A Cognitive Routing Framework for Reliable Communication in IoT for Industry 5.0

Saptarshi Ghosh, *Student Member, IEEE*, Tasos Dagiuklas, *Senior Member, IEEE*, Muddesar Iqbal, *Member, IEEE*, and Xinheng Wang, *Senior Member, IEEE*

***Abstract*—Industry 5.0 requires intelligent self-organized, self-managed, and self-monitoring applications with ability to analyze and predict the human as well as machine behaviors across interconnected devices. Tackling dynamic network behavior is a unique challenge for Internet of Things applications in Industry 5.0. Knowledge-defined networks (KDN) bridge this gap by extending software-defined networking architecture with knowledge plane, which learns the network dynamics to avoid suboptimal decisions. Cognitive routing leverages the sixth-generation (6G) self-organized networks with self-learning feature. This article presents a self-organized cognitive routing framework for a KDN which uses link-reliability as a routing metric. It reduces end-to-end latency by choosing the most reliable path with minimal probability of route-flapping. The proposed framework precalculates all possible paths between every pair of nodes and ensures self-healing with a constant-time convergence. An experimental test-bed has been developed to benchmark the proposed framework against the industry-stranded link-state and distance-vector routing algorithms SPF and DUAL, respectively.**

***Index Terms*—Cognitive routing, Industry-5.0, rapid convergence.**

I. Introduction

IN 2013, the German Academy of Engineering Sciences presented a recommendation and research agenda for Industry 4.0. Its primary motivation was to achieve seamless integration between physical and virtual technologies to facilitate smart manufacturing, which results in significant inflation of the Internet of Things (IoT) technology in industrial automation. Between 2009 and 2019, the industrial sector has contributed 20% to the EU's GDP. Industry 5.0, as a natural successor, aims to build on top of the existing architectural frameworks of industrial and heterogenous IoT (I-IoT, H-IoT) and interoperability between cyber-physical systems. The Directorate-General of Research and Innovation (EU) has identified a new set of concepts that Industry 5.0 addresses. These are human-centric solutions, bio-inspired technologies, real-time digital-twins technology, network analytics, machine-learning (ML)-based automation, and trustworthy autonomy. A large-scale industry needs to have a scalable network fabric to interconnect all its devices. Software-defined networking (SDN) provides such a programmable, vendor-agnostic communication platform. 5G leverages SDN at its core to virtualize network services (NFV), and Internet service providers (ISPs) use it in wide-area networks (WAN) deployment (SD-WAN). SDN provides a bird's eye view of the network where the control plane (CP) accumulates global knowledge about the underlying topology and flows. Additionally, the data plane generates enough which the controller can mine for analytics. In SDN-based routing, the routing protocol uses the global view to calculate optimal paths without letting the routers exchanging control packets. An efficient routing protocol aims to avoid suboptimal paths and converge rapidly in a dynamic environment. However, highly time-critical industrial communication systems, such as IoT infrastructure for manufacturing plants, cannot tolerate delays due to routing protocol convergence. Therefore, routing optimization by analyzing the network's behavior provides a better heuristic, which eventually reduces the convergence probability.

In SDN [1] routing, the shortest-path calculation is the subjected optimization problem, where a controller calculates the optimal values of the free parameters subject to a set of communication constraints defined as a policy (self-optimization). The controller then configures the parameters into the underlying network devices (self-configuration) and serves alternate routes on-demand, if the primary one fails (self-healing), thus supporting the SON [2]. However, the application of ML in route-optimization is a relatively new domain; at the time of writing this article, there are a handful of works done in developing an intelligent routing algorithm for SDN. The base model of fitting ML in SDN is referred to as knowledge-defined networks (KDN) [3], where the primary objective is to accumulate holistic information from a supervising CP of an underlying IP-network analyzes them to extract knowledge that generalizes the network behavior. This knowledge eventually helps to bypass the need

Manuscript received April 28, 2021; revised July 13, 2021 and December 4, 2021; accepted December 24, 2021. Date of publication January 7, 2022; date of current version May 6, 2022. This work was supported in part by H2020-MSCA-RISE-2016 European Framework Program under Grant H2020-MSCARISE-2016-734545. Paper no. TII-21-1638. *(Corresponding author: Muddesar Iqbal.)*

Saptarshi Ghosh, Tasos Dagiuklas, and Muddesar Iqbal are with the Division of Computer Science and Informatics, School of Engineering, London South Bank University, SE1 0AA London, U.K. (e-mail: ghoshs4@lsbu.ac.uk; tdagiuklas@lsbu.ac.uk; m.iqbal@lsbu.ac.uk).

Xinheng Wang is with the Department of Mechatronics and Robotics, School of Advanced Technology, Xi'an Jiaotong-Liverpool University, Suzhou 215000, China (e-mail: xinheng.wang@xjtlu.edu.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TII.2022.3141403.

Digital Object Identifier 10.1109/TII.2022.3141403

for using costly heuristic routing algorithms, having preserved the equal adaptation capabilities to network dynamics [4].

Self-organized networking (SON) [5] in the fifth-generation cellular communication systems (5G) enhances the requirements of its predecessor. Some of the new requirements involve increasing traffic capacity, improving QoS/QoE, support of heterogeneous radio access networks (RAN), 10 Gbps peak data rate, submillisecond latency, support of ultrahigh reliability, improved security, privacy and flexibility, and reduction of CAPEX and OPEX [6]–[8]. SON constitutes the following three entities.

1) Self-optimization provides several CP optimization strategies such as caching, routing, and load-balancing, which are invoked autonomously. Relevant algorithms calculate the optimal values of several decision variables with respect to a set of constrains, called policies.
2) Self-configuration automates the injection of the decision parameters (e.g., operational and radio config) to the underlying data-plane devices.
3) Self-healing provides high-availability to the overall network. A typical model uses detection, diagnostic, and compensation sequence to automate the recovery process.

Recent development in SON shows a significant use of ML to accelerate the performance of its constituents [9].

In this article, we propose most-reliable-route-first (MRRF), an intelligent routing algorithm for self-organised knowledge-defined networks (SO-KDN). The proposed model initially calculates all possible paths for all pairs of nodes from the networks' topology using the proposed algorithm (MRoute) and aims to learn the reliability of individual links by their statistical measures of volatility over time. The algorithm maintains the rank of the routes based on their cumulative reliability and serves them on-demand in constant time, hence assuring the most reliable routes. We further propose a full-fledged implementation of the KDN model as a test-bed to conduct experiments, which benchmarks MRoutes with diffusion update algorithm (DUAL) [10] and shortest path first (SPF) [11] that powers enhanced interior gateway routing protocol (EIGRP) as open shortest path first (OSPF), respectively.

The rest of this article is organized as follows. Section II introduces elementary concepts of 6G, Section III presents the state-of-the-art of intelligent SDN-routing. The system model, including problem formulation, the design, and analysis of the proposed algorithm, is discussed in Section IV. Section V addresses the ML extension and the learning process, the experimental setup, hyperparameter tuning, benchmarking, and the analysis of results are provided in Section VI. Finally, Section VII concludes this article.

II. State-of-the-Art in KDN

The inception of the KDN comes from the work of Clark *et al.* [3], where the authors propose a unified knowledge plane (KP) that takes decisions based on partial and conflicting information, accumulated from a distributed cognitive framework. The article considers the use of KP in solving the optimal route-preference problem by learning network behavior over time. However, the article lacks information to real-world network types such as ISP, Enterprise, Cellular, etc., and does not include working principles in a heterogeneous networks. These issues are addressed by Strassner *et al.* [12] by their extension of KDN with an interface-plane, that offers a much clear view of the implementation and necessary building blocks. Several surveys show the growing application of ML and deep learning (DL) on SDN architectures in recent times, that aims to achieve the KDN. Xie *et al.* [13] in their survey show the growing application of ML and Deep Learning (DL) on SDN architectures in recent times, that aims to achieve the KDN. Fadlullah *et al.* [14] presents a classification of various ML/DL algorithms and their application to intelligent network traffic control systems. Chen *et al.* [15] focuses on the applications of DL into several Cognitive Wireless communication systems such as the Internet of Things (IoT), Mobile Edge Computing (MEC), Unmanned Aerial Vehicle (UAV) network. Zhao *et al.*[16] reviews the specific applications of ML into SDN problems such as defense mechanism from Distributed Denial of Service (DDoS) attacks [17], Anomaly Detection, Traffic Classification, Routing Optimization, etc. To restrict our scope of the discussion, we now put the relevant state of the art focusing on routing optimization only.

Shortest path algorithm (SPA) and heuristic algorithms (HAs) are the two widely used approaches that solve routing-optimization problems [17]. Among several alternatives, artificial neural networks, reinforcement learning (RL), deep RL (DRL), and lazy learning (LL) are the four learning models primarily used to address routing optimization. Yanjun *et al.* [18] propose an ML-Meta later-based approach, where an ML model is trained by the calculated traffic parameters of a HA and its corresponding network state as input. The proposed framework maps the input and output of the HA that reduces its exponential run-time in a constant one. NeoRoute [19] models traffic characteristics by forecasting future link consumption using the recurrent neural network (RNN) with long short–term memory (LSTM). A similar problem is addressed by Álvaro López-Raventós *et al.* [20] for high-density WANs. The aforementioned works use supervised-ML models for training, which assumes the network characteristics are likely to stay identical over time. Therefore, they are not suitable for dynamic networks, which, in contrast, need an online-training model such as RL or DRL. Sendra *et al.* [21] propose a DRL framework that an agent that weighs the delay, loss, and bandwidth for every possible link of a target network. The network feeds either reward or penalty back to the agent based on the change in end-to-end throughput. The agent uses the feedback to tune its decision-making model. Francois and Gelenbe [22] apply DRL with a *RNN* in cognitive-routing in SDN. The proposed architecture shows consistent performance even in a highly chaotic environment. Applications of DRL in SDN-specific problems include QoS-aware adaptive routing [23].

In this article, we use a time-series analysis model that extracts link volatility trends using RNN+LSTM to ensure reliability in constant time.

III. System Model

Programmable networks consist of a topology of configurable routers. Routers connect the a local area network (LAN) to interface with the switching networks and WANs to interconnect with

neighboring routers. In a heterogeneous routing environment, the routers' computational capacity varies significantly. The load on a route processor delays the control-packets processing (service delay or node-cost) as most of the data-plane traffic are switched at the router application-specific integrated circuits. However, existing enterprise routing protocols (e.g., EIGRP, OSPF) do not include service delay as a metric parameter. That said, the link-cost, i.e., the propagation and transmission delay, is influenced by several link parameters such as throughput, latency, load, and reliability. QoS-aware routing protocols use admission control mechanism to allow only the traffics that meet certain constraints specified in the policies. In the proposed system model, routers have a node-cost, and the link-costs are calculated by optimizing the respective objective function subject to a set of link-specific constraints. The proposed routing model uses both node and link costs as metric parameter. This offers a novel routing model, which includes service delays in route calculation also complying with the QoS routing principle. The routing algorithm (MRoute) proactively enumerates all-possible paths between all pairs of nodes. Further the varying normalized link cost is used to calculate the link-specific reliability. The model uses the reliability as a metric for routing and provides routes on demand. As the paths are proactively calculated, the convergence does not need recomputation of the topology, and hence results in constant time convergence.

We choose to model the topology of a SDN as a *Simple, Finite, Connected* graph. The network consists of programmable routers and switches, which are connected to the controller via a secure and reliable south-bound interface (SBI). The controller treats both the router and switch as a generic edge-node (EN) having a well-defined set of communication (L1) and MAC (L2) protocols configured. Additionally, the routing (L3) and transport (L4) protocols must ensure the following properties. ENs do not exchange CP traffic among each other but only with the controller over the SBI. There exists no neighbor discovery mechanism, and ENs share their local information and keep-alive packets with the controller only. Controller accumulates various telemetry information of the ENs such as memory, CPU, and network interface. The network topology does not change frequently.

Each EN EN_i maintains a local routing table (RT_i) that comprises three disjoint sets of entries: the *connected routes* (CR_i) are networks connected directly to the device interfaces, the *static routes* (SR_i) are configured statically on the device, and *remote routes* (RR_i) are not learnt from the controller. By definition, these sets partition the routing table, i.e., $CR_i \cap SR_i \cap RR_i = \phi$ and $CR_i \cup SR_i \cup RR_i = RT_i$. The controller uniquely identifies each EN by their *NodeID* similar to router ID in OSPF and EIGRP and maps it with their corresponding CR set. When an EN receives a packet with unknown destination address, it forwards it to the controller. The controller then resolves the destination node's ID from from map, finds a route between the source and destination router, and replies it back to the source node [24].

Fig. 1 depicts a reference topology of six routers with Node ID R_1–R_6; the corresponding CR_is are further segregated into the LAN (L_i) and WAN (W_i) links ($L_i \cap W_i = \phi$) following

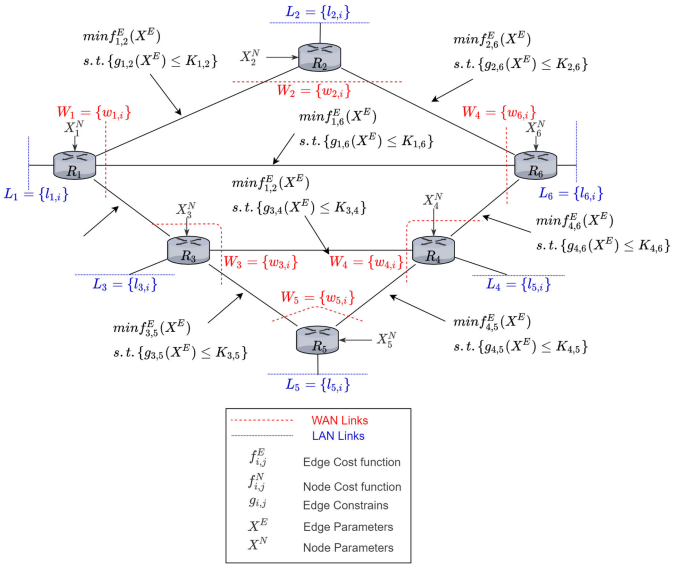

Fig. 1. Reference topology with route-policies.

RFC-1918[]. The controller uses link-state routing approach to build a topology from this information, i.e., nodes having common WAN network are adjacent. However, for the sake of simplicity, we did not include topology with broadcast segments as it requires additional designated node's placement [25]. Hence, we assume all the links are point-to-point in nature.

The network has a set of node-specific parameters (X^N) such as CPU and memory utilization, and a set of edge-specific parameters (X^E) such as bandwidth, delay, load, and reliability. The WAN links are constrained and heterogeneous, i.e., its attributes are bounded above by some predefined values specific to that link. These values are generally dependent on the network policy or the media type, and hence we leave it to be as user-defined. We propose the formulation of link-cost as a set linear programming problems, for individual edges, with a linear cost-function $f^E_{i,j} : X^E \to \mathbb{R}^+$, between R_i and R_j, such that its linear constrains $g_{i,j}(X^E) \le K_{i,j}$ are met. This is to overcome the limitation of OSPF's suboptimal routing issue due to its simplistic metric, and EIGRP's route-flapping problem caused by its dynamic metric parameters. The proposed method uses link attributes defined by RFC-7868 [26]. However, as the metrics are calculated locally to the controller, it diminishes any need of exchanging update-packets between ENs, thus eliminating the cause of route-flapping. Similar to the edge-cost, the node-cost also contributes to the calculation of the final metric. The node-cost function $f^N_i : X^N \to \mathbb{R}^+$ computes a cost based on the node attributes (X^N). In our previous work [27], we have shown the benefits of routing optimization by fusing node and edge costs in metric calculation.

The controller generates a graph structure isomorphic to the network topology, and weighs its edges by relaxing the f^N_i and f^N_j into $f^E_{i,j}$ for all adjacent R_i, R_j stochastic temporal edge normalization (STEN) [27]. As the X^E and X^N vary over time but the topology remains the same, the subjected graph be a dynamic isomorphic graph, which we refer to as *meta-graph*.

The proposed algorithm performs the following steps in order to meet the rapid-convergence criteria. First, it efficiently computes all possible paths between all pairs of nodes from the meta-graph using an algorithm called *MRoute*. This step is invoked whenever the topology changes. Seconds, it computes the reliability of the links by profiling their cost variation over the time using an RNN using LSTM; this is a periodic step. Third, it ranks the computed paths obtained from step 1 based on their cumulative reliability obtained from step 2. This step is invoked every time an update happens. First, returns the most reliable routes on demand as primary route keeping the rest in backup. In case the primary route fails, next best route is served instantly. Hence, the rapid-convergence is achieved.

The following subsections explain the problem formulation in detail.

A. Problem Formulation

The simple, undirected, and connected graph $G(V, E)$ represents the topology of the underlying network, where $V = \{v_i\}$ and $E = \{e_{i,j} | adj(v_i, v_j)\}$ are the vertex and edge set, respectively. V and E are finite and nonempty, $adj(v_i, v_j) = 1$ if v_1, v_j are adjacent, and 0 otherwise. The graph is simple (no self-loop, no parallel edge) as to fit in the SPA criteria. It is undirected as we assume that the links are full-duplex in nature and the connected property ensures a path between any pair of vertices. The following measures are computed from G.

1) Adjacency Matrix: $ADJ(G) = \left[adj(i, j) \in \{0, 1\}\right]^{n \times n}$ is a symmetric binary matrix that represents the adjacency of the $G(V, E)$, where $|V| = n$.

2) Policy Set: Is a finite, nonempty set of policy tuples that includes $f^E_{i,j}$ and $\{g_{i,j} \leq K_{i,j}\}$ [(1)]

$$\text{PLC} = \left\{ < f^E_{i,j}(X^E), \{g_{i,j}(X^E) \leq K_{i,j}\} > \forall (i, j) \in E \right\}. \tag{1}$$

3) Variable Cost Matrix: $VCOST(t) = [c_{i,j}(t) \in \mathbb{R}^+]^{n \times n}$ represents the cost matrix at time instance t [(2)]

$$\{c_{i,j}(t)\} = \left\{ \begin{matrix} min f^E_{i,j}(X^E, t) \text{ if } i \neq j, (i, j) \in E \\ f^N_i(X^N, t) \qquad \text{otherwise} \end{matrix} \right\}. \tag{2}$$

All the n-diagonal values $c_{i,i}$ represent corresponding node-costs f^N_i and the nondiagonal ones represent the edge-costs $f^E_{i,j}$ for all valid edges, i.e., $(i, j) \in E$.

4) Normalized Cost Matrix: As the diagonal elements of $VCOST(t)$ represents weighted self-loops, it violates the "simple-graph" criteria. Therefore, a normalization is needed that relaxes the self-loops but preserves their effects onto the resultant 'simple-graph." We use *STEN* [27] technique to do so, which results $NCOST(t) = \left[\{c'_{i,j}(t) \in [0, 1]\right]^{n \times n}$.

5) Route Tree: The *RouteTree* $T_{s,d}$ is an m-way search tree that represents all possible paths between $v_s, v_d \in V$, and it holds the following properties. The destination vertex v_d is placed at root, all the leaves are identical, i.e., the source vertex v_s, every unit-branch (v_i, v_j) is weighed by its corresponding values in $NCOST[i, j]$ and varies over time, and for any intermediate vertex v_k, if $ANSC(V_k)$ and $DESC(V_k)$ denote its ancestors and descendants, then $ANSC(V_k) \cap DESC(V_k) = \phi$, this prevents any loop. The *MRoute* algorithm generates the tree and is discussed in Section III-C. Fig. 2 depicts the *RouteTree* $T_{1,2}$ with respect to the reference topology (Fig. 1); it shows the hop-counts and cumulative costs for each valid route (terminating at source vertex v_1). At hop-count=5, R_3 has two children, R_1 and R_5, as R_1 is source, and it terminates the search successfully. However, R_5 has no adjacency left that has not appeared in its ancestor set; therefore, $adj(R_5) - ANSC(R_5) = \phi$ and the search registers an unsuccessful termination. The *MRoute* algorithm has two phases: Phase-1 (growth phase) where the tree grows recursively, where it registers several unsuccessful terminations, and Phase-2 (shrink phase) eliminates all such branches.

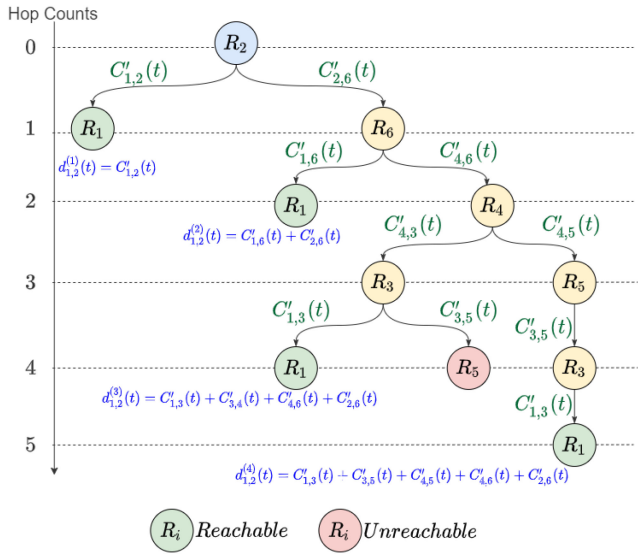

Fig. 2. *RouteTree* of $T_{1,2}$, rooted at R_2, all the reachable paths terminate with R_1 and unreachable node R_5.

6) Route Forest: For an n-node graph, there exists $\binom{n}{2}$ possible pairs of nodes. Each node produces a *RouteTree*. A collection of such trees form a *RouteForest*. It is generated by invoking *MRoute* parallelly $\binom{n}{2}$ times for each pair of nodes. The concurrency in execution is possible as the procedures are computationally independent and only the shared data structures are read.

B. Metric Formulation

We propose a composite metric for *MRoute* that constitutes the node cost $C^N_i(t)$ and edge costs $C^E_{i,j}(t)$. The node and edge parameters are listed in Table I. The formulation of node and edge costs is explained in the following paragraphs.

1) Node Cost: The node cost uses CPU and memory utilization as parameters. However, CPU and memory utilization cannot solely determine performance (i.e., a 20% utilized 8-core

TABLE I
LINK AND NODE PARAMETERS, MONITORED BY CP

Node Parameters (X^N)	**CPU**	Parameter	Core Count (nc)	Frequency (fc)	Utilization (uc)	
		Unit	Integer	MHz	[0,1]	
	Memory	Parameter	Volume (vm)	Frequency (fm)	Utilization (um)	
		Unit	MB	MHz	[0,1]	
Link Parameters (X^E)	**Parameter**	Bandwidth	Delay	Load	Reliability	MTU
	Units	Mbps	ms	[0,1]	[0,1]	[0,1500]

CPU processes more operations than that of a 80% single-core CPU and the same applies to the context of DDR4 vs. DDR2 memory). Moreover, with recent adaptation to network virtualization (e.g., Cisco IoU, CSRv), CPU and memory allocation is more flexible; it yields more heterogeneity in the network. Therefore, we propose a more robust metric formulation. The weight parameters α_c and α_m are left to the user to regulate (e.g., EIGRP K-values); the default value is set to 0.5.

$$C_i^N(t) = f_i^N\left(X_i^N, t\right) = \Big[\alpha_c(fc_i(t)nc_i(t)uc_i(t)) + \alpha_m(fm_i(t)vm_i(t)um_(i))\Big]. \tag{3}$$

2) Link Cost: The link cost function uses the same parameters as of EIGRP's. All the control traffic is targeted to the controller. This not only reduces the diameter of control flow from $O(n)$ (linear) to $O(1)$ constant, but also results in fast convergence. It is due to the fact that the topology is built inside the controller's memory and no control-packets are flooded to build neighborship. The SDN paradigm unifies the benefits of both OSPF and EIGRP as it builds a complete topology view like OSPF and uses all parameters of a more robust composite metric and supporting unequal-cost load-balancing like EIGRP (discussed in Section E).

$$C_{i,j}^E(t) = f_{i,j}^E\left(X_{i,j}^E, t\right) = \Big[(\beta_L MLD_{i,j}(t)) \times (\beta_B BW_{i,j}(t)\beta_D DLY_{i,j}(t)) \times \beta_r(1-RLY_{i,j}(t))\Big]. \tag{4}$$

The formulation in (4) is realized by its three components (BDP, load, and reliability). The bandwidth delay product BDP $= BW(t) \times DLY(t)$ measures the instantaneous end-to-end link capacity. The BDP is scaled by the mean load ($occupancy = BDP(t) \times MLD(t)$) and measures the amount of the occupancy in the link. The occupied capacity is scaled with additive inverse of reliability ($occupancy \times (1 - RLY)$) measuring the unreliability of the occupied capacity.

3) Normalized Metric: With reference to (3) and 4, the cumulative metric for a link $c'_{i,j}(t)$ is obtained by relaxing the node costs of both endpoints $(C_i^N(t), C_j^N(t))$ and scaling them by their corresponding load-share $(P_{i,j}(t), P_{j,i}(t))$ into the link cost $C_{i,j}^E(t)$ [(5)]. The parameter γ_N, γ_E are weighing factors, set by the user. The load-share of an interface is a proportion of the number of packets that passes through that interface over the total packet exchanged. The value is expressed in [0,1].

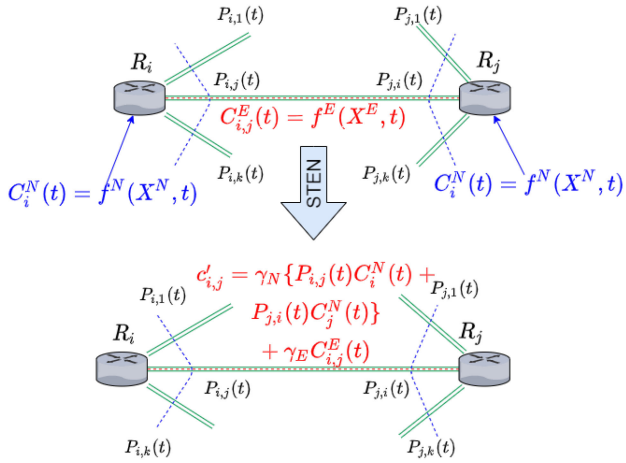

Fig. 3. Relaxation of node costs into edge using STEN [27] $T_{1,2}$.

$$c'_{i,j}(t) = \left[\gamma_N\left(P_{i,j}C_i^N(t) + P_{j,i}C_j^N(t)\right) + \gamma_E\left(C_{i,j}^E(t)\right)\right]. \tag{5}$$

Fig. 3 depicts the relaxation process to calculate cumulative metric $c'_{i,j}(t)$ at time t.

C. Algorithm Design

The proposed algorithm 1 is called *MRoute*; it takes source and destination vertex (v_s, v_d) as input, looks up to global structures $ADJ, NCOST$ during its recursive run-time, and returns a *RouteTree* $T_{s,d}$. The n-ary tree is stored into a hashed-dynamic array structure. It finds all possible paths between a pair of vertices using *backtracking* strategy. The problem is inherently brute-force in nature and the state-space complexity is NP-hard; therefore, we introduce optimization and relaxation, which is further explained in the later section of this article.

1) Optimizing Data Structures: MRoute adds nodes recursively into the *RouteTree*; the algorithm assumes $ANS(V_k)$ is of $O(1)$. Generally, an n-ary tree can be stored using either linked (noncontiguous) or array (contagious) structure. Since the data structure is unordered, each node must maintain $(|V| - 1)$ pointers it consumes in $O(n^2)$ space. However, not every time the network is mesh. Additionally, the recurrence decreases monotonically as more neighbors are visited; they would not appear as children. Therefore, the number of children decreases as the tree gets deeper, and choosing a n-ary tree structure is not space-optimal.

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, $NCOST$
Output: $T_{s,d}$
Data Structure: n-ary tree
Implementation: Dynamic array, Implicit Stack
Strategy: Recursive, Backtracking

```
begin
    if root = φ then
        root ← v_k
        if v_k = v_i then
            //Successful termination
            Return ST
        else
            // Unvisited children
            C_k ← {ADJ(v_k) − ANS(v_k)}
            if C_k = φ then
                //Unsuccessful Termination
                Return UT
            else
                for v_i ∈ C_k do
                    Update_Ancestors()
                    // Recur
                    MRoute(v_i, v_s, v_d)
                end
            end
        end
    end
end
```

In this article, we propose an optimal data structure to accommodate such a sparse array. Furthermore, when a graph is converted into tree, there will be multiple instances where the same node appears in various spaces. To eliminate any confusion during insertion, pointing, and displaying a node, an efficient and light index generation method is needed. For an n-ary tree, the following (6) generalized heap-indexing rule is adapted for this purpose.

$$\text{if } index(v_k) = i \text{ then } Parant(v_k) = \left\lfloor \frac{i}{n} \right\rfloor, \text{ and}$$
$$Child(C_{k,j}) = ni + jindex(root) = 0, n = |V|. \quad (6)$$

To avoid any segmentation error while using large topologies, a noncontiguous data structure is used to store the nodes for better scalability. Nodes are kept in random memory location Loc_k. The ID is calculated using rules in (6) and are kept along with the nodes data. A hash table maps index to location, and, thus, the search time is reduced to $O(1)$. Fig. 4 depicts the process.

2) Optimising Route-Forest Formation: Mroute is a very expensive algorithm in terms of space consumption, while generating a route-forest. The algorithm is invoked $O(n^2)$ times. The

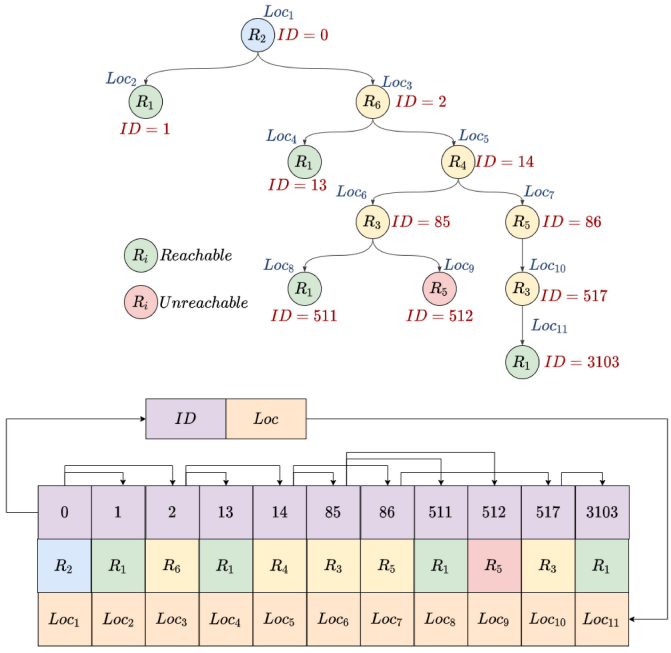

Fig. 4. Dynamic array-list with hash-table organization for fast searching. Loc_i is the virtual memory location that holds the router object R_j with ID k. Hash table maps an ID to its location.

calculation of *Route-tree* for any arbitrary pair of nodes is computationally independent, since they share common data structure $ADJ, NCOST$. This satisfies the criteria to execute them in parallel without any *Race-condition* (as no write operation on global structures takes place). Therefore, each $T_{i,j} \forall (i,j) \in V^2$ is computed parallelly in their individual threads. Also, $T_{i,j}$ can also be realized by reversing $T_{j,i}$ with $O(n^2)$ time.

D. Complexity Analysis

Lemma 1: MRoute is deterministic and loop-free

Proof: The proof is two part; we first show that the algorithm is loop-free, which will lead us to prove that it is deterministic. Also, properties mentioned in Section III-A.5 is referred in this proof.

MRoute selects children C_k of a nonleaf vertex v_k by filtering them with $ADJ(v_k) - ANSC(v_k)$. Therefore, any internal vertex v_i, if visited by a branch, cannot be a part of its descendant. Hence, it satisfies property 4, $ANSC(v_k) \cap DESC(V_k) = \phi$.

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 falls under the exponential class, then reduce it into the 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 equals 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 $ANS(V_k)$. With *Memoization*, these calls can be made fixed

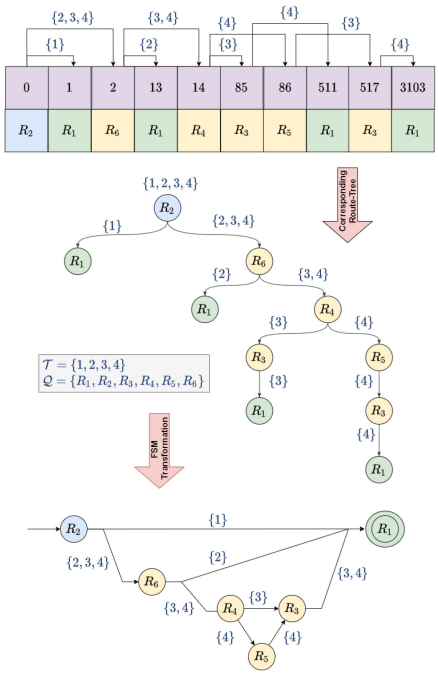

Fig. 5. Implementation of route-tag and generating FSM form route tree. The process depicts the transformation of data structures from the route-tree to route state graph.

through the run-time. Recursion is then invoked as many as b; therefore,

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 1 & \text{if } n = 2 \\ b.T(n-1) + \log_b|V| & \text{otherwise} \end{cases}. \quad (7)$$

Using Master theorem [28], it can be shown $T(n) = O(b^n \log_b|V|)$.

To prove the reduction, we will 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 by a binary string of length $|E|$, setting 1 s to all inclusive edges and 0 s to exclusive ones. Hence, it is reduces to an $n - SAT$ problem, where $n = |E|$. Thus, MRoute is NP-hard.

Lemma 1 also proves the algorithm is deterministic; hence, it is traceable. ■

1) Finite State Machine (FSM) Model and Route-Tag: Let $\mathcal{M}(\mathcal{Q}, \mathcal{T}, \delta, q_0, \mathcal{F})$ be a deterministic finite state machine such that $\mathcal{Q}$ is a finite, nonempty set of states $(\mathcal{Q} = \mathcal{V})$, $\mathcal{T}$ is a finite, nonempty set of route-tags $\mathcal{Q} \cap \mathcal{T} = \phi$, δ is the transition function $\delta : \mathcal{Q} \times \mathcal{T} \to \mathcal{Q}$, q_0 is the initial state, $q_0 = v_s \in \mathcal{Q}$, and $\mathcal{F}$ is a finite, nonempty set of final state(s), $\mathcal{F} = \{v_d\} \subseteq Q$.

Any *RouteTree* tree has unique paths between root and leaves. An identifier called *Route-Tag* tags each path. This compresses the exponentially large *Route-Tree* into state machine of size $O(|V|)$. We term this transformation $\mathcal{F} : T_{i,j} \to \mathcal{M}_{i,j}$ *route state transformation function* and $\mathcal{M}_{i,j}$ as *route state graph (RSG)*. Fig. 5 depicts the transformation with changes in the data structures.

TABLE II
TAG-COST TABLE FOR $\mathcal{M}_{1,2}$

Tags	$e_{1,2}$	$e_{1,3}$	$e_{1,6}$	$e_{2,6}$	$e_{3,4}$	$e_{3,5}$	$e_{4,5}$	$e_{4,6}$	Cost
1	1	0	0	0	0	0	0	0	$c^{(1)}_{1,2}(t)$
2	0	0	1	1	0	0	0	0	$c^{(2)}_{1,2}(t)$
3	0	1	0	1	1	0	0	1	$c^{(3)}_{1,2}(t)$
4	0	1	0	1	1	1	1	0	$c^{(3)}_{1,2}(t)$
Share	1	2	1	3	2	1	1	1	

E. Path Matrix

A path-matrix $\mathcal{P} = V^2$ is defined as $\{p_{i,j} \in \mathcal{P}\} = \mathcal{M}_{i,j}$. Every valid traversal in $\mathcal{M}_{s,d}$ corresponds to a feasible route between R_s, R_d. We propose two methods to encode the RSG.

1) Encode as Grammar: In this approach, the state machine is encoded into a set of production rules called grammar $\mathcal{G}(\mathcal{V}, \mathcal{T}, \mathcal{P}, s)$. This mode of encoding is useful when the routes are generated either as patterns or regular expressions. A grammar $\mathcal{G}$ is expressed as a quadruple, where $\mathcal{V}$ is the set of nonterminals $= \mathcal{Q} \subseteq V$, $\mathcal{T}$ is the set of terminals (route-tags), $\mathcal{P}$ is the set of regular production rules, and s is the start symbol. Encoding RSG into its grammar, summarizes the routes and parsing-ability is enforced using regular expressions.

2) Encode as Tag-Cost Table: The Tag-Cost table $TCT = \mathcal{T} \times E$ is a binary matrix; each row identifies one route-tag ($t_k \in \mathcal{T}$) and its corresponding edge set. The column-sum tells how many routes-tags are sharing a given edge (typically used for load-balancing). The Tag-Cost function is formulated in (8) and the Tag-Cost table in Table II. A Min-heap implementation of storing the tag-costs takes $O(1)$ time to return the best route and $O(\log_{\bar{b}}|V|)$ time to reorder them.

$$c^{(t_k \in \mathcal{T})}_{s,d}(t) = \sum_{(i,j) \in E} \Big(c_{i,j}(t) \times TCT[t_k] \Big). \quad (8)$$

Encoding RSG into TCT does leverage the reactive route-response mechanism, due to its constant time search for best route. Also, the tabular structure makes it easy to program and alter with varying edge costs.

IV. Estimation of Reliability Using RNN

As the normalized costs matrix ($NCOST$) varies over time (due to the variation of node or link cost), it creates a time-series matrix. However, the matrix comprises individual normalized links, which vary independently and does not provide performance analytics directly. Therefore, first, we segregate each link and treat them as individual time-series. Then, unlike predicting the traffic pattern or load, we focus more on predicting the trend. One of the challenges regards the online training in dynamic environment. A trained neural network often rejects to adapt sudden changes as outlier. Therefore, we aim to model the network dynamics by the degree of volatility of individual links.

A. Sharpe-Ratio-Based Approximation

In finance, the Sharpe-Ratio [29] is a widely used metrics in portfolio management that measures the volatility of a stock and

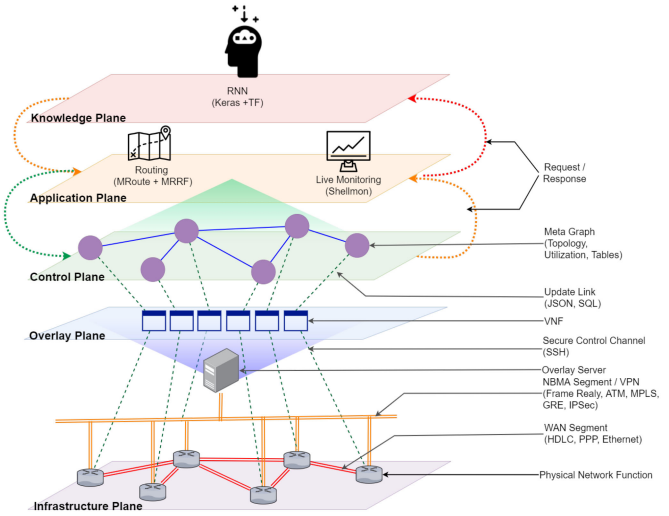

Fig. 6. Deployment diagram of the test-bed. Infrastructure plane holds routers; overlay server receives monitoring information and spawns VNFs per router. Control plane discovers topology and application plane operates on it. Knowledge plane is for self-learning, however, beyond the scope of the context.

estimates the risk associated with it. It is defined as the ratio of the sample-mean and the sample-standard deviation of a set and is proportional to the volatility. The approximation steps are as follows.

1) Calculate volatility $\mathcal{V}_{i,j}(t)$ of each $C_{i,j}(t)$ with a user-defined window size W rolling over time t.

$$\mathcal{V}_{i,j}(t) = \frac{\overline{C'_{i,j}([t-w:t])}}{SD(C_{i,j}([t-w:t]))} \forall (i,j) \in E. \quad (9)$$

2) Estimate the edge-wise hypothesis functions $\mathbf{h_{i,j}} \in \mathcal{H}$ as a autoregressive function using an RNN with a period of W.

$$\mathcal{V}_{i,j}(t+1) = \mathbf{h_{i,j}}(\mathcal{V}_{i,j}[t-w:t]). \quad (10)$$

3) Use $\mathcal{V}_{i,j}(t+1)$ as a metric to choose the best path. The proposed model uses offline training to build the initial model and thereafter uses online training to update it. We define a cutoff value $\epsilon > 30\%$.

V. Implementation

Fig. 6 depicts the deployment-diagram of our test-bed. A multitier approach is conceived for operational and functional segregation. The SDN philosophy of decoupling control and data plane has been the core design principal for the proposed architecture. However, the KP has been integrated on top to support SON capabilities.

A. A 5-Tier KDN Test-Bed

The architecture supports network-automation and SON. It finds optimal route using *MRoute* (self-optimization), then installs them to the underlying nodes by pushing device-specific configuration into the edge-devices (self-configuration), and guarantees a most-reliable route by keep updating them over the time (self-healing). Thus, it meets all the three criteria of SON. The following explains the working of the layers. Refer to the implementation details, including connection API and algorithm's code for more details [30].

1) Infrastructure Plane: This layer hosts physical and/or emulated network nodes (e.g., routers, L2/L3 switches, etc.). Routers are connected to overlay-plane securely using IPSec-DMVPN tunnels to exchange any control-traffic.

2) Overlay Plane: This layer interfaces between the infrastructure and CP. For each downstream router, a VNF process (agent) maintains a secure link (using SSH) to monitor the resource utilization. Additionally, it also injects configuration commands. We use Napalm library to automate the routes.

3) Control Plane: Resource and topology information are fused to generate the *meta-graph* in the CP. RESTConf is used to interface with overlay plane below and application plane above.

4) Application Plane: Application plane brings modularity in the architecture, such as monitoring, routing, etc. *MRoute* is one of such applications. However, there are other functions such as migration and monitoring, which are beyond the scope of the context of this article.

5) Knowledge Plane: The KP leverages the KDN paradigm. This component takes care of all data preprocessing, offline, and online training. It returns a trained model initially as an outcome of offline training. However, the model gets updates during online training whenever the trend changes. KDN functionalists can be divided into four main units.

1) *Preprocessing:* It performs data acquisition, data quality checks and validations, imputing, and standardization. Typically 70% of the overall process time is spent on this phase.
2) *Offline Training:* The offline training starts by dividing the data into training, validation, and testing for the ML model. It utilizes the historical data from the repository to train the model, and predicts the networking characteristics to produce a decision.
3) *Online Training:* It is used when the data is generated in a form of a sequence (such as time-series). Network resource utilization is a form of a time-series (NCOST) of a $n \times n \times t$ tensor, where n be the number of nodes nodes and t be the time.
4) *Modelling:* The learning algorithm learns from the fed data set, and generates a model for prediction. Since the problem can be classified as a time-series prediction type, RNN is chosen as the base architecture.

B. Performance Analysis of MRoute

The comparative analysis between *MRoute*, DUAL, and SPF (Fig. 7) benchmarks algorithms using six parameters. In this section, we present a comprehensive explanation to the results. Fig. 7(a) compares the time complexity with respect to the size of the network, and outcomes are plotted in log-scale; therefore, *MRoute* shows an exponential growth, as shown in Lemma 2; in comparison, DUAL and SPF 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

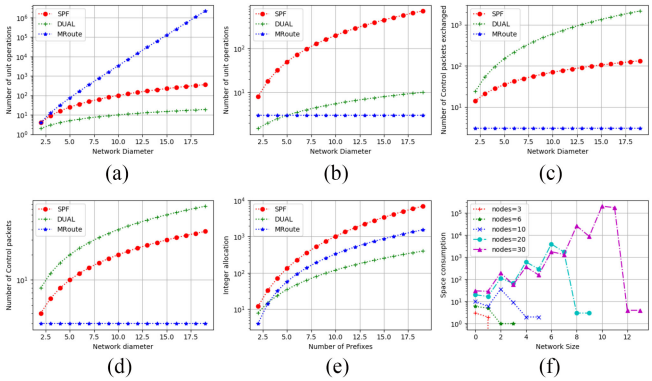

Fig. 7. Experimental results and comparison of *MRoute* against SPF and DUAL using the following parameters. (a) Time consumption to computing paths. (b) Time consumption to converge. (c) Control traffic for topology synchronization. (d) Space consumption for topology maintenance. (e) Control traffic for convergence. (f) Route-tree size.

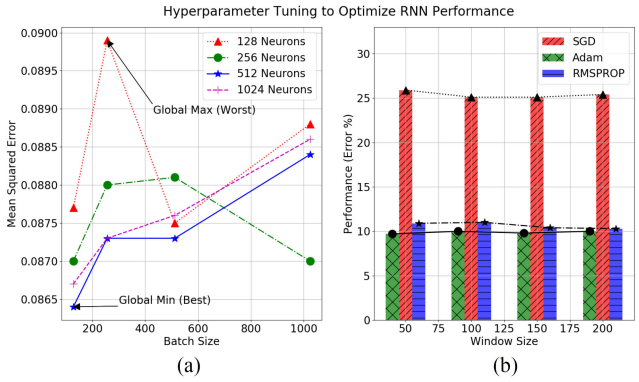

Fig. 8. (a) Comparison of accuracy (by mean squared error) with four network setups (128, 256, 512, and 1024); the global optima is reached with 128 Neuron at a batch size of 512. (b) Comparison of three optimizer algorithms (SGD, Adam, & RMSPROP), over a varying window size of [20]–[200], on which Adam gives the best result on average.

convergence state than that of SPF. We tuned the SPF to run on each downstream topology in parallel, simulating a multiarea OSPF network. It seems initially that DUAL is the optimum than its competitors. *MRoute* calculates all possible paths in advanced. Therefore, in the long run, if the topology remains unaltered, it would never enter a reconvergence process, which is not the case of the rest two. This situation is shown in Fig. 7(b), where the random link failure scenario causes SPF to reconverge every time; DUAL shows a better result as in some of the cases, feasible-successor exists or a neighbor replies with route much before the query reaches the network boundary. However, *MRoute* shows a constant reading here as it is an $O(1)$ 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 generalized 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 reconvergence, and polled for their liveliness using reliable updates and Hello protocols, respectively. Since OSPF uses link-state model, the total number of packets exchanged is higher than that of distance vector based on EIGRP. *MRoute* is designed as a centralized routing algorithm. Therefore, it does not exchange any discovery or update messages with other nodes. It updates only the controller, which is logically one hop away. This justifies Fig. 7(c) and (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. After the complete forest is generated, the state model gets built, which compresses them into tables and relinquishes the memory [Fig. 7(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 Fig. 7(f).

C. RNN Architecture

In this section, the design of the ML architecture is presented. We also introduce a few techniques used like hyperparameters fine-tuning and choosing the best optimization algorithm.

1) Hyperparameter Tuning: In this phase, the hyperparameters such as batch-size and number of neuron are tuned from experimental data. Fig. 8(a) depicts testing mean-squared error (MSE) cross-validation for three layers on a deep RNN using 200 epochs. The reason for this was to choose the appropriate number of neurons and the batch size for the training and validation data sets, error rate is measured using MSE. As highlighted in bold, the optimum hyperparameters have been 128 neurons and 512 batch size at 0.08 MSE.

2) Optimization Algorithm: Fig. 8(b) shows a comparison of the various optimizers proposed by Ghosh [31]. For the LSTM model, different sets of window sizes are tested. Three main variants gradient descent (SGD, ADAM, and RMSPROP) are compared. As a proof of concept, results show that predicting with a 200-ms window size using *Adam* can achieve a mean error rate of 10%.

3) Scoring: The proposed technique performs traffic prediction on the normalized reliability of the links. The result shows that, with the appropriate hyperparameters, reliability can be estimated with a mean 90% accuracy.

D. Online Learning

The online learning phase receives constant feedback from the network. If the predicted reliability deviates from the actual one, within a given threshold, the RNN needs to re-learn to adjust its weights. The relearning process takes place for multiple edges simultaneously. Hence, the tuning needs to be optimised. We use TensorFlow's *Early-Stopping* feature to accelerate the learning process, by monitoring the loss function's value and breaking the iteration whenever the loss converges to a value. Therefore,

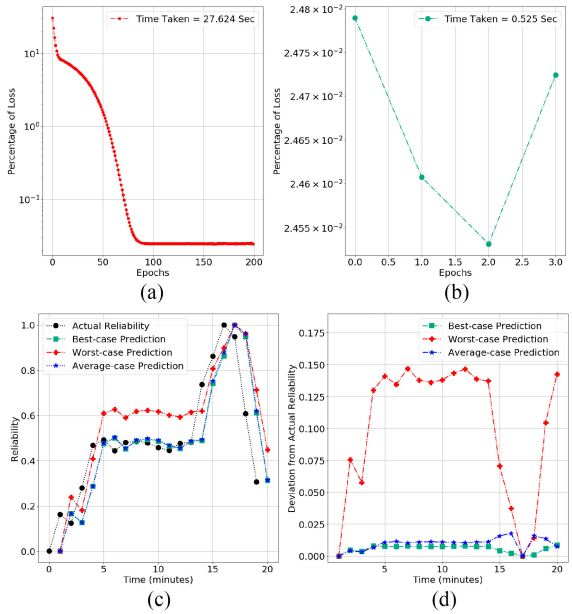

Fig. 9. Evaluation of the online learning. (a) Learning time with 200 epochs. (b) Accelerated learning with early-stopping enabled. (c) Comparing time-series prediction of reliability in the best, average, and worst-case scenario. (d) Comparison of the deviation in log-scale; the comparison is distinctive when there is less fluctuation.

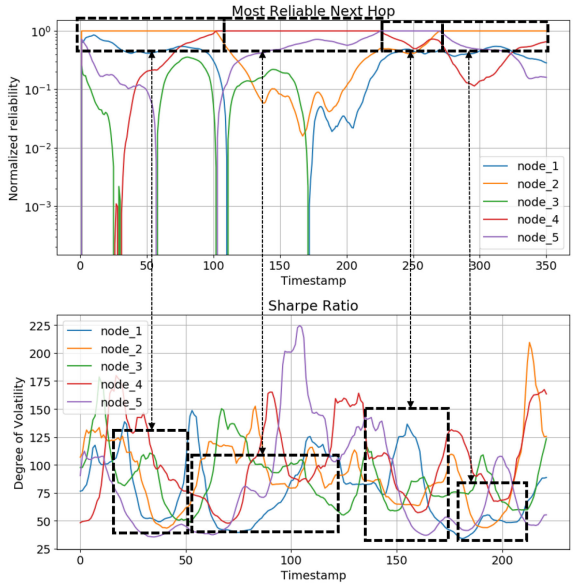

Fig. 10. Demonstration of self-healing through rapid-convergence. At timestamp [0]–[100] $Node_2$ is most reliable as the corresponding rolling Sharpe-Ratio has maximum descending gradient calculated on 100 timestamps. A similar pattern can be noticed for $Node_4$ during [100]–[240], $Node_5$ during [240]–[270] and $Node_2$ during [270]–[350]. The correlation is analytical; however, the RNN learns it.

the learning process does not need to run for all the epochs. Fig. 9(a) shows the loss function's characteristics spanning for 200 epochs, which took 27.6 s to complete learning. However, it can be noticed that the function actually settles around 55th epoch and stays constant since then. The acceleration is depicted in Fig. 9(b), where using early-stopping, the same network could be retrained in just 0.53 s. Thus, it reduces exponentially the time consumption of retraining the RNN, making it feasible for online training.

A more comprehensive comparison between the actual and predicted reliability is shown in Fig. 9(c) and (d). The first compares the best, worst, and average cases, sampling them down to a set of 20 instances, collected over a period of 20 min of online learning. The results show discrimination is prominent when there is less fluctuation in the data sets; it is more comprehensive when the deviation is plotted in log-scale [Fig. 9(d)].

E. Rapid Convergence and Co-Relation to Sharpe-Ratio

Fig. 10 depicts the varying reliability of five ENs over a time period of 350 stamps each of 10 s. The log scale is used to magnify the variation. Over the time, there are three nodes that have come up as the most reliable in the order of $None_2$, $Node_4$, $Node_5$, and again $Node_2$. During the experiment, we have emulated this dynamics by randomly altering various node and edge attributes. This causes the network to be extremely chaotic and the routing protocols to reconverge frequently, an effect that appears clearly in Fig. 7(a) and (e). *MRoute* has shown an $O(1)$ time convergence as routes are not only chosen in constant time. Additionally, the most-reliable node is switched instantly. A clear correlation between the learnt reliability and the *Sharpe-ratio* is also drawn using the relative dotted-boxes. As the *Sharpe-ratio* measures the degree of volatility, every time it meets a rapid depression. The corresponding router is chosen as most reliable. During the training, the RNN captures this trend and predicts accordingly. We set the window size of 100 time-stamps, and thus an offset of 100 can be seen in the time-axis of the two plots.

VI. Conclusion

In this article, we proposed a cognitive routing framework for KDN to support IoT applications for Industry 5.0. The framework uses an SPA named *MRoute*, that proactively computes all-possible paths between all pairs of nodes. Further, it uses Sharpe-ratio to measure volatility of each link and RNN with LSTM to learn trend. The framework uses online learning to tackle any dynamic network behavior. Results showed that the *MRoute* gives a constant-time convergence.

References

[1] K. Nisar *et al.*, "A survey on the architecture, application, and security of software defined networking: Challenges and open issues," *Internet Things*, vol. 12, 2020, Art. no. 100289. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2542660520301219

[2] P. V. Klaine, M. A. Imran, O. Onireti, and R. D. Souza, "A survey of machine learning techniques applied to self-organizing cellular networks," *IEEE Commun. Surv. Tut.*, vol. 19, no. 4, pp. 2392–2431, Oct.–Dec. 2017.

[3] D. D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski, "A knowledge plane for the internet," in *Proc. Conf. Appl., Technol., Architectures, Protocols Comput. Commun.* New York, NY, USA: Association for Computing Machinery, 2003, pp. 3–10.

[4] R. Hajlaoui, H. Guyennet, and T. Moulahi, "A survey on heuristic-based routing methods in vehicular ad-hoc network: Technical challenges and future trends," *IEEE Sensors J.*, vol. 16, no. 17, pp. 6782–6792, Sep. 2016.

[5] O. G. Aliu, A. Imran, M. A. Imran, and B. Evans, "A survey of self organisation in future cellular networks," *IEEE Commun. Surv. Tut.*, vol. 15, no. 1, pp. 336–361, Jan.–Mar. 2013.
[6] G. P. Fettweis, "A 5G wireless communications vision," *Microw. J.*, vol. 55, no. 12, pp. 24–36, 2012.
[7] J. G. Andrews *et al.*, "What will 5G be?," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 6, pp. 1065–1082, 2014.
[8] P. Wainio and K. Seppänen, "Self-optimizing last-mile backhaul network for 5G small cells," in *Proc. IEEE Int. Conf. Commun. Workshops*, 2016, pp. 232–239.
[9] P. V. Klaine, M. A. Imran, O. Onireti, and R. D. Souza, "A survey of machine learning techniques applied to self-organizing cellular networks," *IEEE Commun. Surv. Tut.*, vol. 19, no. 4, pp. 2392–2431, Oct.–Dec. 2017.
[10] W. T. Zaumen and J. J. Garcia-Luna-Aceves, "Loop-free multipath routing using generalized diffusing computations," in *Proc. IEEE Conf. Comput. Commun. 17th Annu. Joint Conf. IEEE Comput. Commun. Soc. Gateway 21st Century*, 1998, pp. 1408–1417.
[11] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, Dec. 1959. doi: 10.1007/BF01386390.
[12] J. Strassner, M. O'Foghlu, W. Donnelly, and N. Agoulmine, "Beyond the knowledge plane: An inference plane to support the next generation internet," in *Proc. 1st Int. Glob. Inf. Infrastructure Symp.*, 2007, pp. 112–119.
[13] J. Xie *et al.*, "A survey of machine learning techniques applied to software defined networking (SDN): Research issues and challenges," *IEEE Commun. Surv. Tut.*, vol. 21, no. 1, pp. 393–430, Jan.–Mar. 2019.
[14] Z. M. Fadlullah *et al.*, "State-of-the-art deep learning: Evolving machine intelligence toward tomorrow's intelligent network traffic control systems," *IEEE Commun. Surv. Tut.*, vol. 19, no. 4, pp. 2432–2455, Jan.–Mar. 2017.
[15] M. Chen, U. Challita, W. Saad, C. Yin, and M. Debbah, "Artificial neural networks-based machine learning for wireless networks: A tutorial," *IEEE Commun. Surv. Tut.*, vol. 21, no. 4, pp. 3039–3071, Jan.–Mar. 2019.
[16] Y. Zhao, Y. Li, X. Zhang, G. Geng, W. Zhang, and Y. Sun, "A survey of networking applications applying the software defined networking concept based on machine learning," *IEEE Access*, vol. 7, pp. 95397–95417, 2019.
[17] M. R. Haque, S. C. Tan, Z. Yusoff, C. K. Lee, and R. Kaspin, "Ddos attack monitoring using smart controller placement in software defined networking architecture," in *Comput. Sci. Technol.*, R. Alfred, Y. Lim, A. A. A. Ibrahim, and P. Anthony, Eds. Singapore: Springer, 2019, pp. 195–203.
[18] L. Yanjun, L. Xiaobo, and Y. Osamu, "Traffic engineering framework with machine learning based meta-layer in software-defined networks," in *Proc. 4th IEEE Int. Conf. Netw. Infrastructure Digit. Content*, 2014, pp. 121–125.
[19] A. Azzouni, R. Boutaba, and G. Pujolle, "Neuroute: Predictive dynamic routing for software-defined networks," in *Proc. 13th Int. Conf. Netw. Serv. Manage.*, 2017, pp. 1–6.
[20] Á. López-Raventós, F. Wilhelmi, S. Barrachina-Muñoz, and B. Bellalta, "Machine learning and software defined networks for high-density wlans," 2018, *arXiv:1804.05534*.
[21] S. Sendra, A. Rego, J. Lloret, J. M. Jimenez, and O. Romero, "Including artificial intelligence in a routing protocol using software defined networks," in *Proc. IEEE Int. Conf. Commun. Workshops*, 2017, pp. 670–674.
[22] F. Francois and E. Gelenbe, "Optimizing secure SDN-enabled interdata centre overlay networks through cognitive routing," in *Proc. IEEE 24th Int. Symp. Model., Anal. Simul. Comput. Telecommun. Syst.*, 2016, pp. 283–288.
[23] S. Lin, I. F. Akyildiz, P. Wang, and M. Luo, "QoS-aware adaptive routing in multi-layer hierarchical software defined networks: A reinforcement learning approach," in *Proc. IEEE Int. Conf. Serv. Comput.*, 2016, pp. 25–33.
[24] ONF, "OpenFlow switch specification," pp. 11–200, 2015. [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf
[25] "ONLP APIs for Applications–OpenNetworkLinux," Accessed: Jan. 19, 2022. [Online]. Available: http://opencomputeproject.github.io/OpenNetworkLinux/onlp/applications/
[26] R. Moskowitz, D. Karrenberg, Y. Rekhter, E. Lear, and G. J. de Groot, "Address allocation for private internets," RFC 1918, Feb. 1996. [Online]. Available: https://rfc-editor.org/rfc/rfc1918.txt
[27] S. Ghosh, T. Dagiuklas, and M. Iqbal, "Energy-aware ip routing over SDN," in *Proc. IEEE Glob. Commun. Conf.*, 2018, pp. 1–7.
[28] C. Yap, "A real elementary approach to the master recurrence and generalizations," in *Proc. Int. Conf. Theory Appl. Models Comput.*, 2011, pp. 14–26.
[29] W. F. Sharpe, "The Sharpe ratio," *J. Portfolio Manage.*, vol. 21, no. 1, pp. 49–58, 1994. [Online]. Available: https://web.stanford.edu/ wfsharpe/art/sr/sr.htm
[30] S. Ghosh, "GitHub–RishiCSE17/SO-KDN: Self organised knowledge defined netwrok," Accessed: Jan. 19, 2022. [Online]. Available: https://github.com/rishiCSE17/SO-KDN
[31] M. Wang, Y. Cui, X. Wang, S. Xiao, and J. Jiang, "Machine learning for networking: Workflow, advances and opportunities," *IEEE Netw.*, vol. 32, no. 2, pp. 92–99, Mar. 2018.

Saptarshi Ghosh (Student Member, IEEE) received the B.Sc. (Hons.) degree in computer science from the University of Calcutta, Kolkata, India, in 2010, the M.E. degree in software engineering from Jadavpur University, Kolkata, in 2016, and the M.Sc. degree in smart networks from the University of the West of Scotland, Glasgow, U.K, in 2017. He is currently working toward the Ph.D. degree in computer science and informatics with London South Bank University, London, U.K.

He is a module Leader of several core CS modules with London South Bank University. He is JNCIA (DevOps) certified, and was a Software Developer and Network Engineer. He has contributed to several research and software engineering projects funded by Erasmus+, Innovate U.K., and Defence Science and Technology Laboratory. His research interests include SD-WAN, network programmability and automation, cognitive-routing, and deep reinforcement learning.

Mr. Ghosh is a recipient of GATE and Erasmus-Mundus Scholarship. His Ph.D. research is under the EU-Horizon 2020 project, supported by Marie-Curie Fund with the research area focused in machine learning's application to self-organized SDN for 5G and beyond.

Tasos Dagiuklas (Senior Member, IEEE) received the engineering degree from the University of Patras, Patras, Greece in 1989, the M.Sc. degree from the University of Manchester, Manchester, U.K. in 1991, and the Ph.D. degree from the University of Essex, Colchester, U.K., in 1995, all in electrical engineering.

He is currently a Leading Researcher and expert in the fields of smart Internet technologies. He is the Leader with the Smart Internet Technologies (SuITE) Research Group, London South Bank University, London, U.K., where he is the Head of Cognitive Systems Reseach Centre. He is a Principal Investigator, Co-Investigator, Project and Technical Manager, Coordinator, and Focal Person of more than 20 internationally R&D and capacity-building and training projects in the areas of fixed-mobile convergence, 4G/5G networking technologies, VoIP, and multimedia networking. His research interests include systems beyond 5G/6G networking technologies, programmable networks, UAVs, V2X communications, and cybersecurity for smart Internet systems.

Muddesar Iqbal (Member, IEEE) received the Ph.D. degree in wireless communication networks from Kingston University, Kingston, U.K., in 2010.

He was with several universities in U.K., EU, and South East Asia. He is currently a Senior Lecturer with London South Bank University, London, U.K. He has been a Principal Investigator, Co-Investigator, Project Manager, Coordinator, and Focal Person of more than 15 internationally teamed research and development, capacity-building and training projects, resulting in several patented inventions and commercial products. His research interests include Internet of sense for Industry 5.0, intelligent autonomous machines/robotics, conversational AI and Chabot's, personalization/recommendation, 6G context-aware systems, and collaborative cognitive–communication systems.

Xinheng Wang (Senior Member, IEEE) received the B.Eng. and M.Sc. degrees in electrical engineering from Xi'an Jiaotong University, Xi'an, China, in 1991 and 1994, respectively, and the Ph.D. degree in electrical engineering and electronics from Brunel University, Uxbridge, U.K., in 2001.

He is currently the Founding Head with the Department of Mechatronics and Robotics, Xi'an Jiaotong-Liverpool University, Suzhou, China. He is also a Team Leader with the Jiangsu Innovation and Entrepreneur Programme. He has broad academic working experience in China, England, Wales, and Scotland for more than 20 years. He has extensive research experience in Internet of Things (IoT), wireless mesh networks, indoor positioning, Big Data analytics, and applications for smart cities. Along with nearly 30 research projects sponsored from EU, U.K. EPSRC, Innovate U.K., China NSFC, and industry, his research in each area has led to an impactful industrial product. His research interests include 6G networks and industrial Internet of Things, indoor positioning and data services, acoustic localization, communications and sensing, and smart services for group travelers.

Dr. Wang is an IET Fellow. His collaborative research in acoustic localization with Zhejiang University has won the first place in Microsoft Indoor Localisation Competition in sound group.