Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

ARTICLE TYPE

A Centralised Hybrid Routing Model for Multi-Controller SD-WANs

Saptarshi Ghosh | Muddesar Iqbal | Tasos Dagiuklas

[1]Division of Computer Science and Informatics, School of Engineering, London South Bank University, United Kingdom

Correspondence
Saptarshi Ghosh.
Email: ghoshs4@lsbu.ac.uk
Muddesar Iqbal.
Email: m.iqbal@lsbu.ac.uk
Tasos Dagiuklas
Email: tdagiuklas@lsbu.ac.uk

Summary

The increasing complexity of contemporary mobile and application-centric networks is pushing traditional WAN architectures to its limit. Due to the rise of Cloud-based services and Edge-computing, the inter-site data transfer volume in enterprise networks is rising rapidly. Moreover, enterprises are tending to integrate several communication technologies such as Broadband, LTE, MPLS, etc. for a high-available dynamic WAN connectivity, which a traditional WAN doesn't support natively. Additionally, its distributed computing model results in the Control-plane traffic to consume a significant amount of backhaul bandwidth. Software-Defined WAN (SD-WAN) is a generation shift that adapts the centralized model of SDN to WAN that fills the bottlenecks of its predecessor. It provides over twice the bandwidth having the same backhaul [1] with more managibility, autonomy, and security of the network. In this article, we propose a hybrid routing model for multi-controller SD-WANs that computes all-possible routes proactively and serve them on-demand. It results in a rapid convergence across the edge devices. The article further discusses the synchronization mechanism among several controllers and a testbed implementation to conduct the experiments.

KEYWORDS:
SD-WAN, MEC,Hybrid Routing

1 | INTRODUCTION

Telecommunication industries across the world are going through a massive transformation phase. With the increasing demand for high-quality online content like streaming from "Over the Top" (OTT) platforms (e.g. Netflix, Amazon Prime, Youtube), are driving the telcos to optimize their existing network architectures. Content Ditribution Network (CDN) that caches static contents into a proxy server e.g. Point of Presence (PoP) enables various alternatives to reduce the delays. A clever and possibly most widely used method is content-caching [2], that keeps the response of most frequently requested contents and serves them from local storage rather redirecting towards the origin. Research shows the use of a predictive approach may reduce the cache overhead value per day up to a fraction (10% - 20%) of the cache size [3].

Multi-acces Edge Computing (MEC) [4] enhances the availability of cloud-service by distributing them into the edge, bringing them closer to the mobile end users. MEC nodes host several virtualized services and attach them to a dedicated network called the Backhaul network. SDN based backhauling enables a centralised control, which results control-plane functionalities such as

Routing, Policying, etc. to execute as services. Unlike traditional distributed computing models where network devices synchronise by exchanging control packets, SDN decouples the control plane, from the forwarding plane.[?] . Every network device sends its local information to the controller to compute the forwarding logic. This results in a higher degree of autonomy and programmability in a network configuration, this includs rapid policy deployment, that collectively reduce the CapEx and OpEx.[5]. A logically centralized cluster of remote servers (SDN-Controllers) hosts the control plane. The SDN controller interfaces with applications and hardware switches via the North-Bound (NBI) and South-Bound (SBI) interfaces respectively. The communication method for NBI is typically RESTFul, where the SBI uses Openflow. Applications send generic RESTful configuration requests to the controller defined by a policy, which then gets translated in device-specific configuration and injected into the devices. Therefore, SDN-controllers play a principal role in abstracting the granularity of the network infrastructure and ease the configuration for application developers.

With the introduction of Network Function Virtualization (NFV), it is now possible to virtualize network functions (VNF) and to host them in a remote compute platform (e.g. Cloud, Remote servers etc.). However, not all network functions such as Radio, Sensor, etc. can be virtualized. For instance, Virtualized appliances like routers (e.g. Cisco CSR, HP VSR, Juniper vEX, Quagga, etc.), switches (e.g. Cisco Nexus-9k, 10k, HP Flex-Fabric, Cumulus, etc.), and firewalls (e.g. Cisco ASAv, PF-Sense, etc.) are a pretty common sight in production networks.[6]. The Orchestrator program manages o the VNFs in a MEC environment, such as optimal placement, resource allocation, and provisioning.[7]. A challenge in SDN-MEC based design of CDN is to optimize the forwarding traffic. SDN offers a birds-eye view of the network, which eases the traffic control by taking forwarding decisions at the control plane. The high-level traffic management also leverages optimal connectivity under mobility conditions, using efficient ad-hoc routing techniques. Reactive routing is a type of ad-hoc routing process that discovers routes on-demand, whereas proactive routing discovers routes before applying them. SDN is also effective for hardware independence. For a mobile ad-hoc network in a simulated LTE test-bed, the SDN-NFV approach demonstrates better agility for high volume data than of a non-SDN based one, which stands superior in large-signal load.[8].

Although SDN offers a wide range of benefits over traditional network models, it falls short in implementational acceptance. Giant-sized network infrastructure owners such as service providers, data Centers, and telcos are reluctant to scrap all their existing non-SDN-compliant forwarding hardware for the sake of enjoying the SDN benefits[1]. The primary reasons are first, the cost of re-investment over the expected profit from service quality escalation, and second, the resource spending to retrain for a smooth operational transition. This results in the Overlay-SDN model (initially introduced by VMWare through their NSX platform[9]), which cancels the need to replace the Data-plane devices, rather creating a virtual overlay network that connects it to the control plane. The overlay tunnels enable the edge devices, i.e. routers and layer 3 switches, to communicate with the controller using the Internet as a fabric. The control plane gets segregated, as the global controls reside to the remote-orchestrator, whereas the device-specific controls stay in the edge devices. The resultant architecture is called Software-defined WAN or SD-WAN[10 11]. In SD-WAN architecture, the orchestrator hosts the application plane and interacts with the controller cluster. Application plane performs network operations like routing and sends generic results to the controller. The controller then translates it to device-specific commands and pushes it to the downstream edge nodes. Cisco uses Overlay Management Protocol (OMP)[12] and Citrix uses Adaptive Transport Protocol[13] for this purpose.

SD-WAN architecture leverages the centralised routing model where edge-devices do not exchange control information, rather they update the central controller. Routing as a part of the Layer-3 operations executes within the controller. This surfaces a fundamental problem in adapting traditional routing protocols such as OSPF[14] and EIGRP[15] which are inherently distributed in nature. This opens up a new dimension in the routing protocol design philosophy that aims to compute routes from a centralised perspective. This is not to be misinterpreted by drawing parallel to some of the centralised mechanism in traditional routing such as Designated Routers in OSPF, Route-Reflector in BGP, Root-Brige in Spanning Tree protocol or Next-hop Server in DMVPN. In all the mentioned cases, the central node's job is to collect and distribute information network information, the ultimate computation is done on the nodes in a distributed fashion. SDWAN Routing on the other hand calculates routes on an aggregates topology that is build by fusing information from individual edge nodes, and configure the routing tables to the edge. This paper presents a centralised rapid-convergence routing algorithm (*MRoute*) for SD-WAN[16 17] that proactively finds all-possible paths for all pairs of nodes, rank them and updates rank over time and serve routes on-demand.Furthermore, a multi-controller implementation *MRoute* is also presented. Runtime performance compared with OSPF and EIGRP emulating them on a SDN testbed that comprises[18] IaaS Cloud, Opendaylight[19] as the controller, and Mininet[20] as the forwarding plane.

1.1 | Contributions & Organization

In this research, we have further proposed a solution to some key issues of SD-WAN with CDN as a test case scenario.

1. A model of sharing routing information in a multi-controller SD-WAN.
2. A hybrid routing algorithm that proactively calculates all-possible paths between all pair of nodes and reactively server them on demand.
3. An SD-WAN test-bed to implement, experiment and benchmark the proposed model.

The remainder of the paper is organised as follows, Section II gives an overview of the related works in this context, Section III discusses the mathematical modeling and the convergence of the proposed algorithm. section IV devises the *MRoute* algorithm, that calculates all-possible paths for all-pair of nodes, and proposes an space-efficient data-structure to maintain and allow dynamic ranking of routes. Section V addresses the update mechanism among several controllers. We conclude by section VI that covers experimental setup and results.

2 | RELATED WORKS

In the recent past, several studies have shown their interest in the area of performance optimization of mobile networks within the perimeter of SDN. Management and Orchestration (MANO) projects such as Cloud Band[21 22], OpenMano[23], etc. lack the presence of distributed management[24].On the other hand, The research on a route optimization has become versatile, e.g. a Broker based routing on the SDN controller, a hierarchical controller design that can optimize the traffic up to 50% than a non-hierarchical one.[25].

Egilmez Et.al. in[26], presents an Inter-Domain Routing algorithm over an open-flow networks using (i) topology aggregation and Link summarization, (ii) flow-based end-to-end QOS provision over multi-domain networks. The authors uses optimization based on Lagrange Relaxation Algorithm (LRA) to find the best *n*-level QoS route. The article[27], introduces tenant isolation and flows prioritization using spanning tree to generate routes. the author presents an improved version of LRA with a heuristic technique called *"Min-cost QoS routing algorithm (MCQRA)"*, the is limited to operate in a single controller environment. The authors in[28], proposed an intelligent CDN-SDN architecture, that uses an *Intelligent Center (IC)* on top of the SDN controller. SDN controllers relay the topology, delay, and traffic information into the IC, which response to the SDN controllers with selected path. The controller then writes it onto the forwarding devices. The centralized nature of this approach suffers from the risk of Single-point-of-failure and can also face performance degradation while scaling the control plane.

A multi-controller flow scheduling scheme is also proposed in[29] that uses a Flat-Tree architecture over a single/multi-controller platform. It uses a fixed number of switches to create pods for intercommunication between the switches. Since the controllers coordinate via the pods instead of having dedicated networks, with high hand-off probability, the inter- controller communication traffic would share the switch-to-switch communication.

A multi-domain mobility management scheme[30] is also proposed with a reactive controller co-ordination approach where a controller only communicates with the neighbors when users migrate and results in increased traffic due to a frequent handover scenario. In our previous work, an efficient resource provisioning scheme over MEC is proposed that avoids deadlock condition[31], an efficient content-caching algorithm is introduced that uses MEC collaboration. To minimize traffic load among MEC instances and a Routing algorithm[32] which makes use of Node and Link costs to find best-path over an SDN, we devised *STEN*[33] for this purpose, which has also used to cater rapid-convergence in a network with high resource-variance, SDN-SIM[34]

Apart from the route optimization, security, and incorporation of machine learning also leverage the centralized-control model of SDN. Qian *et. al.* proposed ReFeR[35], a flow monitoring system for preventing Distributed Denial of Service (DDoS) attacks in SDN. Zhang[36] *et.al.* proposed a collaborative architecture in the Edge for performing AI-intensive tasks by offloading them on-demand.

3 | SYSTEM ARCHITECTURE & MATHEMATICAL MODEL

3.1 | Use-Case Architecture

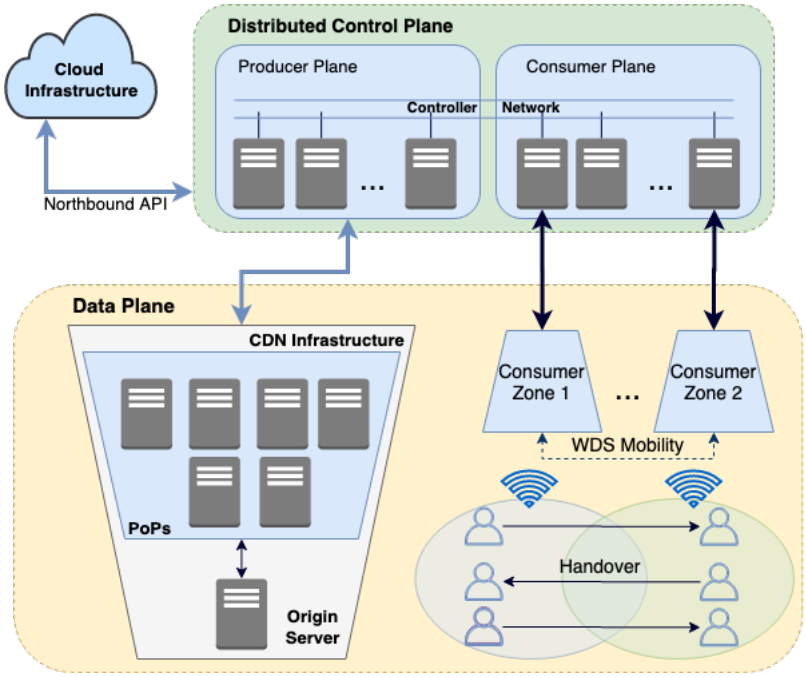


FIGURE 1 A use-case model of CDN implemented over an SD-WAN

Figure 1 depicts a proposed system architecture of a CDN implementation over SD-WAN infrastructure. The Data plane (Edge), constitutes two groups of servers, first, that originates the traffic, such as a Point-of-present (PoP) and a CDN infrastructure and second, that hosts the consumers. Controllers that managers the producer side optimizes the egress traffic, whereas the same for consumer plane optimizes the ingress one. An overlay network logically segregates the producer and consumer plane, maintaining connectivity with respective edge devices. The edge devices must manage any mobility management such as handover. The consumer side data-plane segments its user-base into several zones to facilitate hierarchical routing. Inter-zonal communication takes place via the controller. However, technology like Dynamic Multi-point VPN (DMVPN phase-3) can provide site-to-site on-demand connectivity with summarised routes.

3.2 | System Model

Each controller aggregates partial topology information from downstream edge routes in a link-state manner and generates a topology graph. Each controller shares full topology information to its direct neighbors and summarises any topology information while being a transit; this is a Distance Vector approach. The process limits the size of the all-pair shortest path three by pruning those prefixes which are reachable via a neighbor. Generally, all routing protocols follow a 4 step process in execution. First, neighbor discovery, next Topology synchronization, followed by shortest-path finding, and finally, when the routing table converges, it stays idle in control-plane until a primary route fails and a re-convergence is needed. We assume the controller and edge network topology is unvarying. The following steps describe the process in detail

3.2.1 | Phase 1 : Neighbour Discovery

Controllers create end-to-end tunnels to form adjacencies. Although, in the production network, each edge server registers themselves to a policy server and gets tunnel parameters for establishing a 2-way communication. However, for the sake of simplicity, we opt for simple GRE tunnels between the controllers. In other words, our neighbourship is static, given, it can also

provision dynamic too. Every controller maintains a neighbor-table to keep track of the neighbors' activity. Figure 2 shows a sample topology of a controller network with flow vectors after the neighbor discovery.

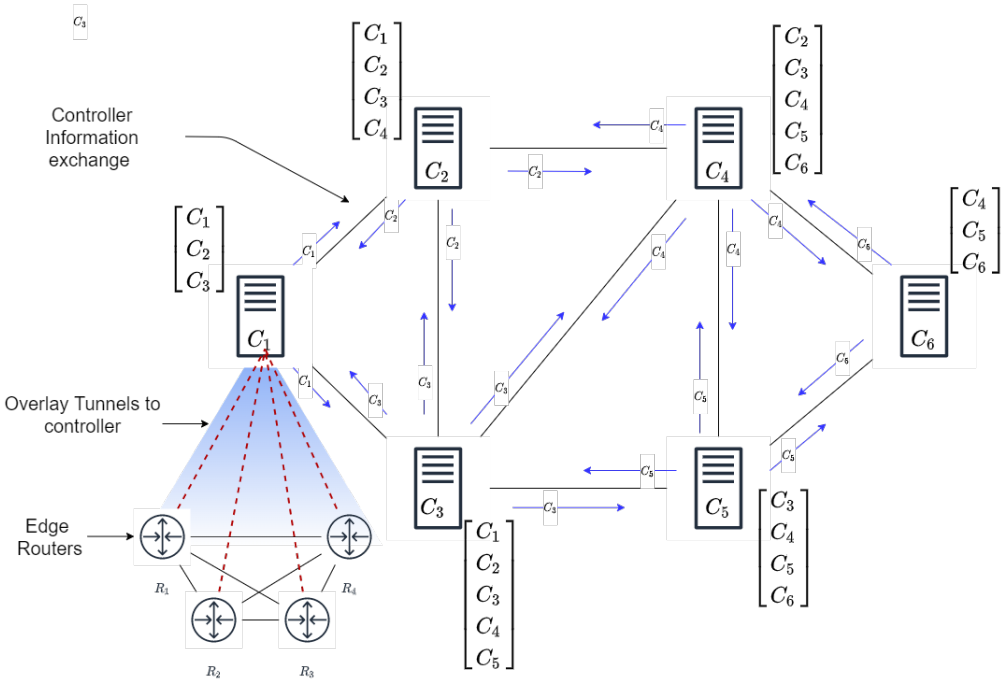


FIGURE 2 Flow vectors after the neighbor discovery

3.2.2 | Phase 2: Initial Controlled Advertisement

In traditional routing protocols like RIPv2[37], OSPFv2[14], and EIGRP[15] that exchanges control packets using multicast and unicast methods, SD-WAN thoroughly depends on overlay networking, where point-to-point or Point-to-multipoint VPN tunnels connects the Edge devices to the controllers. Each controller CON_i generates a topology $G_i(V_i, E_i)$ for its underlying edge network, where V_i E_i are the edge routers and links set respectively. CON_i computes the topology matrix $C_i = [V_i^2] \ni \{c_{i,j} | i, j \in V_i\}$, member of which are set of possible costs between a pair of nodes. For static neighbourship, the neighbour database populates entries with indefinite aging time, while adding neighbours. Whereas, for dynamic neighbour discovery, a pair of neighbours connects on demand and removes any tunnel that ages above a limit (typically 2 hours). any CON_i only advertises its full C_i to its direct neighbours, $\forall CON_j \in \mathcal{N}(CON_i)$ only. When a controller acts as a transit node i.e. forwarding its neighbours cost matrix to another neighbour, it only sends list of reachable networks prefix.

3.2.3 | Phase 3: Vertex set augmentation

After the neighbor discovery and initial advertisement, all controllers become aware of their neighbors' topology and networks accessible by non-neighbor controllers. The Vertex-Set Augmentation (VSA) process augments the producer side, that comprises of origin servers from the consumer side, that hosts end-users. The segregation enables easy policy maintenance, especially for QoS and PBR (Policy-Based Routing). Next, each controller also dynamically changes routes between networks based on the load profile of intermediate routers. A load profile of an edge-router $Load(R_i)$, measures its instantaneous processing load (CPU and Memory) and communication load (Bandwidth Utilization and Congestion) parameters. A router which is heavily loaded, maintains a longer service queue, this results delay in packet processing. In this phase the controller ranks routers based on their load. We reuse the load calculation model from STEN[33], one of our former work.

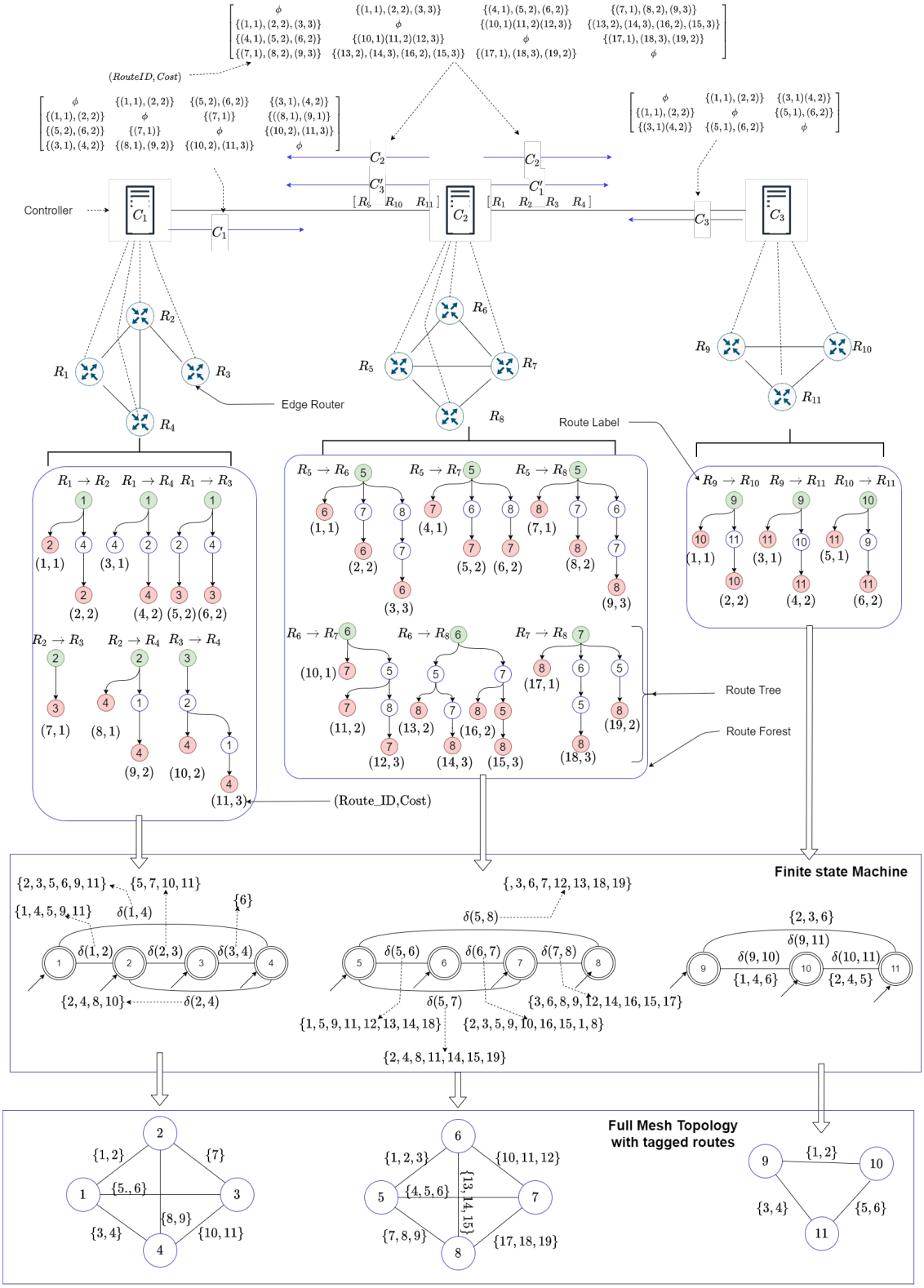


FIGURE 3 Complete process of computing all-paths for all-pair of nodes. First *MRoute* generates *Route Trees* $T_{s,d}$ for all pair of vertices that results route forest RF_i for every controller C_i. Next, FSM compresses RF_i preserving the path information using *RouteID* and Finally Full mesh graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is generated that maps *RouteID*s into Edge-set $\mathcal{E}$.

3.2.4 | Phase 4: Route Calculation

A controller computes a full-mesh graph $\mathcal{G}(V, \mathcal{E})$ from the topology underlying networks topology $G(V, E)$ to maintain the database of all possible paths between all-pair of nodes. It tags the paths with an unique *RouteID*. The process starts with a controller C_i computing a *Route Forest* RF_i, that comprises of several *Route Tree* $T_{s,d}$ which constitutes all possible paths between $v_s, v_d \in V$. RF_i is a ternary tree data structure, thus having a high space complexity. Moreover, we assume the underlying network topology is not dynamic, thus once computed, RF_i serves no purpose of maintaining as no update is expected. An efficient way to preserve RF_i is by turning it into a Finite State Machine (FSM). *RouterID* all paths of all trees in RF_i. Each ID associates a binary cost vector $VCB \in \{0, 1\}^{|E|}$, where $VCB[i] = 1$ if the path in $T_{s,d}$ corresponding to the ID covers the edge $e_i \in E$. The record (*routeID*,VCB) represents each possible path uniquely, as VCB holds the relevant edges, it is used to compute instantaneous cost when any metric parameters (Bandwidth, Delay, CPU load, etc.) changes.

The FSM is $|V|$ states and $|E|$ bidirectional transition functions $\delta(i, j)$ and the $RouteID$ is used as input symbols. All states in the FSM is set as final and initial, this allows to start transition from any arbitrary state. With a given *RouteID* and an initial state, the complete path can be realised by recursive transition on the SMF, keeping the ID same at each iteration.

Finally, the full-mesh graph $\mathcal{G}(V, \mathcal{E})$ is generated by aggregating all *RouteID*s of $T_{s,d}$ and mapped as $e_{s,d} \in \mathcal{E}$. Thus, The FSM stores the node-sequence for every path and G stores *Route ID* mapping. This brings the storage complexity to $O(|V|^2)$ as both the FSM and $\mathcal{G}$ are graphs of $|V|$ nodes, and it's trivial to represent graphs in their matrix form with $O(n^2)$ space. These matrices are exchanged during database synchronisation between controllers, immediate neighbours exchange full matrix so each controller can aggregate its topology to its neighbours. However while being as a transit controller, it suppresses majority of details and only advertises Router IDs learnt from a remote controller. The primary reason being the efficient space management.
Figure 3 depicts the complete process of controllers generating full mesh graph from their underlying topology.

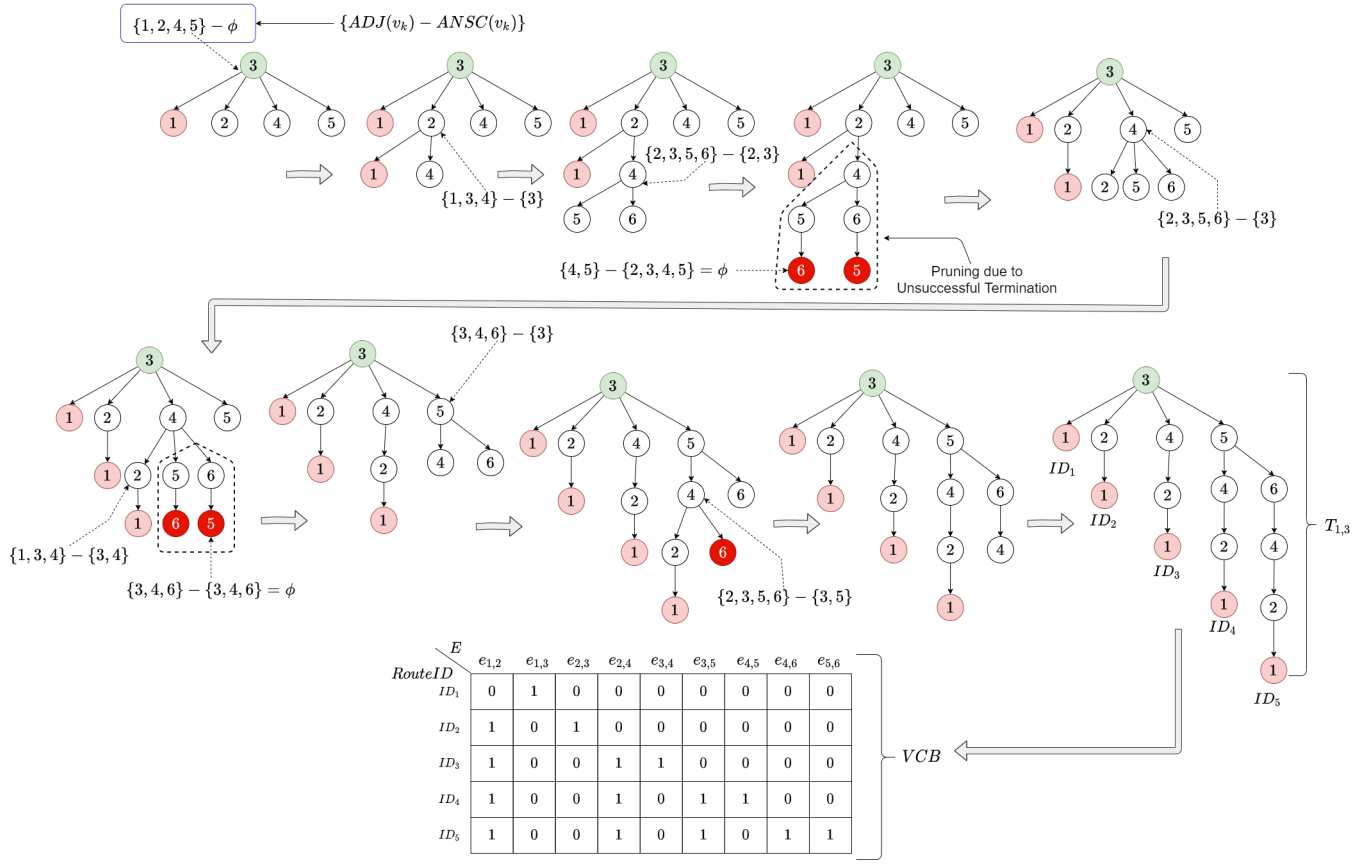
4 | COMPUTING ALL-POSSIBLE PATHS

4.1 | The MRoute Algorithm

Controllers run *MRoute* (Algorithm 1) for their underlying network topology to compute all-possible paths between all-pairs of vertices. The algorithm takes a graph $G(V, E)$,a pair of vertices $v_s, v_d \in V$ and returns a *Route-Tree* $T_{s,d}$. Every root-to-leaf traversal of $T_{s,d}$ is a possible path between v_s and v_d. Figure 4 depicts the generation of *Route-Tree* $T_{1,3}$ with reference to the topology shown in Figure 2.

Figure 4 shows route tree corresponding to $R_{1,3}$. The algorithm uses backtracking principle to enumerate all possible routes between source and destination vertices, in this context, (v_1, v_3).The following explains the working principle of *MRoute*.

- **Initialization**: The process initiates by crating an n-ary tree data structure, keeping v_d as root
- **Recursion**: Every intermediate node $v_k \notin \{v_s, v_d\}$ adds its adjacent nodes $ADJ(v_k)$ as its children, if they are not in its ancestors $ANSC(v_k)$ i.e. $Children(v_k) = ADJ(v_k) - ANSC(v_k)$. In this case,if $Children(v_k) \neq \phi$ the tree grows, thus we refer it as *The Growth-phase*.
- **Termination**: The recursion terminates if any of the following condition hold.
  1. Added child is the source vertex, This case we refer to as *Successful Termination*
  2. All adjacent vertices appear as ancestor i.e. $children(v_k) = \phi$, this case we refer to as *Unsuccessful Termination*
- **Optimisation**: All decedents leading to a leaf that it not the source vertex, is pruned to optimise the space of the tree. This is referred to as *The Shrink-Phase*



RouteID	$e_{1,2}$	$e_{1,3}$	$e_{2,3}$	$e_{2,4}$	$e_{3,4}$	$e_{3,5}$	$e_{4,5}$	$e_{4,6}$	$e_{5,6}$
ID_1	0	1	0	0	0	0	0	0	0
ID_2	1	0	1	0	0	0	0	0	0
ID_3	1	0	0	1	1	0	0	0	0
ID_4	1	0	0	1	0	1	1	0	0
ID_5	1	0	0	1	0	1	0	1	1

FIGURE 4 RouteTree generated by *MRoute* w.r.t. for $r_{1,3}$

- **Result**: A traversal from all leaves to root, gives a set of all possible path between the source and destination vertices, Equation 1.

$$
\begin{aligned}
ID_1 &: n_1 \rightarrow n_3 \\
ID_2 &: n_1 \rightarrow n_2 \rightarrow n_3 \\
ID_3 &: n_1 \rightarrow n_2 \rightarrow n_4 \rightarrow n_3 \\
ID_4 &: n_1 \rightarrow n_2 \rightarrow n_4 \rightarrow n_5 \rightarrow n_3 \\
ID_5 &: n_1 \rightarrow n_2 \rightarrow n_4 \rightarrow n_6 \rightarrow n_5 \rightarrow n_3
\end{aligned}
\tag{1}
$$

4.1.1 | Route Tree

Let, $T_{s,d}$, is termed as *RouteTree* (RT) be an m-way search tree represents all possible paths between $v_s, v_d \in V$, that must hold the following properties

1. The destination vertex v_d is placed at the root
2. All the leaves are identical i.e. the source vertex v_s
3. Every branch (v_i, v_j)in $T_{s,d}$ has positive weight associated called Normalised Cost $NCOST$, and it is calculated from *Rank* function. The *Rank* value varies over time, depending on the the instantaneous load on the CPU and memory.
4. For any intermediate vertex v_k, the functions $ANSC(V_k)$ and $DESC(V_k)$ returns the respective ancestors and descendants. The condition $ANSC(V_k) \cap DESC(V_k) = \phi$ must satisfy during each recursion for preventing routing loops. The recursion terminates when all adjacent vertices of v_k move to the $ANSC(v_k)$.

Algorithm 1 MRoute

Purpose: Finds all possible paths between $(v_s, v_d) \in V^2$
Local Input: $v_k, v_s, v_d \in V$
Global Input: ADJ : The adjacency matrix, $NCOST$: Normalised cost matrix
Output: $T_{s,d}$: Route-Tree for v_s, v_d
Data Structure: n-ary tree
Implementation: Dynamic array, Implicit Stack
Strategy: Recursive, Backtracking

if root $= \phi$ **then**
    root $\leftarrow v_k$ // Initialization
if $v_k = v_i$ **then**
    Return ST // Successful termination
// Unvisited children
$C_k \leftarrow \{ADJ(v_k) - ANS(v_k)\}$ // Non-Preceding Successors
if $C_k = \phi$ **then** // Unsuccessful Termination
    Return UT
for $v_i \in C_k$ **do**
    Update_Ancestors()
    $MRoute(v_i, v_s, v_d)$ // Recurence

4.2 | Cost Calculation and Ranking

MRoute returns a set of paths between v_s, v_d, which is then ordered by ranking the paths. The rank of the a path is determined by its cumulative link load (L) that uses CPU and Memory utilization as parameter, and node cost (C) that is determined by the Bandwidth and Delay . Equation 2 formally presents the *Rank* function.

$$Rank(p_k \in r_{s,d}) = w_n \sum_{i \in n(p_k)} C_i + w_e \sum_{(i,j) \in e(p_k)} L_{(i,j)}$$

Such That, $r_{s,d}$: All paths between v_s and v_d , from *MRoute*
$n(p_k)$: set of nodes at path p_k
$e(p_k)$: set of edges ar path p_k
$C_i = w_c UC_i + w_m UM_i$: cost of node $v_i \in n(p_k)$
$L_{i,j} = w_d DLY_{i,j} / w_b BW_{i,j}$: Load of the link $e_{i,j} \in e(p_k)$
UC_i, UM_i : CPU and Memory utilization of $v_i \in P_k$
BW_i, DLY_i : Bandwidth and Delay $e_{i,j} \in P_k$
$w_n, w_e, w_c, w_m, w_d, w_b$: weighing parameters
$w_n + w_e = 0.5$
$w_c + w_m = 0.5$
$w_b + w_d = 0.5$ (2)

Node costs i.e. CPU, memory utilization and Link load i.e. Bandwidth, delay varies over time which alters the rank accordingly. Route with lowest rank offers a path having higher cumulative bandwidth and lower delay with less utilised nodes. The route selection is not only depending on the link's condition but also the load of the underlying controller, also a heavily loaded controller is given less priority despite being in a relatively light traffic route over a lightly loaded controller hence providing Flexible load balancing.

4.3 | Complexity Analysis

Lemma 1. MRoute is Deterministic and Loop-Free

Proof. The proof is two part, we will first show that the algorithm is loop-free, which will lead us to prove it is deterministic. Also, properties mentioned in Section III(A.5) is referred in this proof.
MRoute selects Children C_k of an non-leaf vertex v_k by filtering them with $ADJ(v_k) - ANSC(v_k)$. Therefore, any internal vertex v_i if visited by a branch, can't be a part of the descendant. Hence it satisfies property 4, $ANSC(v_k) \cap DESC(V_k)$.
Since, the algorithm is loop free, thus maximum depth the tree can recur is the diameter (d) of $G(V, E)$. Since $1 \leq d \leq |V|$, the recursive process has a deterministic termination. □

Lemma 2. MRoute is NP-hard and Traceable

Proof. We first prove the recurrence relation corresponding to the algorithm gives an exponential class, then reduce it into Satisfiability problem to prove it is NP-hard and Traceable.

Let us assume the average branching factor for $T_{s,d}$ be $\bar{b}$ which equal to the mean degree of $G(V, E)$. The algorithm takes $O(1)$ time to fetch $ADJ(v_k)$ and $O(log_b|V|)$ for $ANSC(V_k)$. With *Memoization*, these calls can be made fixed through the run-time. Recursion is then invoked as many as $\bar{b}$ therefore,

$$T(n) = \begin{cases} 0 & \text{if } n = 1, \\ 1 & \text{if } n = 2 \\ \bar{b}T(n-1) + log_{\bar{b}}|V| & \text{otherwise} \end{cases} \tag{3}$$

Using Master theorem[38], it can be shown that $T(n) = O(\bar{b}^n log_{\bar{b}}|V|)$.
To prove the reduction, we'll use an intuitive approach. Since $|E|$ is finite, and G is connected, there exists a path $Path(i, j)$ between all pair of vertices v_i, v_j. Therefore a path $Path(i, j) = \{e \in E\} \subseteq 2^E$.Every path can be encoded into a binary string of length $|E|$, setting 1s to all member edges and 0s otherwise. Hence, it reduces to an $n - SAT$ problem where $n = |E|$. Therefore MRoute is NP-Hard.
Finally, Lemma 1 also proves the algorithm is deterministic, hence it is traceable.

□

5 | SYNCHRONIZATION OF TOPOLOGY BETWEEN CONTROLLERS

The previous section discusses the working principal of *MRoute* that generates all possible pats between all pair of vertices. The source and destination vertices belong to the producer and consumer class respectively. This calculation is done by an SDN controller for its local network. Controllers, then share this information to it's neighbours. In a large distributed SDN network, this process may cause an overflow of controllers' memory. However, routes learned from remote controllers is only significant to routers with close proximity. Therefore, we propose that the advertisement of locally learned routes, is restricted to the neighbouring controllers, and only the best path is advertised further. This results a limited flooding in the controller-network and prevents overflow of controllers' routing table.

Figure 5 depicts the controlled advertising of local routes across a distributed SDN. The user belongs to a consumer network controller by C_1, thus the full routing information is only advertised to C_1's neighbours i.e. C_2, C_3. based on proximity, the SDN is therefore partitioned into adjacent Areas. In General, $Area_k$ advertises full routing information with $Area_{k+1}$, $Area_{k-1}$ and Advertises only the best route learned from $Area_{k-1}$ to $Area_{k+1}$ and vice-versa.

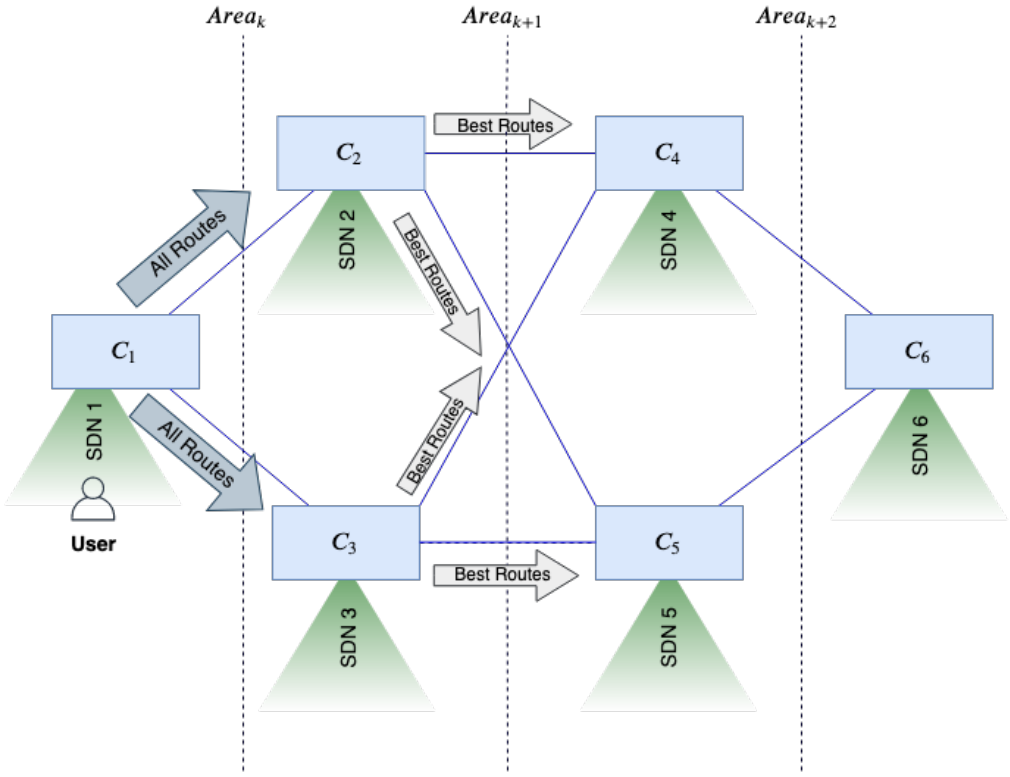


FIGURE 5 Controller network of distributed SDN

6 | IMPLEMENTATION AND RESULTS

This section discusses the test-bed implementation followed by the methodology, used in for the experiments. A detailed description which may be used for reconstruction of the test bed is made publicly available to Github, *ShellMon Sock*[39] is the the monitoring agents that fetches realtime network parameters, *MRRF*[40] runs *MRoute* and *SO-KDN*[41] is the communication protocol between the simulator and the custom management script.

6.1 | Benchmarking

Figure 7 depicts the test-bed architecture and workflow. various open-source tools are used to develop the test-bed, Table 1 lists them with their purpose and brief description of usage. The workflow of the test-bed is as follows.

The test-bed runs a Python script that uses Mininet API to interact and build topology in the Mininet-Server. A series of test-cases of four topology configurations (i.e. Linear, Regular, Tree and Mesh) with increasing number of nodes $[0 - 100]$ is fed into the emulator. Mininet talks a controller-cluster with Openflow, and the cotrollers discover their downstream topology and feed back openflow rules to the respective switches. Openflow rules are generated by translating the routes calculated by the routing engine. The script then stat disconnecting random links $[0 - 10000]$, from the topology which invokes network re-convergence. Eventually, Switches contacts their upstream controller for a new rule. The controller contacts the routing-engine for a new route. In case of the proposed model, routes are pre-computed and ranked. This diminishes the the need of entering into the convergence process, rather it gives the next best route on demand. The rapid-convergence feature of *MRoute* give it an edge over its competitors. During the process a number of parameters (listed in the next section) are collected which are further used for comparison and benchmarking.

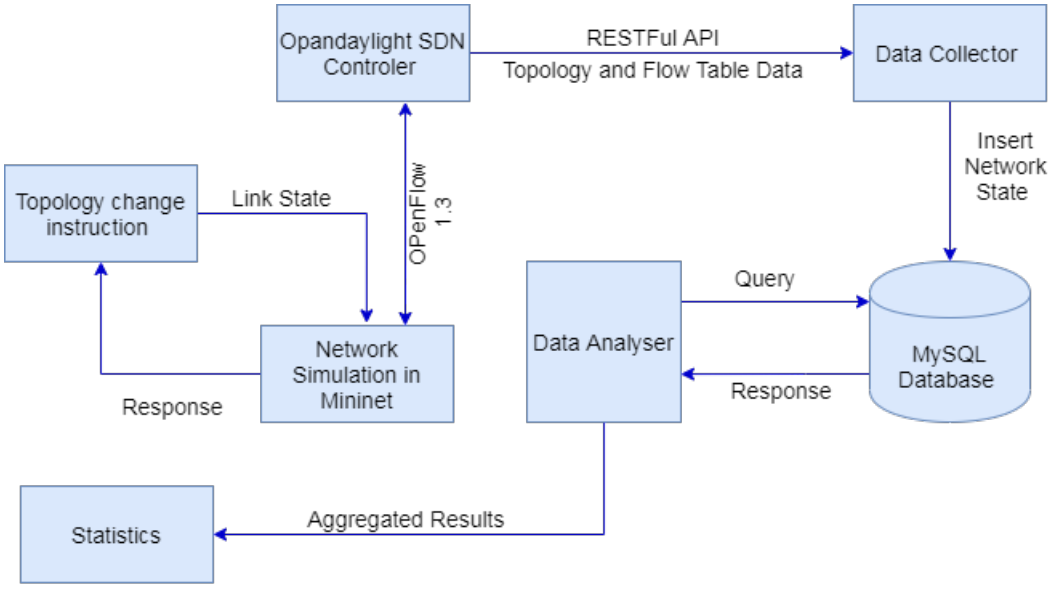


FIGURE 6 Workflow of the proposed test-bed

Tool	Purpose	Description
Mininet	Open-source SDN simulator	Simulates SDN Python API to automate network creation node and link state manipulation
Opendaylight	Opensource SDN Controller	Interfaces with SDN network, simulated with Mininet with Openflow 1.3 Provides topology and flow table information using RESTful API
MySQL	Opensource Database	Stores Node and link states varying over time
Python	Main Programming Language	Automation of SDN Implementation of MRoute Algorithm and various interfaces with ODL and MySQL server

TABLE 1 Lists of open-source tools used to develop the test-bed

6.2 | Comparative Parameters

The experiment aims to compare *MRoute* against OSPF and EIGRP considering their wide acceptance in the enterprise networks with their respective classes classes i.e. Link-state and Advanced Distance Vector routing. The experiment simulates OSPF and EIGRP at the control plane by using their underlying algorithms i.e. SPF and DUAL respectively and comparing with the proposed algorithm. The comparison benchmarks *MRoute* in six parameters namely,

1. **Discovery Time**: How long on average the algorithm takes to calculate all-routes for all-pairs. Analytically *MRoute* is an NP-Hard problem, therefore the time complexity is exponential, however for SPF and DUAL its $O(n^2)$.
2. **Convergence Time**: How long on average the algorithm take to calculate an alternative path if the primary path fails. Since, *MRoute* proactively pre-calculates all possible routes and maintain their dynamic rank, it's always guaranteed that until there is at least one valid route, the controller will reinforce it to the network instantly. As a result the network will converge in a constant order time.
3. **Communication cost for Discovery**: Routing protocols use distributed computing models, to discover and monitor neighbours they use "Hello" protocol over Multicast. The number of control packets in *MRoute* is always constant as all Edge device sends their local information to the controller using a tunnel, therefore it is independent of the network diameter
4. **Communication cost for Convergence**: *MRoute* is free from re-routing, as any time a re-routing request comes, the controller returns the next best active route. Thus, no control messaging is needed.
5. **Space Consumption**: The amount of memory needed to maintain the topology information including all the data structures and look-up tables.

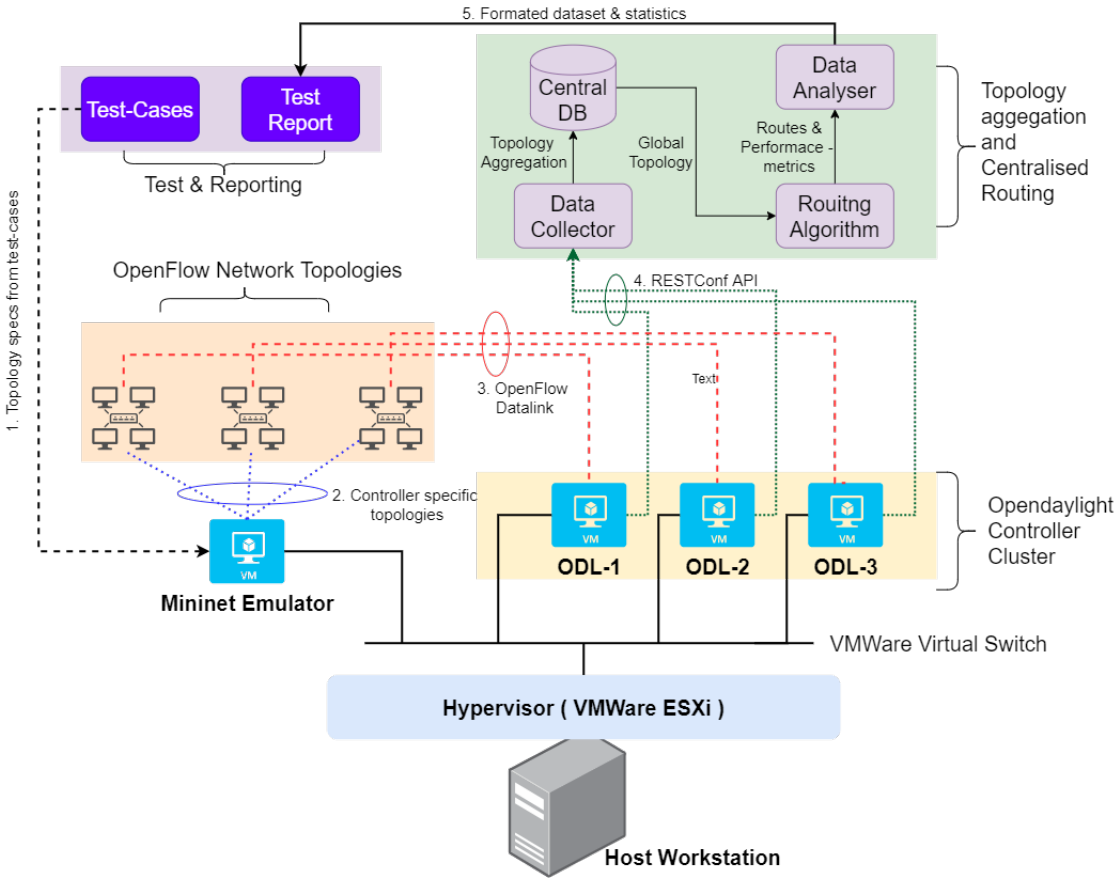


FIGURE 7 Deployment diagram of the experiment

6. **Route Tree size**: The algorithm is inherently exponential yet deterministic. During the grow phase, the tree adds children and removes the invalid paths during the shrink phase. The growth of tree size is also tested to examine the temporal space complexity of the n-ary tree data structure.

6.3 | Experiment Setup

Figure 7 depicts the detailed setup of experiment and implementation of the test-bed, the workflow is denoted by the numbered events. The test-bed comprises of a number of virtual machines designated for the Minitet and OpenDaylight instances, they share a common network segment provided by the hypervisor. A set of network topology configurations is listed in the test-case database which are pushed into the Mininet instance. One Iteration comprises of five phases that terminates with a report summarising parameters, listed in the previous section. Each topology configuration creates 3 OpenFlow LAN networks, each represents an edge segment and designated to a specific controller from the Controller-Cluster. Opendaylight (ODL) controllers listen to their respective TCP port 6633 with which the Open V-Switches (OVS) corresponding to their downstream topologies establish an OpenFlow data-link. Each controller maintains its downstream topology map and flow-tables and exposes them using RESTConf API to the northbound using TCP port 8181. Topologies Flow-tables from individual Controller is accumulated by the Data-Collector module in the application plane which is then fused into a global topology (as described in Figure 3). The routing algorithm module executes SPF, DUAL and MRoute on the topology and returns benchmark information to the Data-Analyser, which finally formats the comparison information in a CSV file and streames it to the Reporting module. For the test purpose and due to recourse constrain we limit the benchmarking with 3-Controller (each with 4-vCPUs & 4GB RAM) configuration; however, the same process is scalable to a larger configuration with adequate resource given. A clarification for the readers' comprehension on Controller-Cluster, The cluster configuration does not yield a controller aggregation (e.g. Akka clustering) rather a collection of multiple autonomous controllers.

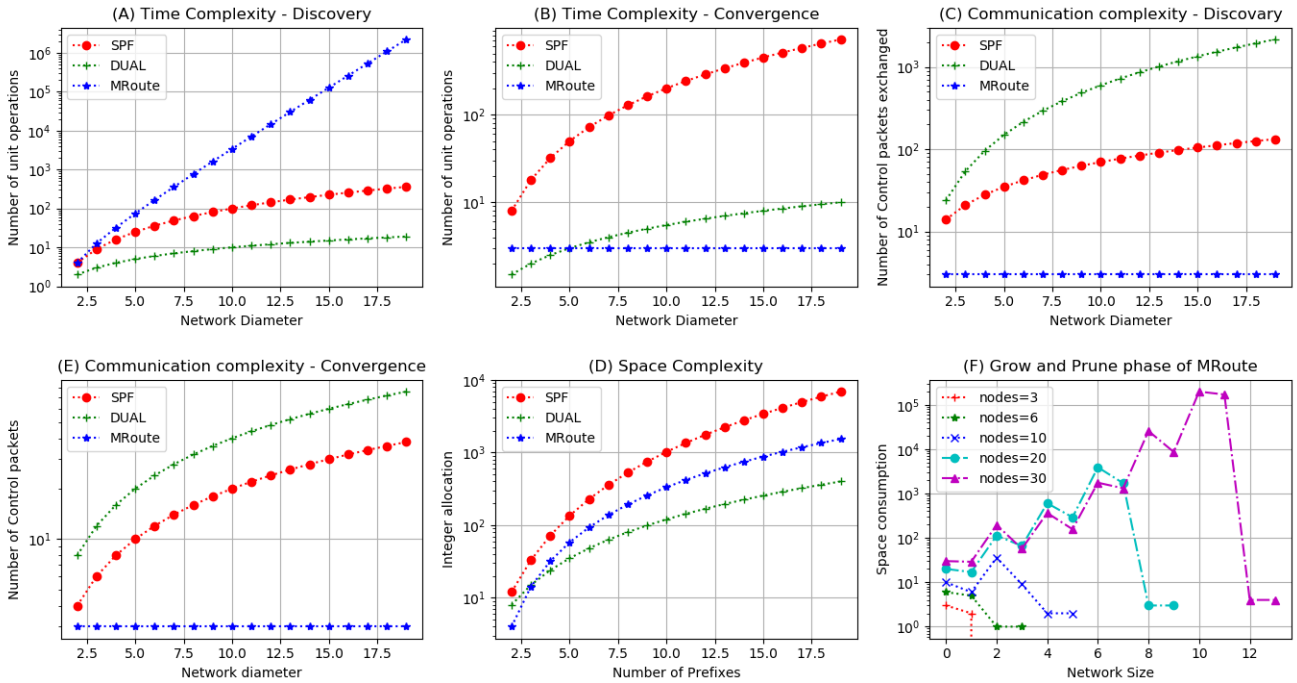


FIGURE 8 Experimental Results and Comparison *MRoute* against SPF and DUAL using the following parameters (A)Time Consumption to computing paths, (B) Time consumption to converge, (C) Control traffic for topology synchronization, (D) Space consumption for topology maintenance (E) Control traffic for convergence, (F) Route-Tree size .

6.4 | Analysis of Results

The comparative analysis between *MRoute*, DUAL and SPF (Figure 8) benchmarks algorithms using six parameters as discussed in Section 6.2., In this section we present a comprehensive explanation to the results. Subplot 8(A) compares the time complexity with respect to the size of the network, outcomes are plotted in log-scale therefore *MRoute* shows an exponential growth, as shown in Lemma 2; in comparison, DUAL and SPF which are bounded above by $O(n^2)$. Due to the Diffusion-Computation model and the presence of Feasible-Successor, DUAL goes less deep into the convergence state than of SPF. We tuned the SPF to run on each down-stream topology in parallel, simulating a multi-area OSPF network. Although, it seems initially that DUAL is the optimum than its competitors, however the way these algorithms work in SD-WAN, flips the perception. *MRoute* calculates all possible paths in advance, therefore in the long run if the topology remains unaltered, it would never enter a re-convergence process, which is not the case of the rest two. This situation is shown in the subplot 8(B), where the random link failure scenario (Section 6.1) causes SPF to re-converge every time, DUAL shows a better result as in some of the cases feasible-successor exists or a neighbour replies with route much before the query reaches the network boundary. However, *MRoute* shows a constant reading here as it is a $O(1)$ task that requires a fixed number of operation involving querying and getting reply for the next best route. The process can be thought as a generalised case of DUAL where all backup routes are ranked and listed.

The communication complexity measures the number of packets exchanged between the nodes while discovering or converging into the network. In case of SPF and DUAL, the algorithms are inherently distributed, therefore the local routes are advertised, Queried during re-convergence and polled for their liveliness using Reliable Updates and Hello protocols respectively. Since OSPF uses Link-state model, the total packet exchanged is higher than that of Distance-vector based EIGRP. *MRoute* is designed as a centralised routing algorithm, therefore it does not exchange any discovery or update messages with other nodes, rather it updates only the controller which is logically one hop away, This justifies the subplot 8(C,E).

The State-Model representation of the Route-forest reduces the space consumption of *MRoute* drastically by tagging routes as a fixed length binary vector of edges with *RouteID*. However, while generating the *Route-Tree*, it consumes memory in an exponential rate, although the pruning phase releases some memory but the overall growth remains exponential. It is only after the complete forest is generated the state model gets built which compresses them into tables and relinquishes the memory

(subplot 8(F)). Space complexity of *MRoute* sits between SPF and DUAL as OSPF maintains identical link-state database for all nodes and EIGRP topology tables lists the Successor and Feasible successors for each destination prefix depicted in subplot 8(F).

7 | CONCLUSION

This paper proposes a routing algorithm *MRoute* for constrained SD-WAN, that converges in constant time. Also, it proposes an advertisement model for inter-controller synchronization of locally routes . Using Branch-and-Bound scheme, *MRoute* computes all possible paths between all pair of nodes, ranks them and feeds back with the optimal one on demand. Further, a finite-state model is also proposed to reduce the space complexity and routing table formation. A multi-objective comparison with SPF and DUAL algorithm is also given to verify the acclaimed mathematical models.

An extension to this algorithm, with self-learning capability using Deep Reinforcement Learning, for determining reliability of routes is under development.

8 | ACKNOWLEDGMENT

The present work was undertaken in the context of the "Self-Organization Toward Reduced Cost and Energy Per Bit for Future Emerging Radio Technologies" with contract number 734545.The project has received research funding from the H2020-MSCA-RISE-2016 European Framework Program.

References

1. Butler B. Business Value of Cisco SD-WAN Solutions : Studying the Results of Deployed Organizations. 2019: 1–13.
2. Mijumbi R, Serrat J, Gorricho J, Latre S, Charalambides M, Lopez D. Management and orchestration challenges in network functions virtualization. *IEEE Communications Magazine* 2016; 54(1): 98-105. doi: 10.1109/MCOM.2016.7378433
3. Li Ling , Ma Xiaozhen , Huang Yulan . CDN cloud: A novel scheme for combining CDN and cloud computing. In: . 01. ; 2013: 687-690
4. Taleb T, Samdanis K, Mada B, Flinck H, Dutta S, Sabella D. On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration. *IEEE Communications Surveys Tutorials* 2017; 19(3): 1657-1681. doi: 10.1109/COMST.2017.2705720
5. Thyagaturu AS, Mercian A, McGarry MP, Reisslein M, Kellerer W. Software Defined Optical Networks (SDONs): A Comprehensive Survey. *IEEE Communications Surveys Tutorials* 2016; 18(4): 2738-2786. doi: 10.1109/COMST.2016.2586999
6. Mijumbi R, Serrat J, Gorricho J, Bouten N, De Turck F, Boutaba R. Network Function Virtualization: State-of-the-Art and Research Challenges. *IEEE Communications Surveys Tutorials* 2016; 18(1): 236-262. doi: 10.1109/COMST.2015.2477041
7. Sarrigiannis I, Ramantas K, Kartsakli E, Mekikis P, Antonopoulos A, Verikoukis C. Online VNF Lifecycle Management in a MEC-enabled 5G IoT Architecture. *IEEE Internet of Things Journal* 2019: 1-1. doi: 10.1109/JIOT.2019.2944695
8. ONF*OpenFlow-enabled SDN and Network Functions Virtualization*. 2014. https://www.opennetworking.org/wp-content/uploads/2013/05/sb-sdn-nvf-solution.pdf.
9. Fojta T. VMware vCloud ® Architecture Toolkit™ for Service Providers Architecting a VMware vCloud Director ® Solution for VMware Cloud Providers™. 2018(January). http://www.vmware.com/download/patents.html.
10. *Cisco SD-WAN Getting Started Guide*. Cisco Systems . 2019. https://www.cisco.com/c/en/us/td/docs/routers/sdwan/configuration/sdwan-xe-gs-book.pdf.
11. Citrix Product Documentation . *Citrix SD-WAN 10.2*. https://docs.citrix.com/en-us/citrix-sd-wan/10-2/citrix-sd-wan-10.2.pdf.
12. Cisco Systems I. Unicast Overlay Routing Overview. tech. rep., https://sdwan-docs.cisco.com/Product_Documentation/Software_Features/Release_18.2/03Routing/01Unicast_Overlay_Routing.
13. Citrix Product Documentation . Adaptive transport. 2017. https://docs.citrix.com/en-us/citrix-virtual-apps-desktops/technical-overview/hdx/adaptive-transport.html.
14. Moy J. OSPF Version 2. RFC 2328; 1998. https://rfc-editor.org/rfc/rfc2328.txt
15. Savage D, Ng J, Moore S, Slice D, Paluch P, White R. Cisco's Enhanced Interior Gateway Routing Protocol (EIGRP). RFC 7868; 2016. https://rfc-editor.org/rfc/rfc7868.txt
16. Jain A, Sadagopan N S , Lohani SK, Vutukuru M. A comparison of SDN and NFV for re-designing the LTE Packet Core. In: ; 2016: 74-80
17. Nikbazm R, Dashtbani M, Ahmadi M. Enabling SDN on a special deployment of OpenStack. In: ; 2015: 337-342
18. Zhang H, Yan J. Performance of SDN Routing in Comparison with Legacy Routing Protocols. In: ; 2015: 491-494
19. Zhang T, Bianco A, Giaccone P. The role of inter-controller traffic in SDN controllers placement. In: ; 2016: 87-92
20. Singh P, Manickam S. Design and deployment of OpenStack-SDN based test-bed for EDoS. In: ; 2015: 1-5
21. Nokia: Nokia CloudBand VMware vCloud NFV Application Note. https://onestore.nokia.com/asset/201686.
22. Nokia: Nokia CBIS Differentiation Case Study. https://onestore.nokia.com/asset/200841.

23. Welcome to Open Source MANO's documentation! — Open Source MANO 6.0 documentation. .

24. *Mininet Walkthrough*. . http://mininet.org/walkthrough/.

25. Karakus M, Durresi A. A Scalable Inter-AS QoS Routing Architecture in Software Defined Network (SDN). In: ; 2015: 148-154

26. Egilmez HE, Tekalp AM. Distributed QoS Architectures for Multimedia Streaming Over Software Defined Networks. *IEEE Transactions on Multimedia* 2014; 16(6): 1597-1609. doi: 10.1109/TMM.2014.2325791

27. Kaiming Liu , Yahui Cao , Yuanan Liu , Gang Xie , Chao Wu . A novel min-cost Qos routing algorithm for SDN-based wireless mesh network. In: ; 2016: 1998-2003

28. Qin F, Zhao Z, Zhang H. Optimizing routing and server selection in intelligent SDN-based CDN. In: ; 2016: 1-5

29. Lan H, Liu L, Yu X, Gu H, Guo Y. A novel multi-controller flow schedule scheme for Fat-tree architecture. In: ; 2016: 1-3

30. Hata M, Izumi S, Abe T, Suganuma T. A proposal of SDN based mobility management in multiple domain networks. In: ; 2016: 1-4

31. Ugwuanyi EE, Ghosh S, Iqbal M, Dagiuklas T. Reliable Resource Provisioning Using Bankers' Deadlock Avoidance Algorithm in MEC for Industrial IoT. *IEEE Access* 2018; 6: 43327-43335. doi: 10.1109/ACCESS.2018.2857726

32. Ugwuanyi EE, Ghosh S, Iqbal M, Dagiuklas T, Mumtaz S, Al-Dulaimi A. Co-Operative and Hybrid Replacement Caching for Multi-Access Mobile Edge Computing. In: ; 2019: 394-399

33. Ghosh S, Dagiuklas T, Iqbal M. Energy-Aware IP Routing Over SDN. In: ; 2018: 1-7

34. Ghosh S, Busari SA, Dagiuklas T, et al. SDN-Sim: Integrating a System-Level Simulator with a Software Defined Network. *IEEE Communications Standards Magazine* 2020; 4(1): 18-25. doi: 10.1109/MCOMSTD.001.1900035

35. Qian Y, Liu Y, Kong L, Wu M, Mumtaz S. ReFeR: Resource Critical Flow Monitoring in Software-Defined Networks. In: ; 2018: 1-7

36. Zhang L, Wu J, Mumtaz S, Li J, Gacanin H, Rodrigues JJPC. Edge-to-Edge Cooperative Artificial Intelligence in Smart Cities with On-Demand Learning Offloading. In: ; 2019: 1-6

37. Schulzrinne H, Conroy LW, Lu HL, et al. Toward the PSTN/Internet Inter-Networking–Pre-PINT Implementations. RFC 2458; 1998. https://rfc-editor.org/rfc/rfc2458.txt

38. Yap C. A Real Elementary Approach to the Master Recurrence and Generalizations. In: Ogihara M, Tarui J., eds. *Theory and Applications of Models of Computation*Springer Berlin Heidelberg; 2011; Berlin, Heidelberg: 14–26

39. Monitoring agent over socket based communication. https://github.com/rishiCSE17/ShellMon_sock.

40. Most Reliable Route First. https://github.com/rishiCSE17/MRRF/tree/master/Scripts.

41. Self Organised Knowledge Defined N etwrok. https://github.com/rishiCSE17/SO-KDN.