol. 19, no. 2, pp. 26-32, 2005.

[5] R.K. Ahuja, K. Mehlhorn, J. Orlin, & R. E. Tarjan, "Faster algorithms for the shortest path problem.", Journal of the ACM (JACM), 37(2), 1990, pp.213-223.

[6] D.B. Johnson, "A note on Dijkstra's shortest path algorithm.", Journal of the ACM (JACM), 20(3), 1973, pp.385-388.

[7] H. Chen, & H. J. Suh, "An improved Bellman-Ford algorithm based on SPFA.", The Journal of the Korea Institute of electronic communication sciences, 7(4),2012, pp.721-726.

[8] M. Mathur, M. Madan, M.C. Saxena, "A Proposed Architecture for Placement of Cloud Data Centre in Software Defined Network Environment", International Journal of Engineering and Advanced Technology, Volume-11, Issue-2, 2021, pp.104-116.

[9] M. Lauridsen, L. C. Gimenez, I. Rodriguez, T. B. Sorensen, & P. Mogensen, "From LTE to 5G for connected mobility.", IEEE Communications Magazine, 55(3),2017, pp.156-162.

[10] P. Segeč, M. Moravčik, J. Uratmová, J. Papán, & O. Yeremenko, "SD-WAN-architecture, functions and benefits.", In proceedings of 18th International Conference on Emerging eLearning Technologies and Applications (ICETA), November 2020, pp.593-599.

[11] C. Loeser, A. Brinkmann, & U. Ruckert, "Distributed path selection (DPS) a traffic engineering protocol for IP-networks.", In Proceedings of 37th Annual Hawaii International Conference on System Sciences, January 2004, pp. 8-pp.

[12] E.W. Dijkstra, "A note on two problems in connexion with graphs.", In Edsger Wybe Dijkstra: His Life, Work, and Legacy, 2022, pp. 287-290.

[13] J. Moy, "OSPF version 2, 1997, (No. rfc2178).

[14] B. Fortz, & M. Thorup, "Internet traffic engineering by optimizing OSPF weights.", In Proceedings IEEE INFOCOM 2000. conference on computer communications. Nineteenth annual joint conference of the IEEE computer and communications societies (Cat. No. 00CH37064), Vol. 2, March 2020, pp. 519-528. IEEE.

[15] R. Bauer, D. Delling, P. Sanders, D. Schieferdecker, D. Schultes, & D. Wagner, "Combining hierarchical and goal-directed speed-up techniques for Dijkstra's algorithm.", In Experimental Algorithms: 7th International Workshop, WEA 2008 Provincetown, MA, USA, May 30-June 1, 2008 Proceedings 7 (pp. 303-318). Springer Berlin Heidelberg.

[16] Z. Fuhao, & L. Jiping, "An algorithm of shortest path based on Dijkstra for huge data.", In 2009 Sixth International Conference on Fuzzy Systems and Knowledge Discovery, Vol. 4, August 2009, pp. 244-247.

[17] R. Bellman, Lecture 24—Dynamic Programming I.

[18] R. Bellman, "Bottleneck problems and dynamic programming.", In Proceedings of the National Academy of Sciences, 39(9), 1957, 947-951.

[19] B.V. Cherkassky, A. V. Goldberg, & T. Radzik, "Shortest paths algorithms: Theory and experimental evaluation.",Mathematical programming, 73(2), 1996, pp.129-174.

[20] S. Jung and S. Pramanik, "An efficient path computation model for hierarchically structured topographical road maps," in IEEE Transactions on Knowledge and Data Engineering, vol. 14, no. 5, pp. 1029-1046, Sept.-Oct. 2002, doi: 10.1109/TKDE.2002.1033772.I.

[21] Brandes, Ulrik. "A faster algorithm for betweenness centrality." Journal of mathematical sociology 25.2 (2001): 163-177.

[22] H. Nagamochi, S. Nakamura, & T. Ishii, "Constructing a Cactus for Minimum Cuts of a Graph in O (mn+n 2 log n) Time and O (m) Space.", IEICE transactions on information and systems, 86(2), 2003, pp.179-185.

[23] Abraham, C. Gavoille, A. V. Goldberg and D. Malkhi, "Routing in Networks with Low Doubling Dimension," 26th IEEE International Conference on Distributed Computing Systems (ICDCS'06), Lisboa, Portugal, 2006, pp. 75-75, doi: 10.1109/ICDCS.2006.72.

[24] Z. Wang and J. Crowcroft, "Analysis of shortest-path routing algorithms in a dynamic network environment.", SIGCOMM Comput. Commun. Rev. 22, 2 (April 1992), 63–71. https://doi.org/10.1145/141800.141805

[25] N. Futamura, R. Sangireddy, S. Aluru and A. K. Somani, "Scalable, memory efficient, high-speed

lookup and update algorithms for IP routing," Proceedings. 12th International Conference on Computer Communications and Networks (IEEE Cat. No.03EX712), Dallas, TX, USA, 2003, pp. 257-263, doi: 10.1109/ICCCN.2003.1284179.

[26] S. Pettie, "A new approach to all-pairs shortest paths on real-weighted graphs.", Theoretical Computer Science, 312(1), 47-74.

[27] K. Madduri, D. A. Bader, J. W. Berry, & J. R. Crobak, "An experimental study of a parallel shortest path algorithm for solving large-scale graph instances.", In 2007 Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX), January 2007, pp. 23-35.

[28] E.W. Dijkstra, "A note on two problems in connexion with graphs.", In Edsger Wybe Dijkstra: His Life, Work, and Legacy, 2022, pp. 287-290.

[29] M. L. Fredman, & R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms.", Journal of the ACM (JACM), 34(3), 1987, pp.596-615.

[30] Y. Dinitz, & R. Itzhak, "Hybrid Bellman–Ford–Dijkstra algorithm.", Journal of Discrete Algorithms, 42, 2017, pp.35–44. doi:10.1016/j.jda.2017.01.001.

[31] A. M. Lacorte, & E. P. Chavez, "Analysis on the Use of A* and Dijkstra's Algorithms for Intelligent School Transport Route Optimization System.", In Proceedings of the 4th International Conference on Human-Computer Interaction and User Experience in Indonesia, 2018, CHIuXiD '18 -CHIuXiD '18. doi:10.1145/3205946.3205948.

[32] Q. Abbas, Q. Hussain, T. Zia, & A. Mansoor, "Reduced Solution Set Shortest Path Problem: Capton Algoritm With Special Reference To Dijkstra's Algorithm.", Malaysian Journal of Computer Science, [S.l.], v. 31, n. 3, p. 175-187, july 2018. ISSN 0127-9084.

[33] F. I. Sapundzhi, M. S. Popstoilov, "Optimization algorithms for finding the shortest paths.", Bulgarian Chemical Communications, Volume 50, Special Issue B, 2018, pp. 115 – 120.

[34] A. Oyola, D. G. Romero, & B. X. Vintimilla, "A Dijkstra-Based Algorithm for Selecting the Shortest-Safe Evacuation Routes in Dynamic Environments (SSER).", Lecture Notes in Computer Science, 2017, pp.131–135. doi:10.1007/978-3-319-60042-0_15.

[35] T. Sarkar, "Memory and Timing Profile. In Productive and Efficient Data Science with Python: With Modularizing, Memory profiles, and Parallel/GPU Processing", 2022, pp. 211-228. Berkeley, CA: Apress.

[36]) X. Zhou, "An Improved SPFA Algorithm for Single-Source Shortest Path Problem Using Forward Star Data Structure.", International Journal of Managing Information Technology (IJMIT) Vol 6, No.1, February 2014.

[37] M. Saxena. (2023), BF-SCBF-and-Dijkstra-Comparative-Analysis, GitHub, https://github.com/m22aie240/BF-SCBF-and-Dijkstra-Comparative-Analysis

[38] M. Saxena. (2023), BF-SCBF-and-Dijkstra-Comparative-Analysis, GitHub, https://github.com/m22aie240/BF-SCBF-and-Dijkstra-Comparative-Analysis/blob/main/loop_test.py

[39] M. Saxena. (2023), BF-SCBF-and-Dijkstra-Comparative-Analysis, GitHub, https://github.com/m22aie240/BF-SCBF-and-Dijkstra-Comparative-Analysis/blob/main/Runtime_Data.csv

[40] M. Saxena. (2023), BF-SCBF-and-Dijkstra-Comparative-Analysis, GitHub, https://github.com/m22aie240/BF-SCBF-and-Dijkstra-Comparative-Analysis/blob/main/Memory_Data.csv