A Cognitive Routing Framework

for

Self-Organized Knowledge-Defined Networks

Doctoral Thesis

by

Saptarshi Ghosh

School of Engineering, London South Bank University

Author Note

This thesis is submitted in partial fulfilment of the requirement for the PhD degree in

Computer Science and Informatics

Dedication

I dedicate this thesis to my parents, Mr Nilmani & Mrs Subhra Ghosh for guiding me to differentiate between the need and the temptation and teaching me never to compromise with ethics. The research would never be possible without the guidance and constant push of my supervisors Prof. Anastasios Dagiuklas & Dr Muddesar Iqbal. And nevertheless, my lovely wife Riu, for being the support during some of the most decisive moments.

I also dedicate this work to all my teachers and professors who have inspired me to fly high, fueled me to never give up, and instilled confidence in me to aim for prepending the "Dr." before my name. Thank you, Mr Kumarjit Mandal, Mrs Paulami Bhattachariya, Dr Siddhartha Roy, Dr Maumita Mitra, Prof. Samiran Chatterjee, Prof. Amitabha Gupta, and Mr Arindam Saha.

My beloved students deserve their due credit. I'd like to dedicate the work to all my students for nurturing my teaching and presentational skills. My rocking colleagues from the Smart Internet Technology Lab; Tim, Kasra, Emeka, Godwin, John, and Brahim, for those sleepless nights.

Finally, I'd like to dedicate this thesis to the One, for giving me the strength to decide during the most idiosyncratic moments, which has landed me here writing my concluding remarks. I'd strongly wish to continue my work and dedicate it to the service of mankind for a better tomorrow.

Abstract

This study investigates the applicability of machine learning methods to the routing protocols for achieving rapid convergence in self-organized knowledge-defined networks. The research explores the constituents of the Self-Organized Networking (SON) paradigm for 5G and beyond, aiming to design a routing protocol that complies with the SON requirements. Further, it also exploits a contemporary discipline called Knowledge-Defined Networking (KDN) to extend the routing capability by calculating the "*Most Reliable*" path than the shortest one.

The research identifies the potential key areas and possible techniques to meet the objectives by surveying the state-of-the-art of the relevant fields, such as QoS aware routing, Hybrid SDN architectures, intelligent routing models, and service migration techniques. The design phase focuses primarily on the mathematical modelling of the routing problem and approaches the solution by optimizing at the structural level. The work contributes *Stochastic Temporal Edge Normalization* (STEN) technique which fuses link and node utilization for cost calculation; *MRoute*, a hybrid routing algorithm for SDN that leverages STEN to provide constant-time convergence; *Most Reliable Route First* (MRRF) that uses a Recurrent Neural Network (RNN) to approximate route-reliability as the metric of MRRF. Additionally, the research outcomes include a cross-platform SDN Integration framework (*SDN-SIM*) and a secure migration technique for containerized services in a Multi-access Edge Computing environment using Distributed Ledger Technology.

The research work now eyes the development of 6G standards and its compliance with Industry-5.0 for enhancing the abilities of the present outcomes in the light of Deep Reinforcement Learning and Quantum Computing.

Keywords: *Routing, KDN, SDN, SON, ML*

Acknowledgements

Funder	Funding details
Horizon-2020	This project is funded by MSCA-RISE - Marie Sklodowska-Curie Research and Innovation Staff Exchange (RISE) scheme under the project SONNET (*Self-OrganizatioN towards reduced cost and eNergy per bit for future Emerging radio Technologies*). with the grant agreement no. H2020-MSCA-RISE- 2016-734545.
Innovate-UK	The research received partial funding from the Innovate UK project VICTORY under the Cyber Security Academic Startup Program (CyberASAP) Knowledge Transfer Program.
DSTL	This research received partial funding from the Defense Science and Technology Laboratory for the project *Cognitive Routing System* (CoRoS) under the Defense and Security Accelerator program (DASA).

Table of Contents

List of Figures

List of Tables

List of Algorithms

Contribution to Knowledge

Major Publications

Title	Published in	Ref
Energy-Aware IP Routing Over SDN	**2018** IEEE Global Communications Conference (GLOBECOM)	[1]
Reliable resource provisioning using bankers' deadlock avoidance algorithm in MEC for industrial IoT	**2018** IEEE Access, vol. 6	[2]
Co-operative and hybrid replacement caching for multi-access mobile edge computing	**2019** European Conference on Networks and Communications	[3]
BlueArch – An implementation of 5G Testbed	**2019** Journal of Communication	[4]
SDN-Sim: integrating a system-level simulator with a software-defined network	**2020** IEEE Communications Standards Magazine, vol. 4.	[5]
A centralized hybrid routing model for multi-controller SD-WANs	**2020** Transactions on Emerging Telecommunications Technologies, vol 32	[6]
A Self-Organized Knowledge Defined Networks Architecture for Reliable Routing	**2020** 4th International Conference on Information Science and Systems ICISS 2021	[7]
Robust, Resilient, and Reliable Architecture for V2X Communications	**2021** IEEE Transactions on Intelligent Transportation Systems, vol 22.	[8]
A Cognitive Routing Framework for Reliable Communication in IoT for Industry 5.0	**2021** IEEE Transactions on Industrial Informatics, vol. 18, issue 8.	[9]

Patent

Title	Published by	Doc. ID.
Intelligent SDN Routing Framework	UK Intellectual Property Office	GB2578453

Book Chapter

Book title	Chapter title	Publisher
Blockchains: Empowering Technologies & Industrial Application	Security, privacy, and trust of distributed ledger technology	IEEE/Willey

Chapter 1: A Preamble to Network Intelligence

1.1.Towards the Industry 5.0 paradigm

The continuity in development has been a prominent feature of the history of mankind. Engineering, the purposeful implementation of science, perhaps is a perfect example of the preceding statement. Arguably, the most influencing driver for this substantial advancement is the effect of the Industrial Revolution. A tiny fraction of three centuries has inked monumental changes in our native instincts along a timeline of three hundred thousand years that humans have occupied the earth. During the early stages of industrialization in the year 1800, the world population was less than a billion; on the verge of the Fifth industrial revolution in 2021, it is reaching 8 billion, and expected to be 11 by the year 2100. However, despite this exponential growth, the mean life expectancy has boosted from 35 years in 1800 to 72 years in 2021, falsifying the Malthusian growth model.

The rapid digital transformation driven by the Internet came in three phases, often called The-Three-Waves. The first wave came from 1985 to 99 when companies like Microsoft, Apple, IBM, Cisco, Sun, etc., developed hardware and software for the Internet. During the second wave (2000-15), companies like Google, Amazon, Facebook, etc., built applications over the Internet infrastructure; App-Economy was born. The third wave (2015 onwards) has brought Internet Ubiquity. On the bright side, the consequences of digitization and globalization have significantly improved the socio-economic landscape worldwide.

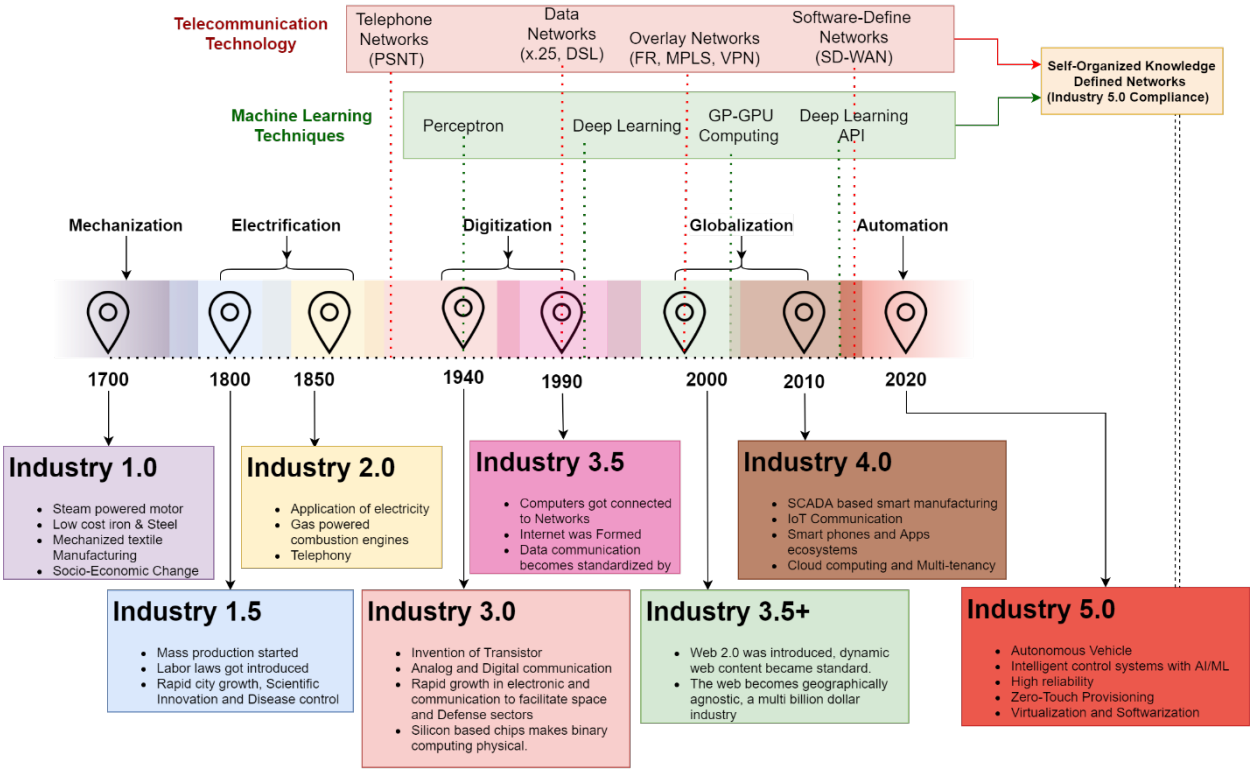

Figure 1 A summary of the evolution of modern industries and the convergence of Networking technologies and Machine Intelligence

Figure 1 summarizes the four iterations of industrial revolutions with their respective contributions to modernizing society and the promises of the fifth one. However, the timeline needs to be analyzed slightly differently to understand the key enablers of such futuristic claims. From the age of electrification, two parallel yet mutually exclusive development have taken place. These include telecommunication systems and machine intelligence. Presently, communication networks have become enough Softwarized that programming and automating them are now mainstream functions. Likewise, the rich ecosystem of machine learning and deep learning APIs has made their implementation seamless and agile. Hence, the Software-Defined Networking (SDN) model has paved the path for seamlessly integrating the power of machine learning in solving networking problems. Such a network is called a Knowledge Defined Network (KDN), which is self-aware and needs minimal human intervention to be managed, complying with the industry 5.0 needs.

1.2. Software-Defined Networking and SD-WAN

The increasing complexity of contemporary mobile and application-centric networks is pushing traditional WAN architectures to their limit. Due to the rise of cloud services and edge computing, enterprise networks' inter-site data transfer volume rise rapidly. Moreover, enterprises tend 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 consuming 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 with more manageability, autonomy, and network security. Chapter 3 of this thesis discusses a hybrid routing algorithm for multi-controller SD-WANs that computes all-possible routes proactively and serves them on-demand. It also presents a model that results in a rapid convergence across edge devices and the synchronization mechanism among several controllers and a test-bed implementation.

1.2.1. Content Distributed Networking over SD-WAN, a use-case

Telecommunication industries across the world are going through a massive transformation phase. The increasing demand for high-quality online content like streaming from "Over the Top" (OTT) platforms (e.g., Netflix, Amazon Prime, YouTube) is driving the telcos to optimize their existing network architectures. Content Distribution Network (CDN) that caches static contents into a proxy server, e.g., Point of Presence (PoP), enables various alternatives to reduce the delays. Content-caching is a clever and widely used method [9] that keeps the response of most frequently requested contents and serves them from local storage rather than redirecting from the origin. Studies have shown that using a predictive approach may reduce the overhead cache value per day up to a fraction (10% - 20%) of the cache size [10].

Multi-Access Edge Computing (MEC) [11] enhances the availability of cloud services by distributing them to 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 centralized control, which results in control-plane functionalities such as routing, policing, etc., to execute as services. Unlike traditional distributed computing models where network devices synchronize by exchanging control packets, SDN decouples the control plane from the forwarding plane [12]. A network device sends its local information to the controller to compute the forwarding logic. SDN results in greater autonomy and programmability in a network configuration, including rapid policy deployment that collectively reduces the CapEx and OpEx.[13]. 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. The communication method for NBI is typically RESTFul, whereas the SBI uses OpenFlow. Applications send generic RESTful configuration requests to the controller defined by a policy, which gets translated into device-specific configuration and injected into the devices. Therefore, SDN controllers play a principal role in abstracting the granularity of the network infrastructure and easing the device configuration for application developers.

With the introduction of Network Function Virtualization (NFV), it is possible to Virtualize Network Functions (VNF) and host them in a remote computing platform (e.g., Cloud, Remote servers, etc.). However, not all network functions, such as Radio transceivers, sensors, 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 [14]. The orchestrator program manages the VNF associated with optimal placement, resource allocation, and provisioning [15]. A challenge in the SDN-MEC

based design of the CDN environment is to optimize the forwarding traffic. SDN offers a birds-ey view of the network, which eases 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 an ad-hoc routing process that discovers routes on-demand, whereas proactive routing finds Routes before applying them. SDN is also effective for hardware independence. For a mobile ad-hoc network, the SDN-NFV approach demonstrates better agility for high volume data than a non-SDN-based one, which stands superior in large-signal load [16].

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 centres, and telcos are reluctant to scrap all their existing non-SDN-compliant forwarding hardware for the sake of enjoying the SDN benefits [17]. The primary reason is 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. These results in the overlay-SDN model (initially introduced by VMWare through their NSX platform [18]), which cancels the need to replace the Data-plane devices instead of creating a virtual overlay network that connects it to the control plane. The overlay tunnels enable the edge devices, i.e., routers and layer-three switches, to communicate with the controller using the Internet as a fabric. The control plane gets segregated, as the global controls reside in the remote-orchestrator, whereas the device-specific controls stay in the edge devices. The architecture is called Software-defined WAN or SD-WAN [19][20]. In SD-WAN architecture, the orchestrator hosts the application plane and interacts with the controller cluster. The 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) [21], and Citrix uses Adaptive Transport Protocol [22] for this purpose.

SD-WAN architecture leverages the centralized routing model where edge devices do not exchange control information. Instead, 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 [23] and EIGRP [24], inherently distributed in nature. This opens up a new dimension in the routing protocol design philosophy that aims to compute routes from a centralized perspective. This is not to be misinterpreted by drawing parallels to some centralized mechanisms in traditional routing such as Designated Routers in OSPF, Route-Reflector in BGP, Root-Bridge in Spanning Tree protocol, or Next-Hop Server in DMVPN. In all these cases, the central node's functionality is to collect and distribute network information. The ultimate computation is carried out on the nodes in a distributed fashion. SD-WAN Routing calculates routes on an aggregated topology built by fusing information from individual edge nodes and configuring the routing tables to the edge. Chapter 3 presents a centralized rapid-convergence routing algorithm (*MRoute*) for SD-WAN [25][26] that proactively finds all-possible paths for all pairs of nodes, ranks them, updates rank over time, and serves routes on-demand and a multi-controller implementation *MRoute*. Runtime performance is compared with OSPF and EIGRP emulating them on an SDN test-bed that comprises [27] IaaS Cloud, OpenDaylight [28] as the controller, and Mininet [21] as the forwarding plane.

1.2.2. Contribution to Knowledge

Chapter 3 of this thesis describes a solution to some critical 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 proactively calculates all-possible paths between all pairs of nodes and reactively serves them on demand.
3. An SD-WAN testbed to implement, experiment, and benchmark the proposed model.

1.3. System-Level Simulation for 5G and Hybrid-SDN Integration

Design and structural complexity are skyrocketing with the introduction of diverse technology paradigms in next-generation cellular and vehicular networks. The beyond- 5G use cases such as mobile broadband, URLLC, 5G-V2X, and UAV communications require ultra-low latency and high throughput and reliability with limited operational complexity and cost. These use cases are explored in 3GPP Releases 16 and 17. To facilitate end-to-end performance evaluation for these applications, SDN-Sim - integrating a System Level Simulator (SLS) with a Software Defined Network (SDN) infrastructure is proposed. While the SLS models the communication channel and evaluates system performance on the physical and data link layers, the SDN performs network and application tasks such as routing, load balancing, etc. Chapter 4 discusses an architecture that replicates the SLS-defined topology into an SDN emulator for offloading control operations. It uses link and node information calculated by the SLS to compute routes in SDN and feeds the results back to the SLS. The chapter also proposes the data modelling and processing, replication, route calculation frameworks, and architecture.

1.3.1. SDN integration, a Vehicular Networking perspective

Towards 5G/B5G, the third-generation partnership project (3GPP) is finalizing Release 16 and defining Release 17[1]. In the area of vehicular networks, the 3GPP, in partnership with the Fifth Generation Automotive Association (5GAA), is driving the efforts on the 5G-based vehicle-to-everything (V2X) paradigm, which adds advanced features to the LTE-V2X from Release 14, particularly in the areas of support for ultra-reliable and low-latency communication (URLLC) applications for the future intelligent transport systems (ITS) [29],[30], [31]. In the evolution path from LTE-V2X to 5G-V2X, the authors in [29] advocated the incorporation of software-defined networking (SDN) in the architecture to enhance the system performance through SDN's capabilities in facilitating intelligent multi-hop routing, dynamic resource allocation, and advanced mobility support, among others.

To evaluate the performance of proposed algorithms, techniques, and frameworks for any new era of communication networks, numerical simulations, mathematical analyses, and field trials are the three main approaches being employed. Though analytically tractable, mathematical methods (e.g., stochastic geometry tools) are often constrained by simplifying assumptions that potentially limit their use in modelling large-scale, highly complex, and dynamic networks. Realistic performance can be measured in live operating environments. However, the financial and operational requirements are costly and practically infeasible for the early design and development stages. Hence, in the past few decades, simulations have become essential tools for the assessment of network performance due to the apparent cost and implementation advantages [32].

Depending on the performance metrics under investigation, simulators can be categorized into three: Link Level Simulator (LLS), System Level Simulator (SLS), and

[1] https://www.3gpp.org/news-events/2058-ran-rel-16-progress-and-rel-17-potential-work-areas

Network Level Simulator (NLS). The LLS examines detailed, bit-level physical (PHY) layer functionalities of a single link. The SLS evaluates the performance of links involving many base stations (BSs) and user equipment (UEs) at the medium access control (MAC) layer (with the PHY abstracted). It focuses on the radio access network/air interface and facilitates analyses of resource allocation, capacity, coverage, spectral and energy efficiencies, among others. The NLS, however, assesses the performance of protocols across all layers of the network, including control signalling and backhaul/fronthaul issues. Performance is characterized using metrics such as latency, packet loss, etc. [33].

Besides metric-based classification, simulators can also be grouped based on radio access technologies supported (e.g. cellular, vehicular, Wi-Fi, etc.), programming languages (MATLAB, Python, C++, etc.), licensing option (open source, proprietary) or network scenario capabilities (LTE, 5G, B5G, etc.) [33][2]. While the SLS does not simulate beyond the MAC layer, the NLS simulates networks up to the application layer. However, the implementation and computational complexity of NLS become very high when a large number of nodes are involved [34].

Another significant paradigm shift in network design took place with the advent of SDN [34]. It decouples the control (signalling) plane from the data (forwarding) plane and runs applications in the application plane to manage the network. This brings transparency to network design and lets software developers write applications for managing the networks, keeping the internal structure in abstraction. Each layer uses several interfaces to communicate with each other. The control plane communicates with both application and data planes using

[2] Representative simulators include the Vienna LTE-A and 5G simulators for LLS and SLS (https://www.nt.tuwien.ac.at/research/mobile-communications/vccs/), and the 5G-K Simulators for the NLS (http://5gopenplatform.org/main/index.php).

north and southbound interfaces, respectively. In the case of a cluster of controllers, east and westbound interfaces are used for communicating among them.

As the default southbound protocol for SDN, OpenFlow uses Flow Tables (FT) to perform packet forwarding. Each FT entry is a forwarding rule determined by the controller. A forwarding rule has mainly three significant fields, a “match,” an “action,” and a “priority.” A “match” is some criteria for an inbound packet to be checked. A packet that satisfies the criteria is termed a “table hit”; otherwise, it is a “table miss.” For each case, an action is defined such that the OpenFlow switch executes on the subjected packet. If a packet satisfies matches from multiple flow rules, priority breaks the tie. The SDN Controllers populate flow entries. The OpenFlow switch requests the controller for every table miss and the controller replies with a flow entry. If the controller cannot resolve an action, it is set as a “drop,” The switch does not process the packet. The decoupled control plane reduces computational cost on forwarding devices by offloading the control packet processing tasks to the controller. Therefore, SDN offers better modularity, programmability, agility, automation, and load balancing capability than traditional networks. Also, the SDN-based approach is becoming the norm in network design practices for cloud computing and 5G.

1.3.2. Contribution to Knowledge

Chapter 4 of this thesis presents a novel SDN-based System Level Simulator (SDN-Sim) platform where the SLS-Stage runs in MATLAB and the NLS stage using python3. The architecture uses the SDN design philosophy to reduce the overall computational complexity of the system considerably. The computationally demanding upper layer network functions (e.g., inter-cellular routing) are offloaded to the virtualized cloud infrastructure. Low-level network information (e.g., Channel model, topology, etc.) is mapped from SLS to SDN. The virtual infrastructure uses VMWare ESXi servers; OpenFlow and RESTConf are the south and

northbound protocols, OpenDaylight as the SDN controller, and GNS3 and Mininet-wifi as the data plane emulation.

1.4. Efficient IP Routing for SDN

The routing protocols play a vital role in saving energy, especially by minimizing the time a packet travels from source to destination. Energy-aware routing protocols aim to select a route that engages routers to minimum overall energy consumption.

1.4.1. Energy-aware SDN Routing

Energy awareness techniques in routing algorithms have been in the limelight of the research community for a while. For the last few decades, it has been evident that Moor's law is broken, and devices are becoming more powerful. However, on the flip side, they are becoming more power-hungry, and the advancement in battery capacity is not coping with the rate. Therefore, designing energy-efficient software has become a trend in the research community to meet the green objective. Contribution from several fields has made it a prosperous domain. In [35], the authors present how energy savings can be optimized by using Microsoft's MAUI framework by offloading the application. But when the local energy is saved by executing an intense part of the program remotely, communication cost is proportional to the routing time. The routing algorithm plays a vital role in the energy savings schemes. Routing protocols developed for homogeneous networks, such as Ad-hoc On-Demand Distance Vector (AODV), don't work for the heterogeneous environment, as the resource utilization of network devices affects efficiency. Hence, Resource-Aware Routing for Low powered and Lossy Networks (RPL) was standardized (RFC 6550) [36], which also formulates the node cost calculation metric. Link cost calculation typically depends on the nature & type of the network; however, some generalized techniques are discussed in [37], [38].

Software-Defined Networking (SDN) [39] is also becoming the de facto standard of modern networking. It decouples the control and data plane. The control plane (CP) is a logically centralized entity hosted by one or many devices called Controllers; it instructs the traffic forwarding rules to the Data Plane (DP), which constitutes switches, which only forwards. CP bridges with the DP with OpenFlow protocol and switches register the instructions in OpenFlow Tables.

1.4.2. Contribution to the Knowledge

Within the context of this research work, an energy-aware routing algorithm has been designed and developed that exploits application offloading. Furthermore, it proposes a resource-aware routing algorithm for SDN, which monitors the resource utilization of network devices (nodes) and channels (links), using a push agent and fetches topology and flow table information from the controller. Using Link Queue Modelling [40] and Stochastic Network Calculus [41] a route is guaranteed that avoids busy nodes and uses unutilized ones. Results show the validity of the algorithm.

1.5. Cognitive Routing, an Industry 5.0 perspective

Complex communication across interconnected devices poses a unique reliability challenge on the verge of the 4th industrial revolution and the beginning of Industry 5.0. Time-critical applications such as industrial and mission-critical communication systems demand stability in scale. Recent progress in SDN routing has shown significant improvement in various 5G KPIs but fulfilling the Ultra-Reliable Low-Latency Communication (URLLC) to achieve seamless Industrial-IoT communication stays inadequate. One of the significant challenges deals with dynamic network behaviour. The Knowledge-Defined Networks (KDN) bridges the gap by extending SDN architecture with Knowledge Plane (KP) on top, which learns the

network dynamics to avoid suboptimal decisions. Cognitive Routing is a relatively young discipline that uses Machine Learning (ML) algorithms to optimize routing decisions. Research shows that the majority working in this area focus on traffic prediction and route optimization; however, the reliability approximation exploration is limited. Cognitive Routing leverages the Sixth Generation (6G), SON, with the self-learning feature.

Chapter 3 of this thesis covers a self-organized cognitive routing framework to support URLLC. In this context, a bespoke KDN reduces end-to-end latency by choosing the most-reliable path with minimal probability of route-flapping. The proposed framework pre-calculates all possible paths between every pair of nodes and ensures Self-Healing with a constant-time convergence. Furthermore, it uses Sharpe-Ratio to measure volatility and forecasts its trends using RNN with LSTM. The framework uses online learning to tackle network dynamics. An experimental testbed benchmarks the proposed framework to compare the convergence parameters against SPF and DUAL.

1.5.1. Self-Organized Knowledge-Defined Networking (SO-KDN)

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 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 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 a programmable, vendor-agnostic communication platform. 5G leverages SDN at its core to virtualize network services (NFV), and ISPs use it in WAN deployment (SD-WAN). SDN provides a bird's eye view of the network where the control plane accumulates global knowledge about the underlying topology and flows. Additionally, the data plane generates enough that 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 exchange control packets. An efficient routing protocol aims to avoid sub-optimal 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 using analyzing the network's behaviour provides a better heuristic which eventually reduces the convergence probability. In SDN [42] 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 [43]. However, the application of Machine Learning (ML) in Route-Optimization is a relatively new domain; at the time of writing this paper, 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)[44], where the primary objective is to accumulate holistic information from a supervising Control Plane (CP) of an underlying IP network, analyze them to extract knowledge that generalizes the network behaviour. This knowledge eventually helps to bypass the need for using costly heuristic Routing algorithms, having preserved the equal adaptation capabilities to network

dynamics[45]. Self-Organized Networking (SON) [46] 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), 10Gbps pack data rate, sub-millisecond latency, support of ultra-high reliability, improved security, privacy and flexibility, and reduction of CAPEX and OPEX [47] [48] [49]. SON constitutes the following three entities.

- **Self-Optimization** provides several control-plane (CP) optimization strategies such as Caching, Routing, load balancing, etc. which are invoked autonomously. Relevant algorithms calculate the optimal values of several decision variables w.r.t., the set of constraints, called policies.
- **Self-Configuration** automates the injection of the decision parameters (e.g., operational and radio config) to the underlying data-plane devices.
- **Self-Healing** provides high availability to the overall network. A typical model uses detection, diagnostic and compensation sequences to automate the recovery process.

Recent development in SON shows significant use of ML to accelerate the performance of its constituents [50].

1.5.2. Contribution to Literature

Chapter 3 of this thesis describes Most-Reliable-Route-First (MRRF), an Intelligent Routing algorithm for Self-Organized Knowledge-Defined Networks. The model initially calculates all possible paths for all pairs of nodes from the Networks' topology using our proposed algorithm (*MRoute*) and aims to learn the reliability of individual links by their statistical measures of volatility over time. The algorithm maintains the routes' ranks based on their cumulative reliability and serves them on-demand in constant time, assuring the most reliable Routes. A full-fledged implementation of the KDN model as a test-bed to conduct experiments, which benchmarks *MRoute* with Diffusion Update Algorithm (DUAL) [51] and Shortest Path First (SPF) [52] that powers EIGRP as OSPF, respectively. Result confirms the validity of a constant time switch-over of Routes guaranteeing the highest reliability.

1.6. Motivation and Problem Statement

This section summarizes the motivation behind the research and presents a problem statement that it aims to address.

1.6.1. Motivation

As per the contemporary landscape of SDN and its applications are concerned, three broad elementary issues have contributed to the motivation of this research.

1. **Limitation of the SDN/SD-WAN solutions:** At the time this thesis is being compiled, the existing SDN/SD-WAN solutions are limited in their usage in the infrastructure orchestration. It offers more automation than control, i.e., SDN controllers are more often used to automate the underlying devices than offloading control plane functionalities.
2. **Centralized Routing Model:** The classic implementation routing protocol families (i.e., Distance Vector, Link-State or Path-Vector) is based on a distributed computing model, where the speaker nodes advertise their local view of the topology with their neighbours and flood any topology change via link-local multicast. However, this introduces a propagation delay for a relatively large topology which affects the convergence speed. As the contemporary Software-Defined Network model provides a centralized view of the topology, thus, running a distributed algorithm does not take advantage of the centralized computing model of the SDN. This includes Routing as a Service and out-of-band control with zero control packet exchange at the data plane during convergence. Therefore, a routing model dedicated to the SDN is a need.
3. **uRLLC Compliance:** The 5G specification introduces uRLLC as a KPI. Although several research in complying with the uRRLC has contributed to the data plane, there is a void for the same at the control plane. None of the existing routing models (to the best of our knowledge) consider statistical end-to-end reliability while calculating the routing metric. Also, routing as a reinforcement-learning problem is a novel concept.

1.6.2. Research Questions

This section summarizes the research questions and methodologies in two sections the "*Why?*" And the "*How?*", the former lists the legitimacy of the research problem and the latter outlines the key ideas for tackling them.

***The Why?* – problems this research addresses:**

1. How to provide rapid convergence in a dynamic network to achieve end-to-end Low-Latency communication.
2. How to facilitate network automation & programmability as an integral part of the design.
3. How to leverage Machine-Learning models to optimize routing decisions.
4. How to use both the Link (Communication) and Node (Computation) costs together to influence the routing decision.
5. How to use statistically evaluated end-to-end Reliability as a routing metric?

***The How?* – Methodologies to solve the above problems.**

1. Refactoring the Single-Source Shortest-Path-Problem.
2. Running routing protocol as a pluggable application module, i.e., Routing-as-a-Service.
3. A hybrid graph-search algorithm that pre-computes all possible paths between all pairs of nodes and maintains their order or preference in the runtime.
4. A robust telemetry protocol that feeds the network state to the application plane.
5. A Meta-Graph processing approach, that fuses multiple topology information (e.g., Neighborship, Flows, Utilization, etc.) into a single structure to perform centralized topology computation.
6. Using Recurrent Neural Network (RNN) with Long Short-Term Memory (LSTM) to model network state as an Auto-Regressive problem and compute anticipated reliability based on historical behaviour of the network.

Chapter Summary

This chapter introduces the concept of Network Intelligence in the view of the Industry 5.0 compliance. In the context of this thesis, the chapter discusses four major concepts that contribute to the development of a cognitive routing framework. Each Section trails the "contribution to the knowledge" of the concept. The chapter first gives a closer look into the Software-Defined WAN (SD-WAN) architecture with a use case of Content Distribution Networking (CDN). The Section elaborates on the need and architectural migration that the contemporary networks are going through which would need the SD-WAN implementation. Additionally, it includes a brief market survey on various SD-WAN technologies from renowned vendors. The next section puts a use case on integrating an SDN platform with a System-Level Simulator, which extends the discussion on the former concept towards a hybrid implementation where part of the network is Softwarized and the rest remains traditional. The section briefly discusses various simulation platform that exists in the literature and presents a generic view in the context of vehicular networks. Further, a specific use case of routing in such an environment is explored in the context of this Thesis which the next section explains in detail. Further, it presents a more subjective discussion on SDN routing and its fundamental differences from a traditional distributed model. The chapter concludes with a discussion on the cognitive routing for a Self-Organized Knowledge Defined Network leveraging the concepts discussed in the previous sections. It presents a detailed study on the same concept in the light of 5G and Industry 5.0 compliance.

In summary, this chapter sets the stage for discussion in the following chapters. It justifies the rationale, motivation and need for the research and its potential adaptability and compliance to the evolving networking technologies such as 5G, URLLC and Industry 5.0.

Chapter 2 Related work

Cognitive routing is a breed of routing algorithms where ML algorithms enhance the optimal route computation. The domain of cognitive routing is a constructive blend of several developments that have been progressing for quite a time now. The contemporary form of cognitive routing aims to comply with several other objectives; these include meeting the QoS requirements, compatibility with a Hybrid SDN architecture, compatibility with the Segment Routing techniques, and indeed, the application of deep learning framework. The following sections will touch upon each of the said areas to explore the motivations for cognitive routing as a whole.

2.1. QoS aware routing

Quality of Service is a feature on Layer 2 to 4 on the OSI protocol stack to prioritize traffic flows for streamlining them in congestion. A QoS framework ingests a policy set that describes a list of constraints. The framework uses tools to enforce the policy on the network, e.g., Classification, Marking, Queuing, Shaping, and Policing. The PDU header carries the marking information to tag a flow for prioritization. Table 1 depicts the marking field used in the PDU of each layer.

Layers	Marker	Bits
L3 (IP)	Type of Service (TOS) byte	8
L2.5 (MPLS)	Experimental (EXP) bits	3
L2 (Ethernet)	Class of Service (COS) bits	3

Table 1 QoS Marker on each layer of OSI

A class of routing algorithms has come out as extensive research in this field to support the constraints defined by QoS policies natively. Traditionally, the QoS policies are more stringent towards the delay constrained; hence, it is often referred to as delay-constrained least-cost (DCLC) algorithms.

With the advent of the SDN, centralized QoS enforcement is becoming an alternative to the classical distributed options[53][54][55]. The centralized management diminishes the need for complex control packet exchange between routers to determine an optimal path (e.g., RSVP); also, it provides a mature admission control mechanism to determine the Path before admitting a flow than eventually dropping it during policing at an intermediate router (e.g., DSCP).

2.1.1. The architecture of the QoS Routing framework

A QoS Routing framework has four major components listed in Table 2 and depicted in Figure 2[56]. The state model works as follows; the QoS routing algorithm receives a flow request. Now, the algorithm needs to decide if an optimal path exists at the current state of the network to accommodate the constraints. The network resource model supplies the expected delay based on the present network state. The cost function provides the boundary conditions as per the current network state. The Resource allocator examines the available resources (bandwidth, delay) and supplies the network state with the resource model.

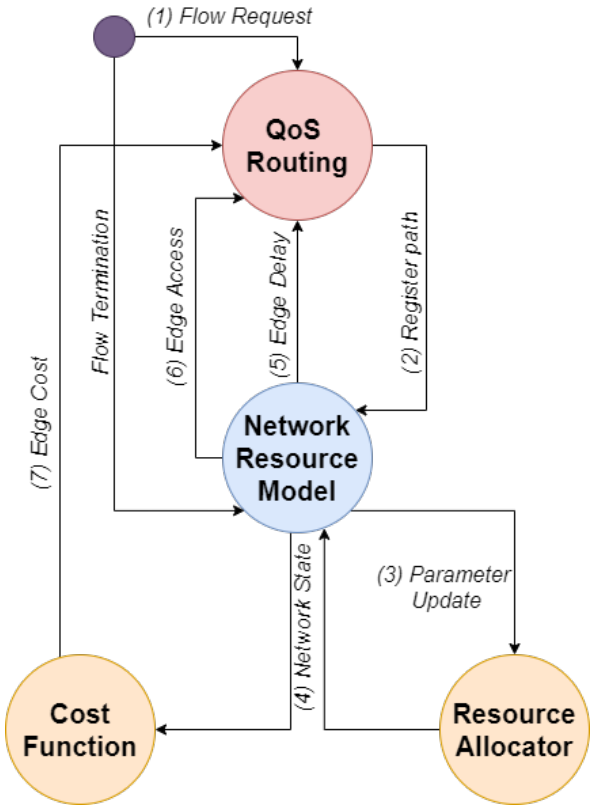

Figure 2 QoS Routing Framework

Module Name	Purpose
The Cost Function	The objective function that the QoS algorithm would optimize.
The Resource allocator	The module allocates bandwidth on different queues to maximize the flow accommodation.
Network Resource Model	A mathematical model that realizes the distribution function after analyzing the network traffic behaviour using stochastic network calculus. It monitors the traffic characteristics and updates the expected consumption.
QoS Routing	The routing module runs the optimization function over the QoS cost function, meeting the resource allocator's constraints and complying with the distribution realized by the network resource model.

Table 2 Component of a QoS routing framework

2.1.2. Fundamental QoS Routing problem

The mathematical formulation of the QoS Routing is a blend of Graph Theory, Queuing Theory, and Stochastic Network Calculus. Let's assume a Simple finite graph $G(V, E)$ represents the network topology, where vertex set $V = \{v_I\}$ represents the nodes, and the edge set $E = \{e_{I,j} \mid \mathrm{adj}(v_I, v_j)\}$ represents the links between nodes. The cost-vector $C = \{c_{I,j}\} \in \mathbb{R}_+^{|E|}$ is a positive-real vector of the cost of each edge $e_{I,j}$. Let there be m constraints defined, each with a boundary value $d_k$forming a positive-real vector $D \in \mathbb{R}_+^m$. Let P_{sd} denotes a set of feasible paths from a source s to a destination d. As every path is a member of the binary power set $\{0,1\}^{|E|}$Where a 1 denotes the subjected edge to be a part of the path. Hence, with |E|, all with $\mathbb{R}_+$ and m $\mathbb{R}_+$ constraints, the constraints matrix $M \in \mathbb{R}_+^{|E|\times m}$ And the formulation of the optimization model looks in Equation (1).

$$Z_{opt} = \min_{x \in P_{sd}} C^T x \quad s.t. Mx \leq D \qquad \textit{Eq. 1}$$

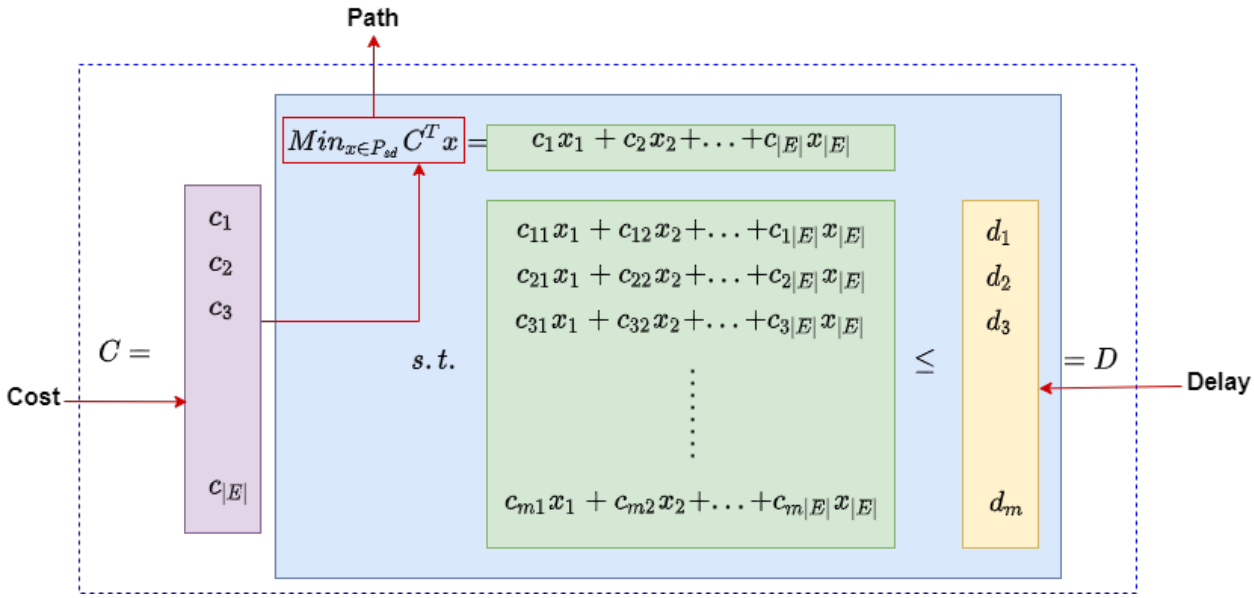

Figure 3 Mathematical framework of a QoS routing module.

Figure 3 depicts the visual representation of the formulation; recall the state model in Figure 2, the QoS Routing module takes the cost, and the delay as input selects an optimal path.

An optimization algorithm finds a solution z'; Equation (Eq. 2) measures the algorithms' efficiency or CI (Cost Inefficiency) in finding an optimal solution. As the objective of QoS routing algorithms is to minimize cost thus, any sub-optimal solution surpluses z_{opt}.

$$CI = \frac{z' - z_{opt}}{z_{opt}} \qquad \text{Eq. 2}$$

An optimization algorithm is said to be optimal if it always finds an optimal Path (P_{sd}). A Complete algorithm always finds a feasible Path if it exists; a Heuristic algorithm might find a sub-optimal path. Table 3 summarizes the different types of optimality and their relationship with CI.

Algorithm Type	**Cost Inefficiency**
Complete	$CI \in [0,1]$
Complete	$CI = 0$
Heuristic	$CI \in (0,1]$

Table 3 Summary of Optimization algorithm and their relationship with CI

2.1.3. Summary of QoS Routing algorithms

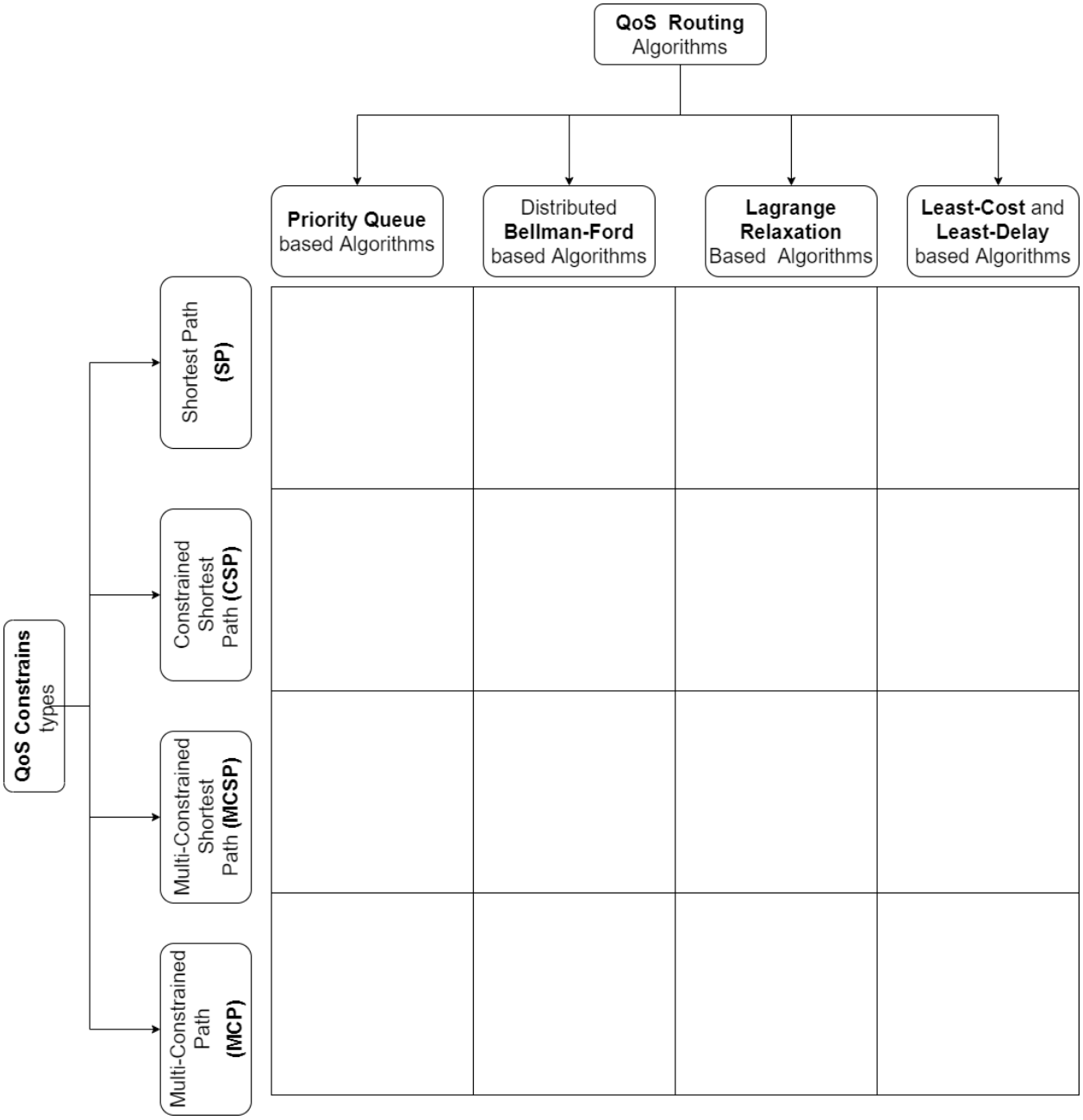

Figure 4 Classification of QoS aware routing models

A comprehensive survey by Guck *et al.* [57] has walked through all state-of-the-art QoS routing algorithms, evaluating their optimality completeness. Following is the list of algorithms that have been evaluated.

Mnemonic	Class	Optimal	Complete	Reference
SPF	PQ + SP			[52]
A*	PQ + SP			[58]
CHA	PQ + SP			[59]
A* Prune	PQ + CSP/MCSP	Yes	Yes	[60]
LDP	PQ + CSP		Yes	
FB	Elem + CSP/MCSP		Yes (CSP)	[61]
DBF	DBF + SP	Yes	Yes	[62]
YNA	DBF + SP	Yes	Yes	[63]
CBF	DBF + CSP	Yes	Yes	[64]
DCBF	DBF + CSP		Yes	[65]
DEB	DBF + CSP		Yes	[65]
LARAC	LR + CSP		Yes	[65],[66],[67]
LARACGC	LR + CSP		Yes	[68]
SCRC	LR + CSP	Yes	Yes	[69]
k-LARAC	LR + CSP		Yes	[65]
NR_DCLC	LR + CSP		Yes	[70]
DCCR	LR + CSP		Yes	[71]
(k)H_MCOP	LR + CSP/MCSP		Yes (CSP)	[72]
(E/MH)_MCOP	LR + CSP/MCSP	Yes (E)	Yes (E/MH) (CSP)	[73]
DCUR	LCLD + CSP		Yes	[74], [75]
DCR	LCLD + CSP		Yes	[76]
IAK	LCLD + CSP		Yes	[77]
SMS-RDM/CDP/PBO	LCLD + CSP		Yes	[78]
SF-DCLC	LCLD + CSP		Yes	[79]

Table 4: Lists of the popular QoS routing algorithms, their Type (optimal or complete), and class. PQ: Priority Queue, DBF: Distributed Bellman-Ford, LR: Lagrange Relaxation, LCDC: Low-cost Low-Delay.

Table 4 lists the popular QoS Routing algorithms along with their optimality. Notice that the Lagrange Relaxation-based algorithms are mostly not optimal but complete. We perform a qualitative comparison of the sub-optimal algorithms based on their Cost Inefficiency (CI) and Runtime Ratio (RR) based on the data available from the works of Guck *et al.*[80]. We fused the CI and RR factor to calculate an efficiency measure for ranking them using a simple rule $efficiency = \frac{100 - RR}{CI}$. As both CI and RR are measured in percentage and RR is proportional to the efficiency but lower is better, but CI is inverse. The result (Figure 5) shows that the k-LARAC is the most efficient method. However, it is a computationally intensive solution.

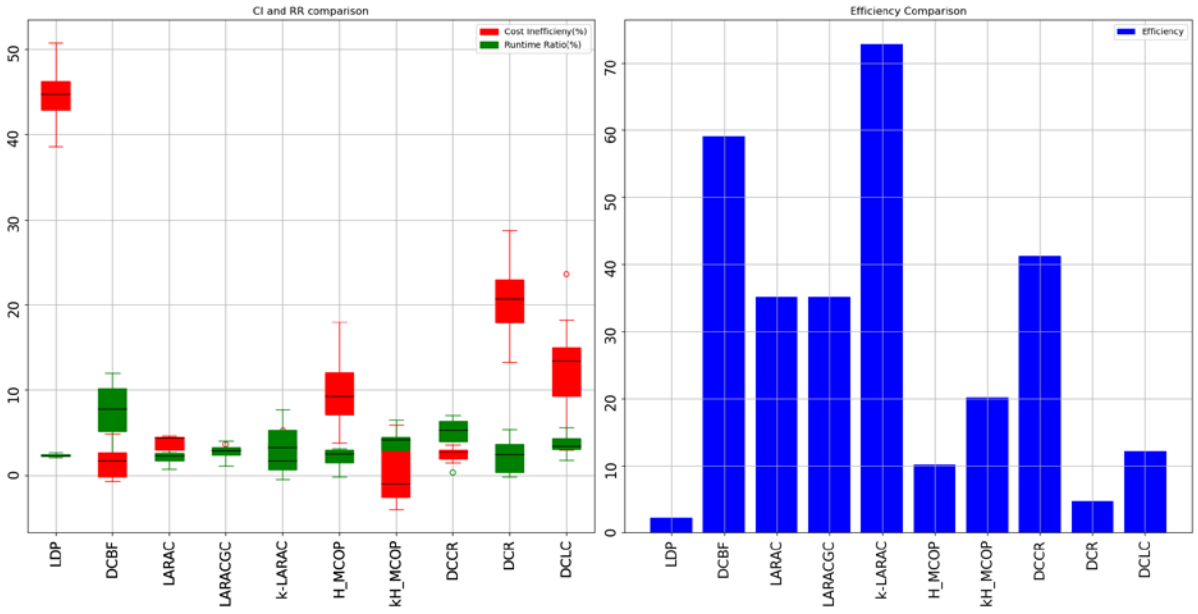

Figure 5 Comparison of the efficiency of the QoS routing algorithms

2.2. Hybrid SDN Architectures

Hybrid SDN is an intermediate state of a traditional network while it is transforming to become an SDN. There could be various influences for an enterprise to opt for an intermediate state. Amin *et al.* [81], in their survey on the Hybrid SDNs, outline several such reasons; this includes optimizing the reusability of existing devices, gradual training of technical staff for a stable migration, and cost optimization.

The domain of Hybrid SDN is vast; however, in the recent past, a considerable amount of research has shown the potential and relevance of the topic. Concerning this thesis, the hybrid SDN model has motivated me to design Software-Defined Self-Organization across an architecture-independent infrastructure plane. Several such influences have helped devise various configuration algorithms and fundamental design logic in the following chapters. The following subsections summarize the state-of-the-art of Hybrid-SDNs, relevant to the scope of the thesis (Figure *6*).

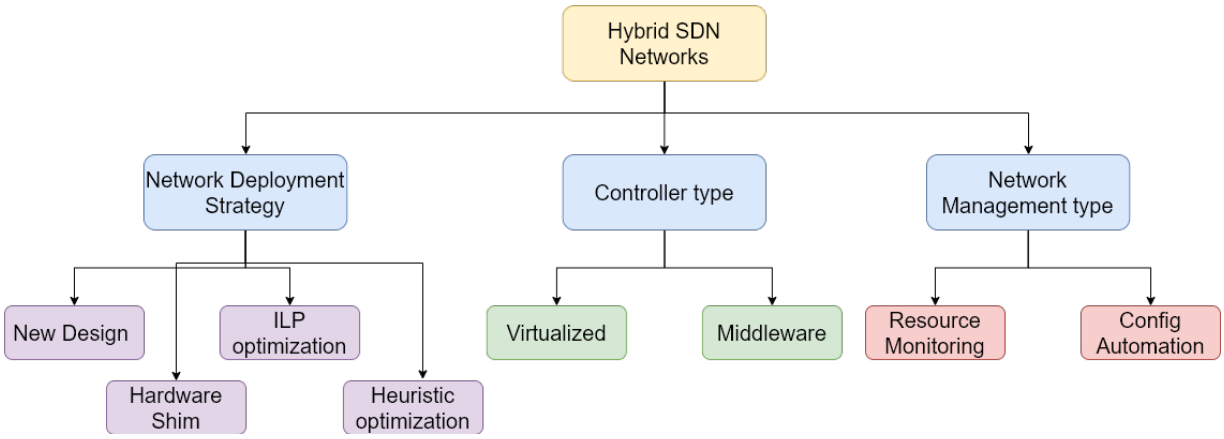

Figure 6 Hybrid SDN networks and their classification [82]

2.2.1. Deployment strategies

While transiting from a traditional, fully distributed network architecture, when an enterprise initiates Softwarization, it might opt for several deployment strategies. The choice is purely subject to the scale and requirements.

1. **Custom Architectural Design**: In this strategy, researchers propose bespoke flow management of a traditional network through an SDN controller. A virtual overlay network based on seminal panoptical design [83] suggests the incremental introduction of SDN switches among an existing legacy topology. All the packets must go through the SDN switches, where the switch builds a solitary confinement Tree to optimize the forwarding logic. However, topologies in a Datacenter (DC) environment need specialized treatment. Most of the DC topologies are of Spine-Leaf [84] in nature, where a cluster of spine switches acts as a single distributed switch, and each spine switch connects to all the leaf switches, resulting in a single hop traversal for any inter-leaf traffic. Softwarizing a DC network requires seamless scalability and load-balancing; thus, the migration strategy should not be abrupt but gradual. A common approach is to classify and segregate the programmable traffic (that flows through the SDN nodes) from the non-programmable traffic and apply traffic engineering to the programmable traffic to maximize the end-to-

end throughput [85]. Several other Softwarization techniques exist in the present literature that concern optical networks [86] [87], wireless mesh networks [88], and Satellite overlays [89]; however, these techniques fall beyond the scope of the thesis.

2. **Hardware Shim**: Unlike the previous model, the Hardware Shim [90] proposes a coexistence of SDN and legacy protocol processing modules within a switch kernel. A shim module resides within the switch that exchanges information between the SDN and Non-SDN networks. Both the CAM/TCAM and Flow-Table data structures share the switch memory and are used by the non-programmable and programmable traffics, respectively. However, this solution is proprietary to the vendors as, in general, they offer a monolithic or non-programmable kernel; thus, there are no commercial generic solutions exits as of the time of writing this thesis.
3. **Integer Linear Programming (ILP) based optimization**: The ILP optimization models the network into a graph and analytically finds the strategic migration. That said, these techniques are simulation-driven and hence do not always align with the economic feasibility. If performance upgrade is concerned, Softwarizing all nodes and applying a greedy algorithm-based traffic engineering would be the optimal choice [91]. However, it would also scrap a large amount of active hardware resulting in a significant CapEx. An alternative strategy is to calculate a Softwarized topology with minimum SDN hardware. An efficient dynamic programming-based algorithm [92] can process a topology of n nodes in $O(\log n)$ time, it minimizes the CapEx, but the end-to-end throughput stays sub-optimal. A sweet spot between the previous two strategies is to partition the network based on their traffic characteristics to put a dense SDN topology for QoS aware network and a sparse SDN topology for the rest [93][94].
4. **Heuristics-based Optimization:** Unlike the ILP optimization strategy, which optimizes the analytical model of the network, the Heuristic-based optimization is rather empirical

in nature. One might use this strategy when a network shows stochastic behaviour (i.e., the traffic characteristics are dynamic and not predetermined but realized by sampling). The controller accumulates telemetry from a large pool of network devices and decides a subset of the topology to Software preserving the configuration (e.g., translating forwarding logics of a legacy switch into flow entries in an OpenFlow switch). Hong *et al.* [95] use this strategy and show a 20% Softwarization reduces 32% control traffic. Xu *et al.* [96] apply the same method in combination with Depth First Search and Randomized Rounding techniques, which results in a 40% throughput gain.

2.2.2. Classification of Hybrid SDN controllers

SDN controller is the heart of SDN; it possesses a bird's eye view of the underlying network topology and enables a central management point. A cluster of SDN controllers uses so-called East-West APIs to establish an inter-controller fabric and offer scalability for large network topology. The controller cluster aggregates the topologies governed by each member controller and optimizes the unified graph collaboratively. The present literature shows several objectives that the controller might orient its optimality criteria. These criteria depend on the requirement and scale of the networks.

1. **Virtualized Controllers:** Their utilization varies in a resource-constrained network where the compute and network resources often don't change. In a hybrid-SDN, the controller unifies the resource management by translating configuration policies into data-plane-specific instructions (e.g., Flow entries for OpenFlow, Route-Maps for Cisco IOS devices). HybNET [97] offers a solution for the OpenStack neutron controller as a central control plane for SDN and legacy switches. It puts an abstraction layer between legacy and SDN networks to achieve a data-plane agnostic control plane. SYMPHONY [98] extends the scope of the control plane by integrating it into the legacy device. It uses the OSPF control

plane to communicate with the controller and maintains a global routing table unifying the SDN and legacy topologies; however, it lacks load-balancing. The SDN Hybrid Embedded Architecture (SHEAR) [99] deploys a small number of SDN nodes within the legacy topology. The SDN nodes become a programmable spine for the legacy leaf switches resulting in a high convergence. Telekinesis [99] spits the control plane by sending Flow instructions to SDN nodes and special packets to legacy nodes to update their CAM entries. The CAM override forces all traffic to pass through the SDN nodes; however, it only offers Routing and does not include TCAM-specific modification (e.g., ACL Filtering, QoS policy enforcement). Cardigan *et al.* [100] implement OpenFlow-based distributed routing by replacing all legacy devices with SDN nodes.

2. **The controller as Middleware:** In this strategy, the controller is a translator between SDN and Legacy configuration. ClosedFlow [101] offers SDN-like control on legacy networks with features like out-of-band management, topology discovery, ACL-based filtering, and event-driven packet processing; however, it lacks load balancing. Exodus [102] offers translation through an intermediate config format called pseudo-SDN rules and compiles them into OpenFlow and Cisco IOS instructions. LegacyFlow [103] bridges two non-SDN networks via an SDN fabric. The controller injects flow instructions to the SDN nodes, which connect the downstream non-SDN topology segment. The SDN nodes themselves are interconnected; thus, the Data-Place gets split into a Spine-Leaf topology with a central control plane on top.

2.2.3. Network management strategies

Network management is a closed-loop operation involving data accumulation from the infrastructure through a telemetry mechanism and injection of the configuration back to the infrastructure. The previous two sections describe the deployment strategies and the SDN

controller classification. The final bit of the story is network management, which describes how efficiently the network resources are managed. In the literature, several existing mechanisms are used by contemporary SDN controllers. However, resource management and self-configuration are the two major strategies that are relevant to this thesis.

1. **Resource Management:** Santosh *et al.* [104] propose a Weighted Fair Queue (WFQ) based programmable software-defined wireless LAN controller to provide load balancing, security policy enforcement, and QoS services. Unified Virtual Monitoring Function (SuVMF) [105] is proposed as a robust telemetry system for large hybrid SDNs. It offers conditional filtering, configuration transformation, and custom monitoring. Katov *et al.* [106] focus on usage consolidation and power consumption minimization in enterprise networks. It monitors the usage patterns of network devices and suppresses unnecessary nodes when the mean usage comes down. The results show a 45% drop in energy consumption with a 47% drop in link utilization. Seiber *et al.* propose a Network Service Abstraction Layer (NASL) [107] that unifies the control and data plane to optimize QoS policies for time-critical applications; additionally, vendor-agnostic programmability and monitoring [108].
2. **Configuration Automation:** Configuration automation releases the human factor from configuring network devices in a scalable infrastructure. It also removes the burden of configuration review before pushing it into production. Katiyar *et al.* [109] propose an external configurator using the DHCP-SDN model for Hybrid-SDNs. Martinez *et al.* [110] offer a semantic-based configuration of legacy devices called the Ontology-Based Information Extraction system. Mishra *et al.* [111] propose a decentralized approach, where each subnet of non-SDN nodes has at least one SDN node that collects the resource information periodically and injects forwarding logic as floating static routes. Amin *et al.* use a Graph Theory based incremental topology change detection mechanism [112] called Automatic Policy-violation Detection for topology Change (Auto-PDTC) [113] which

detects devices that need reconfiguration. Seiber *et al.* [114] use a Queuing theory-based analytical model for detecting devices that need reconfiguration in a partial SDN setup.

2.2.4. Summary

Figure 7 shows a comparative analysis of the hybrid SDNs discussed above. The evaluation counts the publications available in the literature leveraging the techniques and sums them based on six metrics: namely, traffic overhead, link utilization, number of SDN flows, number of path failures, deployment complexity and scalability. High traffic overhead and link utilization leave less bandwidth for the data traffic hence it acts adversely. Several SDN flows could be fixed as proactive flow entries or dynamic as reactive entries. The reactive flows provide dynamic manipulation based on altering network conditions, thus, it is a more robust fit. Path failures could be resilient which gives a quick failover or non-resilient where the network needs re-convergence. The deployment complexity measures the performance of a middlebox, i.e., an SDN box within a traditional network and the scalability measures the complexity in scaling the topology. Scoring each category based on the positives and negatives that the existing publications suggest, the study finds the optimal design would be a hybrid SDN model that deploys SDN nodes within the traditional infrastructure using ILP optimization with a virtualized controller and leverages automation to program the infrastructure plane.

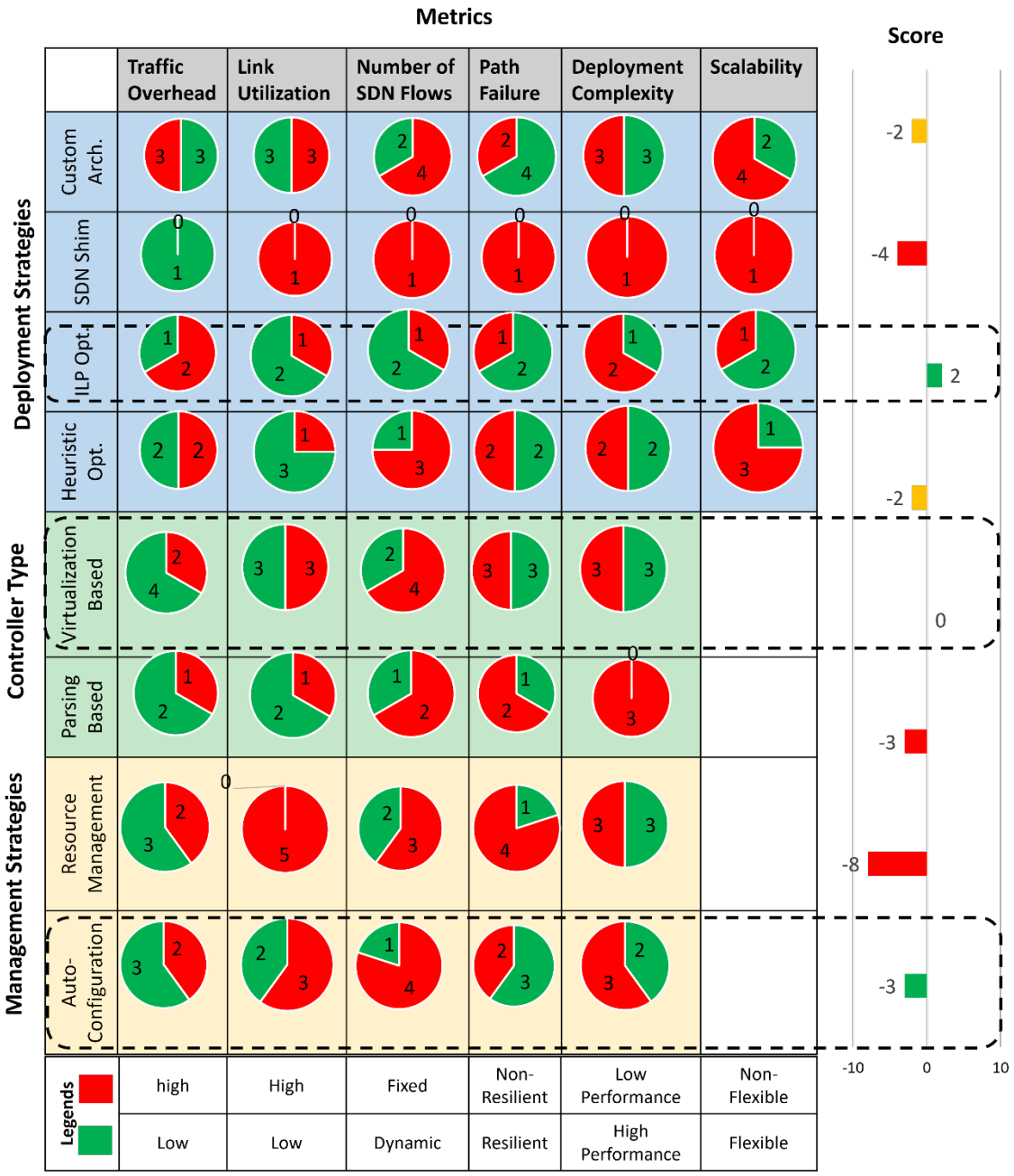

Figure 7 Performace analysis of various hybrid SDN designs based on six attributes. Each design apprach is scored agaisnt the available works in the literature. The result shows the most efficient design uses an ILP optimizer with a virtual controller that perform automated configuration.

2.3. Application of Machine Learning in Routing

The application of ML in solving contemporary SDN problems contributes to relaxing the computational complexity of traditional discrete optimization problems. Traditionally, discrete optimization methods such as Greedy, Dynamic, and Mixed Integer Programming are the de-facto choice for network optimization problems. A heuristic function drives the efficiency of the algorithms' convergence. However, formulating a heuristic is challenging in a dynamic network, where various characteristics vary over time. Moreover, the recent introduction of the Network-Slicing model requires specific treatment for selected traffic flows using Policies. Therefore, ML models have become quite popular to deal with network dynamics. A new layer called Knowledge-Plane (KP) sits on top of the classic three-layer SDN model that accumulates various network telemetry from the application plane, learns the behaviour from the historical dataset, and feeds the (near) optimal solution back to the application plane. The previous section describes the various controller functions in a hybrid SDN; ML-based optimization solves the issue of oscillation that otherwise be a common phenomenon in heuristic optimization. Oscillation happens when a slight change in the network state results in re-convergence. Recall the EIGRP metric; although the Load and Reliability parameters are present in the formulation, they are not used in production. Their fluctuation could force the underlying DUAL algorithm to recalculate the paths, resulting in an unstable network state. ML algorithms instead provide a more stable sub-optimal solution curved for the specific network behaviour.

In the literature, the SDN use-cases of the ML-based solutions exist [115] for: (A) traffic classification for QoS [116], (B) Routing optimization [117], (C) deep packet inspection, (D) resource management for QoS [118], and (E) malicious signature detection for security [119]. To be aligned with the theme of the discussion, we shall only concentrate on the aspect of Routing optimization and the respective development that exists in the literature.

The domain of Cognitive Routing is still undergoing its infancy. Although ML frameworks have helped accelerate many networking problems, routing is one of its recent application domains. The state of the art of cognitive routing solves the route-optimization problem broadly in two ways, State-prediction, and Route-matrix prediction. The following subsection presents the literature below.

2.3.1. Optimizing routes using state-prediction

The state prediction mechanism essentially sees the active Routes of network topology as states and tries to predict the optimal state given a source and destination using an RNN or RL or DRL. The earliest attempt by Yanjun *et al.* [120] uses a meta-learning approach, where an ANN is trained using the input and output of a heuristic algorithm. Eventually, the ANN models the hidden distribution that results in a real-time outcome bypassing the otherwise complex heuristic method. NeuRoute [121] uses RNN with LSTM to predict link utilization patterns to optimize the routes. A reinforcement learning approach by Sendra *et al.* [122] predicts the optimal path using the consequent network state variation as a penalty. For large-scale overlay deployment such as Datacenters, Francois *et al.* [123] leverage the Cognitive Routing Engine (CRE) [124]. The proposed model places the CRE within a logically central SDN controller that oversees the overlay fabric and runs a closed-loop control using RL. The QoS Aware Adaptive Routing (QAR) [125] uses RNN to predict QoS constraints compatibility of the links and determines the optimal paths for hierarchical SDNs. A more complex, however efficient Deep RL-based approach by Stampa *et al.* [126] finds the all-pair optimal path keeping the delay constrained checked.

2.3.2. Optimizing routes using traffic-matrix prediction

In this approach, the machine learning model predicts a given topology's hidden distribution of a varying cost matrix; therefore, the only feasible model is an RNN. Lopez *et al.* [127] propose a traffic prediction mechanism that forecasts the traffic pattern by estimating the trend. It creates proactive flow configuration for the data plane devices and injects them before congestion occurs. Alvizu *et al.* [128] propose a dynamic optical routing technique using metaheuristics. The algorithm has three phases, Offline Scheduling, Online training and Online Routing. Chen *et al.* [129] perform a multivariate evaluation for load-balancing that includes hop count, latency, packet loss, and bandwidth utilization. Azzouni *et al.* [130] propose *NeuTM*, which uses LSTM for the traffic matric prediction method.

2.3.3. Lessons Learned

After analyzing various ML methods applied in solving the SDN-Routing problem, we have concluded that RNN, Reinforcement Learning (RL), and Deep RL are the major techniques that suit the subjected problem class. The SDN Routing problem has primarily two categories, route prediction, and traffic prediction. Table 5 presents the applicability and efficiency of the three ML techniques in these categories.

	RNN	RL	DRL
Route Prediction	Fair	Fair	High
Traffic Prediction	High	X	X

Table 5 Usability of ML techniques in SDN routing

2.4. Self-Healing Technologies

Network outage due to overload or unprecedented failures is the greatest adversary for any network service provider. Contemporary network infrastructure relies on automation technology to battle any outage situation, preferably anticipating them before they occur. On average, the cellular network operators spend 23% - 26% of their annual revenue on managing the operation outages [131]. In 2015, the network outage cost $20B to the mobile network operators and service providers (MNO-SP) worldwide, 7% of their total revenue [132]. 3GPP poses the Self Organized Network (SON) as a solution that solves the problem of network inconsistencies and anomalies more structurally by addressing architectural changes. SON constitutes Self-Optimization (SO) [133] which provides autonomous optimization of performance [134], Self-Configuration (SC), which automates configuration of network nodes; and, Self-Healing [135], which identifies degradation of KPIs and heals it autonomously.

Research shows that for an MNO-SP to sustain itself, it can spend at most 1.7% of its revenue on NetOps without compromising service quality [136]. Modern network infrastructure has shown promising improvement after adapting the 3GPP recommendations on SON. Despite increasing traffic volume and network architecture complexity, it has demonstrated a trend in affected cell coverage during an outage. A key player is the Heterogenous Networks (HetNets) [137] which reduces the network density but increases the network parameters resulting in a better QoE.[138]

2.4.1. Self-Healing in Cellular Networks

Mainstream adaptation of SON begins from the 4G era. Self-Healing technology brings four key features; First, Autonomy, which makes function invocation independent of human input; second, Availability, which allows the network to scale; third, Adaptability, which absorbs the

external Influences and learns from internal failures and, fourth, Intelligence, which learns the network behaviour from historical data [139].

Prominent research on Self-Healing shows the impact of automation in various use cases. EUREKA[140] studies the wireless communication use-case, especially on UMTS and Wi-Fi networks, SOCRATES [141] studies LTE networks, QSON [142] studies SON coordination, and SEMAFOUR [143] studies self-management for heterogeneous RANs. The existing research outlines a Self-Healing framework that involves a network controller to automate network outage management. Further, the framework classifies any outage into either a full outage which results in a total network failure or a Partial-Outage, where the KPI degrades.

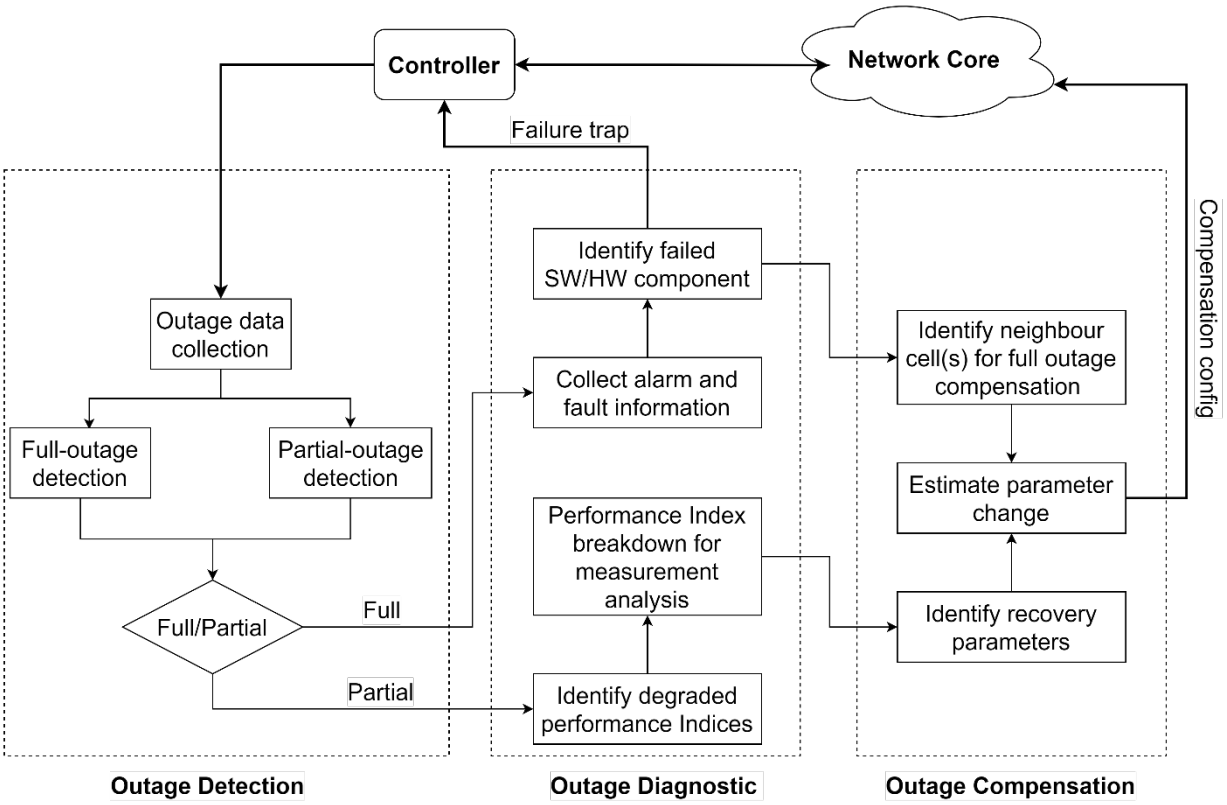

Figure 8 Closed-loop operation cycle of a generic Self-Healing framework

A Self-Healing framework consists of three stages; first, an Outage Detection algorithm determines a full or partial outage and the nodes that need further action. Second, a diagnostic Algorithm that detects the exact cause of the failure; third, a Compensation algorithm that

injects configuration changes based on the diagnostic information that the controller accumulates (Figure 8).

Components	Description and Classification
Methodology	*Techniques used to detect, diagnose and compensate for a failure.* 1. **Analytical**: Breaks down the mathematical model and optimizes. 1.1.Convex optimization 1.2.Non-Convex optimization 1.3.Genetic Algorithms 1.4.Simulated Annealing 1.5.Multi-objective optimization 1.6.Game theory 2. **Heuristic**: Predefined rules and prior knowledge 2.1.Rule-Based: Uses if-else rule 2.2.Framework based: consists of predefined guidelines 3. **Learning-Based**: Uses ML algorithms 3.1.Supervised (SL) 3.2.Unsupervised (UL) 3.3.Reinforcement (RL)
Topology	*Defines the logical structure of the networks* 1. **Homogenous**: One tier of macro-cells covering large areas. 2. **Heterogenous**: Multi-tier of macro and small cells or HetNet
Performance metrics	*Benchmark measurements to evaluate network performance.* 1. **Accessibility:** the ability of the users to use the network resources 2. **Retainability:** retain a session until it finishes without dropping it. 3. **Mobility:** Seamless handover 4. **Others:** RSRP, SINR, RSRQ
Control Mechanism	*Controlling the SON functions* 1. **Centralized**: CP is decoupled from the devices 2. **Distributed**: CP resided in devices 3. **Hybrid**: a combination of both
Control Direction	*Optimization of service link* 1. Node to user 2. User to node 3. Both

Table 6 Summary of the techniques used in Self-Healing techniques

Asghar *et al.* [136] present a comprehensive survey on the recent development in Self-Healing technology; Figure 8 depicts the various stages of a generic Self-Healing framework and its

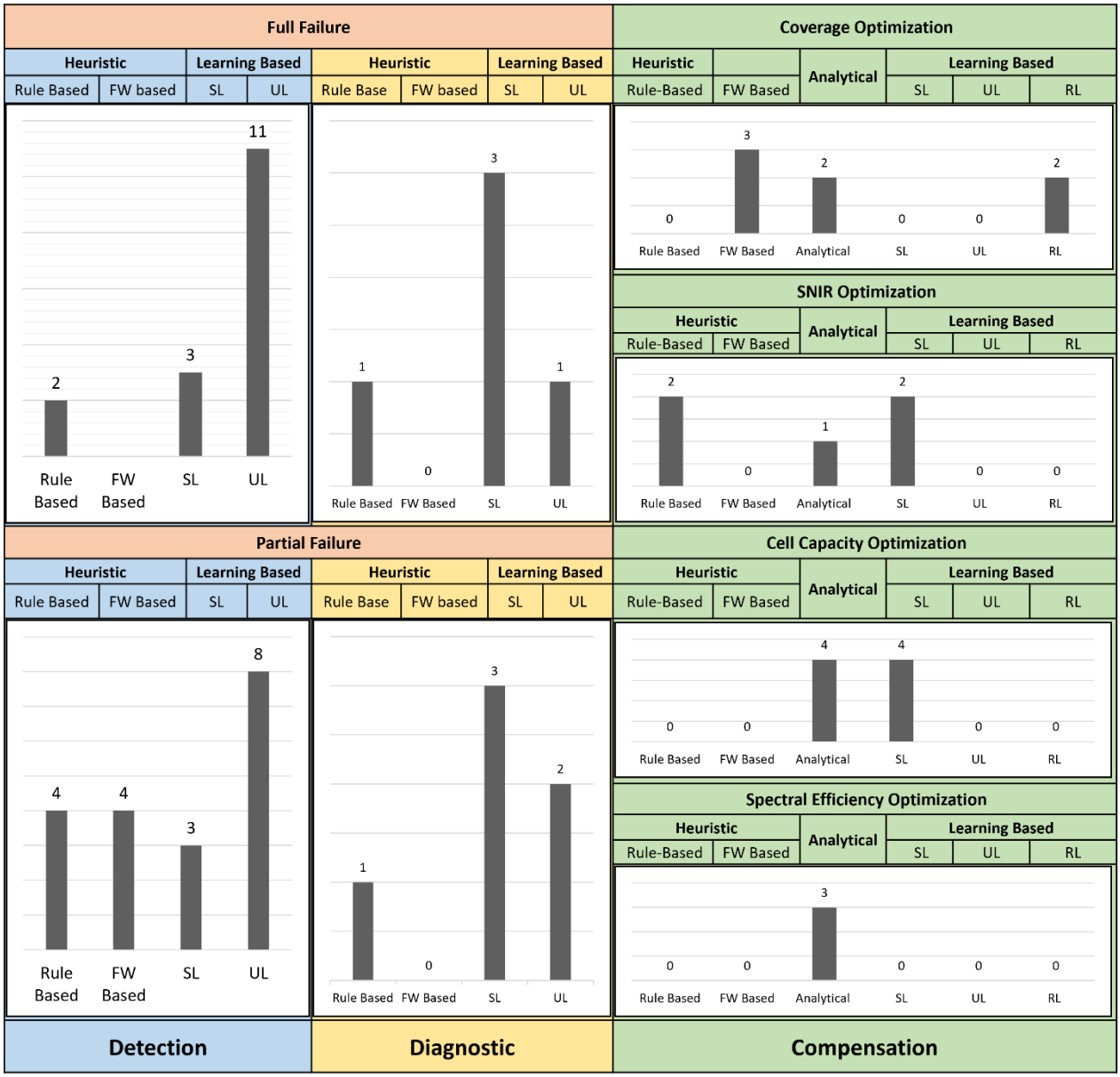

Figure 9 Summary of methodologies used in Self-Healing techniques for Cellular networks

closed-loop control flow, and Table *6* lists a summary of the techniques and taxonomy used in a Self-Healing framework. The current literature shows several works on Self-Healing techniques for cellular networks; Figure 9 summarizes the methodologies applied to Self-Healing techniques. Research shows the dominance of Supervised and Unsupervised learning algorithms for Failure detection and diagnostic problems, respectively. However, Analytical and Supervised learning show more usage in designing the compensation algorithms.

2.4.2. Service Migration based Self-Healing for MEC

The previous subsection describes the various Self-Healing techniques and methodologies used in Cellular networks. However, 5G and beyond technologies significantly leverage the MEC framework to reduce the network OpEx and improve QoS/QoE. A primary driving force has been the massive cloudification of resources. Enterprise networking has shown a steady trend in migrating shared resources to the public or private clouds while keeping the essential resources on-premises. In a distributed system, long-distance communication between remote resources via a cloud breach latency constraint may occur, especially in time-critical applications. MEC lowers the complexity by offloading computational and data resources to a closer cloud infrastructure (MEC node). Self-Healing in MEC architectures needs special treatment. In the literature concerning MEC orchestration, Service Migration has been a well-known technique that addresses the issue of autonomous healing of various failure scenarios.

Factors	Migration	Handover
Volume of transaction	Higher, as it transfers application runtime or memory images between edge servers.	Lower, as it transfers control and data traffic between cells.
Triggering factor	It happens when a device wants to offload or transfer tasks due to load-balancing or failover mitigation.	It happens when a device loses connectivity from its current cell or finds a better-quality cell.
Dependencies	Topology and protocol independent	Topology and protocol-dependent

Table 7 Difference between service migration and Cellular Handover

Service migration involves replicating a running service across the MEC cluster by minimizing the transaction overhead and downtime. There are two service migration models for MEC; first, the Live-Migration which migrates a live application if the host node fails or gets overloaded, and second is the Cellular Handover, which happens when most of the consumer migrates to the vicinity of a remote MEC node. However, the handover and migration have subtle differences (Table 7), which results in definitive treatment. For the sake of the context, this thesis excludes the Cellular Handover mechanisms.

Service Migration Frameworks for MEC

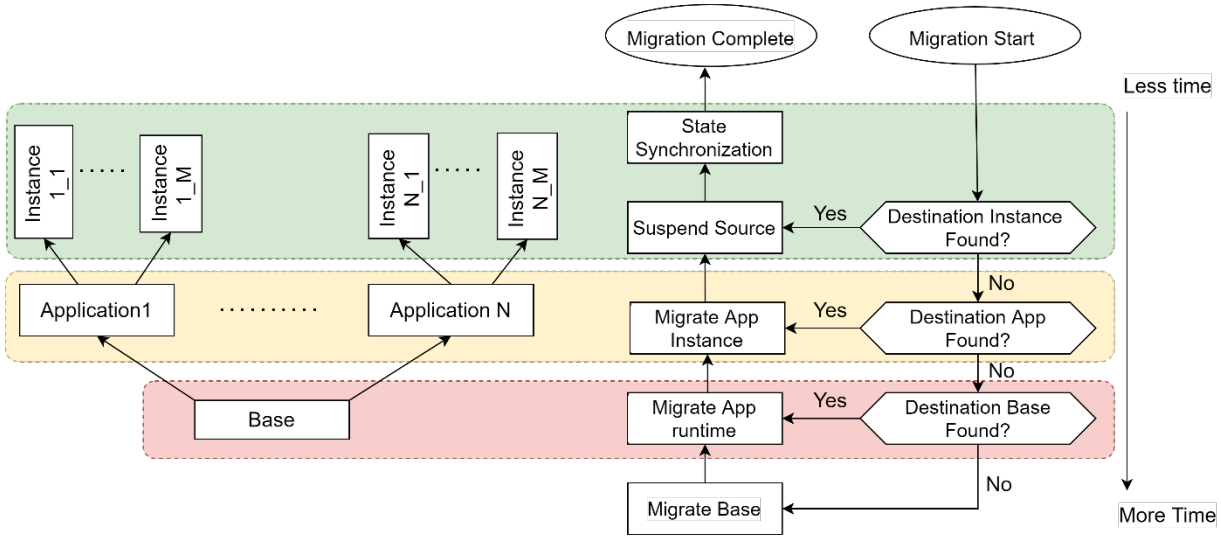

Figure 10 Holding state of an Edge server and the migration decision process

In the literature, there are two principal techniques

for approaching the problem of service migration. The first is the simple Three-Layer Framework (TLF)[144][145], and the second is a relatively complex Payload Optimization Framework (POF) [144][146]. While TLF focuses on transferring a live service between edge nodes, POF aims to optimize the transfer volume.

In the TLF approach, the Edge server has a Base, the OS kernel, which hosts several applications and their corresponding runtime (e.g., Python virtual environment); each application can run multiple instances. When migration is triggered, the source node queries a

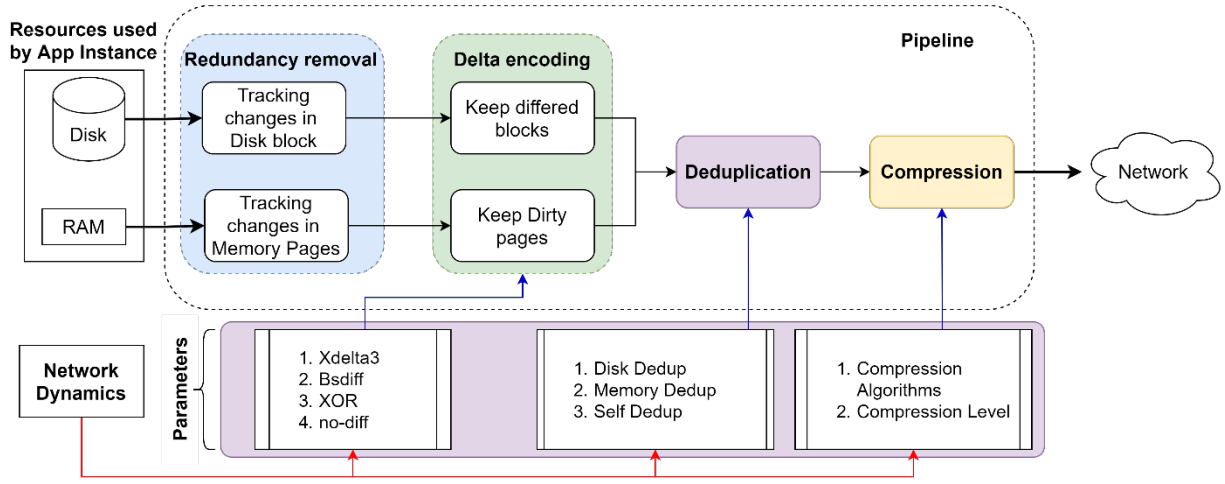

Figure 11 Pipeline stages of the Throughput Optimization framework

given destination regarding its current holding state and transfers the minimum information to restart the process at the remote end. Figure 10[147] depicts the system layout of the decision algorithm.

The POF approach is more complex than TLF; it uses a pipeline mechanism of several stages to optimize the payload size; hence, resulting in a compute-intensive but a space-efficient solution. Figure 11 [147] depicts the pipeline stages in POF; it enhances the TLF concept by inheriting the principle idea of three-layer segregation of a system but optimizes the traffic load. Unlike the TLF approach, where each application instance migrates sequentially, POF parallelizes them using the pipeline. Additionally, POF adapts dynamic network behaviours in choosing various parameters for pipeline stages. There are two Dynamic-Adaptation strategies; a simple Bottleneck-finding strategy that chooses parameters based on end-to-end throughput availability and a more advanced Heuristic adaptation of the former approach.

The bottleneck throughput on a migration path (s, d) *i*s the chocking point, where the forwarding cost is maximum. If BW_i *i*s the Effective bandwidth, p_I is the processing time and $r_i = \frac{Egress\ traffic}{Ingress\ traffic}$ of a node i at the path (s, d) then the system throughput TP_{sys} of the node are the minimum of the processing and network throughput expressed as equation Eq. 3

$$TP_{sys} = min(TP_{Proc}, TP_{net}) = min\left(\frac{1}{\sum_{i\in(s,d)} p_i}, \frac{\min\limits_{i\in(s,d)} BW_i}{\prod_{i\in(s,d)} r_i}\right) \quad \text{Eq. 3}$$

The heuristic extension of the former method works on a sliding time window $[1 - t]$ of t time instance, that enhances the system throughput by scaling its components.

1. Let $P_t = \{p_i \mid i = [1 - t]\},\ R_t = \{r_i \mid i = [1 - t]\}$
2. Measure P_t and R_t periodically
3. Calculate average values P_m and R_m at which the migration probability is above a margin.
4. Calculate scales $S_p = \frac{P_t}{P_m}$ and $S_r = \frac{R_t}{R_m}$
5. Calculate a heuristic of the system throughput (HTP) (Eq. 4)

$$\begin{aligned} HTP &= min(S_t TP_{proc}, S_r TP_{Net}) \\ &= min\left(\frac{s_t}{\sum_{i\in(s,d)} p_i}, \frac{s_r \times \min\limits_{i\in(s,d)} BW_i}{\prod_{i\in(s,d)} r_i}\right) \end{aligned} \quad \text{Eq. 4}$$

6. Select a node as the destination that maximizes the HTP. (Eq. 5)

$$\max_{j\in Nodes}\left[min\left(\frac{s_t}{\sum_{i\in(s,d)} p_i}, \frac{s_r \times \min\limits_{i\in(s,d)} BW_i}{\prod_{i\in(s,d)} r_i}\right)\right] \quad \text{Eq. 5}$$

Migration Strategies

The current literature solves the service migration problem in MEC using three major techniques. First, Follow Me Cloud (FMC), second, Markov Decision Process-based Migration (MDPM), and third, Time Window-based Migration (TWM).

The FMC prototype and its variants [148][149][150][151] migrate services across federated DC nodes (Edge Servers), where the service gets replicated or migrated as the users move. In other words, the cloud service follows the user density, and when it drops below a cutoff, the service expires from its host. The QoS parameters such as Cost, and Delay governs the migration decision.

The MDPM models use stochastic analysis of the users' movement to decide on a migration. MDPM models come in two flavours. A unidimensional MDP (1D-MDP) [151][152] model considers the users' movement along a straight line (e.g., a car on the road). The Edge server decides migration based on the distance of a user consuming a service from the Edge server which hosts the service. A more capable model is the two-dimensional MDP (2D-MDP) [148][153] models user mobility on a 2D surface (e.g., a drone) which is more suitable for Cellular networks.

The TWM strategy searches for an optimal placement sequence of multiple services that minimizes the average cost over a given time window [154][155]. Using time-series prediction methods such as Auto-Regression and Recurrent Neural Networks, an extension to the TWM calculates a Look-Ahead Window[154], which helps the Edge server to take proactive migration decisions based on historical data. Unlike MDPM, TWM can accommodate heterogeneous cost function, network topology, and mobility patterns. It also poses a less complex solution as, unlike MDPM, it suppresses the probability distribution function discovery.

2.4.3. Contemporary Service Migration using Distributed Ledge Technologies

The 5GPPP consortium discusses the potential usage of Softwarization and Orchestration for efficient 5G service management. The three enabling technologies for it are multi-Tenancy support, Cloud and virtualization, and network programmability. The same white paper

document also talks about Network Slicing that bundles the said technology enablers and applies them to the use cases with optimal configuration. Each slice gets its use case-specific Virtual Network Function (VNF) optimized for the service it carries out. Eventually, it offers greater manageability, abstraction, isolation, and throughput. Focus *et al.* [156] in their paper on challenges in Network Slicing in 5G, point out the lack of adaptivity in the present management and orchestration (MANO) framework. It suggests that introducing a slice management plane that analyses the current state and predicts the future State can perform better slice allocation in efficiency and scalability. Efficient architectural alternatives for the infrastructure layer of the network slice architecture exist in the literature. Two of the most efficient are software Defined Network (SDN) and Mobile Edge Computing (MEC). SDN decouples the control and data plane and establishes a centralized control mechanism. It offers better manageability and dynamic control. Haleplidis *et al.*[39] in the RFC 7426 discusses the layered architecture of SDN and its functionalities. MEC, unlike SDN, decentralizes the computational capabilities to the edge of the network, making them closer to the end IoT devices.

MEC significantly reduces the networking between end devices and the cloud, as the most frequently invoked edge nodes execute tasks. Eventually, it reduces response time and enhances efficiency. In his survey on MEC, Abbas *et al.* [157] discuss various components, application areas, and research challenges. The same article also highlights transparent application migration as one of the open issues. Yassin *et al.* [158], in their report on SDN and IoT, present several SDN enabling solutions for IoT, such as IoT protocols, IoT Operating systems, development platforms etc. This provides adequate information to build an SDN-enabled IoT infrastructure. Virtualization plays a significant role in the network function layer of Network Slicing architecture. Although traditional virtualization offers an enormous benefit

in terms of resource utilization for the VNFs, it lacks when it comes to migration. Containers, on the other hand, for their lightness, show better performance on migration.

Tay *et al.* [159], in their article comparing VM (live migration) and Container (kill/Restart migration), concludes Containers lead the stateless migration and VMs lead the state-full. Also, the VM migration is more complex and requires specialized configuration such as Shared Storage (NAS) and a dedicated network interface, which makes it inapplicable for fully distributed IoT infrastructure. Addressing all the issues highlighted in this section, Chapter 5 presents an Intelligent IoT solution for accommodating heterogeneous IoT (H-IoT) under a multi-access mobile edge computing (MA-MEC) platform powered by SDN. A tool called Shell-Mon orchestrates Transparent Application Migration by running apps in Docker Containers. The proposed system is also resource-aware, i.e., it periodically records and distributes local resource state information across other nodes. Also, it decides the best suitable time and destination of migration. Here we state some of the works that are solving the same problem domain. Qiu *et al.*[160] in their works shows an LXE based container migration using a tool called Checkpoint and Restore (CRIU) [161] that records checkpoints of a running docker container and can be used to restore. The article uses multipath TCP communication to transfer the container. Dupont *et al.* [162], in their writing, also used LXE container migration in 2 dimensions, horizontal (roaming) and vertical (offloading), putting flexibility as one of their future aspects. Nadgowda *et al.* [161] introduce a migration system called Voyager that uses CRIU on Docker containers. It does In-Memory state migration and local file system migration through in-band (data federation) and out-of-band (lazy replication) transfer techniques. The IoT architecture presented in Chapter 5 stands out differently from the current literature's present solutions utilizing its intelligent nature. The proposed system does not need a shared memory architecture hence can be fully distributed. It uses CRIU for checkpointing local container state but also monitor other IoT nodes' resource utilization and maintains them

in a Resource Information Database (RIDB). Based on resource utilization trends, it can predict the future state. The behavioural analysis of resource utilization can automatically select the victim container to be migrated and destination too. This decision-making ability enhances self-reliance, scalability, and less pre-configuration overhead. SDN can also empower load-based priority configuration for different V-Switch ports for better Quality of Service (QoS).

Blockchain has been in the limelight for a while as the backbone of one of the revolutionary concepts of the present era of Cryptocurrency. From its first inception [163], potential utilization and use cases have been famous in the research community [164] [165] [166]. One such area is Multi-Operator Network (MoN) and Small Cell (SC), which coincide with the interest of this paper. Backman [167] shows signalling enhancement on the traditional 5G network stack to enable on-demand resource allocation. BC uses Slice Leasing Ledger and Smart contracts to append a trust layer brings two advantages. First, it reduces the coordination and transaction cost by negotiating with a trusted party to undergo an automatic agreement. Second, it diminishes the issue of Single Point of failure by offering distributed architecture hence, the case of DoS attack. The effective use of Smart Contracts for implementing Service Level Agreement (SLA) [168] [169] with various QoS parameters, implementing the decentralized application over the multi-admin domain [170] with three scenarios, software-defined Wide Area Network (SD-WAN), NFV, IaaS, and Network Slicing. Network Slicing also leverages efficient spectrum micro licensing in an Ultra-dense Small-Cell Radio Aria Network (RAN) [171]. Also, a caching/offloading model for blockchain-based MEC by Liu *et al.*[172] that uses a caching algorithm with computational offloading.

2.5. Self-Organization, a beyond 5G perspective

Fifth Generation Mobile technology (5G) has evolved after several years of R&D focused on systems beyond 4G. Similarly, in a similar situation, the ICT industry faces challenges

regarding Systems beyond 5G and the innovation in sixth-generation (6G) technologies. Leading companies and research communities focus on evolving 5G and completely new technologies [173], capabilities, and solutions that will be unique to the market beyond 5G [173]. R&D considerations and the ICT ecosystem include new B5G market-driven business models and opportunities and societal factors, such as the United Nations Sustainable Development Goals. Additionally, the 6G market will also focus on filling 5G capability gaps.

Industry 5.0 requires a context-aware personalized interface between machines and humans with the so-called *'human touch.'* A prime enabler for such services is using on-the-fly contextual profiling and data analytics using Artificial Intelligence (AI) and Machine Learning (ML) techniques. We can anticipate that the number of generated data will grow exponentially over the following years. The vast volume of data can help improve the performance of AI, in particular deep learning. Improved AI drives deeper user engagement and will, in turn, generate even more valuable data. However, in the conventional cloud/client model, the link between the centralized cloud and the end-users has increasingly become a bottleneck, as the virtuous cycle continues to drive ever more data and increasingly lower latency services and applications.

Moreover, the Edge-Cloud has provided offload and caching capabilities to devices connected to the co-located access point. To reduce the Total Cost of Ownership (TCO), AI is needed to push on the edge, allowing innovation and open edge services to partners and developers to create applications that support consumers, enterprises, and multiple verticals while adding significant value to their business. AI can handle many network functionalities such as managing interference, optimizing VNF placement on the edge, detecting DDoS, maintaining QoE of the applications, and making optimal local decisions. Therefore, distributed intelligence at the edge network can be a real differentiation in 6G. Additionally, AI

will be the key to automating/optimizing E2E application provisioning in such a complex network by taking inputs from end-to-end and across protocol layers.

In Networking, Dynamic Routing Protocols are essentially distributed algorithms running across an IP-Network. Each instance runs an identical Shortest Path Algorithm and shares the locally available information with its peers. The process in which the local information gets propagated classifies the protocol family into two groups. In the Distance Vector Routing (DVR), often called "*Routing by Rumors*," routers only know what their neighbours tell. Therefore, routing protocols such as RIP [174], and EIGRP [24], although space-efficient but prone to have routing loops.

On the contrary, in Link-State Routing (LSR), often called "*Routing by Propaganda*," each router is aware of the entire network topology, thus never falling into Routing Loops; however, maintaining a large topology table. Protocols like OSPF [23] manage this problem by segregating the network into multiple Areas; also, it defines various types of Link State Advertisement (LSA) types and limits them to get flooded within certain Areas. The recent time has evident a tremendous escalation in the network scale both in the structural and operational complexity [175]. Traditional routing protocols are not adequately capable of dealing with such complexity, which results in a slower convergence that eventually affects the network's end-to-end performance. One of the sole reasons is that routing protocols are inherently distributed; thus, they rely on the underlying communication systems to exchange information. As the communication speed is far inferior to modern processors, it becomes a bottleneck. To address this problem, Software-Defined Network [39] decouples the Control-Plane (CP) and keeps it in a logically centralized location, keeping the Data Plane (DP) distributed. In this architecture, network devices only forward traffic, whereas all the control functionalities execute centrally at CP. The CP sees the underlying network from a Bird’s-eye-

view like the Link State Request (LSR) model but does not replicate them to individual routers, diminishing the communication bottleneck.

SDN's new paradigm in networking has brought significant industry appreciation; various network models and protocols have been developed in recent times to leverage Softwarization into mainstream networking. Some of the instances are NG-SDN from Open Networking Foundation (ONF) [176], Cisco-Viptella SDWAN [19], SD-Access [177], VMWare NSX [178], 5G-PPP software networks [179], Disaggregated platforms like ONIE [180], ONL [181], SAI [182], SONiC [183] form Open Compute Project(OCP). It is prominent that traditional routing protocols are not a fit for these modern networking models, primarily because they appear to run a distributed algorithm in a centralized computing model, which has motivated the research community to develop SDN-specific routing algorithms. With a more flexible, programmable, and manageable networking model, two of the prominent use cases of SDN-Routing have surfaced in recent times, namely Segment Routing [184] and QoS Routing [57]. The former leverages Multi-Protocol Label Switching (MPLS) [185] based communication at the underlay and replaces Label Discovery Protocol (LDP) [186] and Resource Reservation Protocols (RSVP) [187] in CP. Quality of Service (QoS) Routing steers traffic to an optimal path preserving various communication constraints.

Self-Organized Networks (SON), an adaptive, autonomous, and scalable Network-model is becoming the norm in designing massive network architectures [188] such as Data-Centers (DC), Internet-Service Providers networks (ISP), and large-scale Enterprise Networks. The SON for beyond 5G networks offers Self-Learning ability, i.e., building self-awareness of the underlying network behaviours and characteristics, i.e., Self-Optimization, Self-Configuration, and Self-Healing. In SDN Routing, the Shortest-Path calculation is the optimization subject 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. However, the application of ML in route-optimization is a relatively new domain; at the time of writing, a handful of works are available to develop an intelligent routing algorithm for SDN. The base model of fitting ML in SDN is known as Knowledge-Defined Networks (KDN) [22]. The primary objective is to accumulate holistic information from a supervising CP of an underlying IP network and analyze them to extract knowledge generalizing network behaviour. This knowledge eventually helps bypass the need to use costly heuristic Routing algorithms, having preserved the equal adaptation capabilities to network dynamics [45].

Routing in 6G is likely to use KDN, where the primary objective is to accumulate holistic information from a supervising CP of an underlying network to offer SON capabilities: analyze them to extract knowledge that generalizes the network behaviour. This knowledge eventually helps bypass the need to use costly heuristic routing algorithms, preserving equal adaptation capabilities to network dynamics.

6G will provide SON with the following characteristics: Self-Learning ability, i.e., building self-awareness of the underlying network behaviours and characteristics, i.e., Self-Optimization, Self-Configuration, and Self-Healing.

Chapter 3 explains the mechanism of proposes the Most-Reliable-Route-First (MRRF), an Intelligent Routing algorithm for Self-Organized Knowledge-Defined Networks. The proposed model initially calculates all possible paths for all pairs of nodes from the Networks' topology using our proposed algorithm (*MRoute*) and aims to learn the reliability of individual links by their statistical measures of volatility over time. The algorithm maintains the routes' ranks based on their cumulative reliability and serves them on-demand in constant time, 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 *MRoute* with Diffusion Update Algorithm (DUAL) [51] and Shortest Path First (SPF) [52] that powers EIGRP as OSPF, respectively. Result confirms the validity of a constant time switch-over of routes guaranteeing the highest reliability.

2.5.1. Towards industry 5.0 architecture and beyond 5G compliance

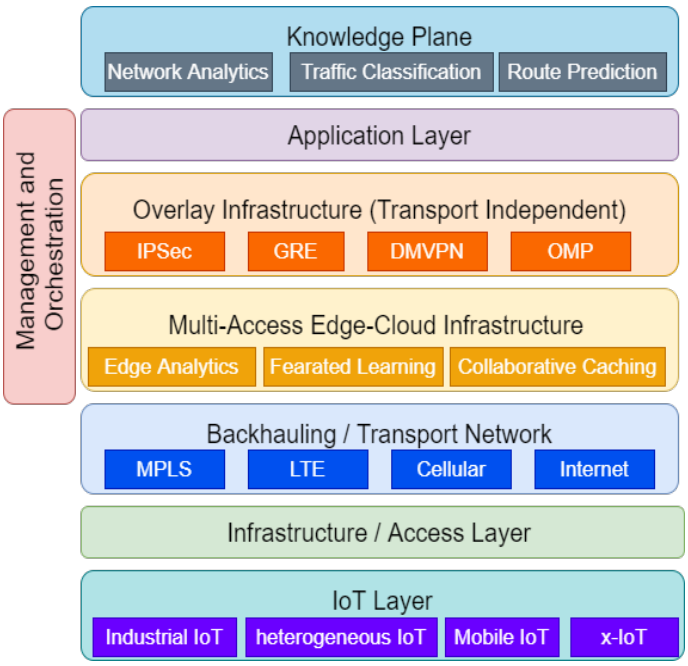

Figure 12 Protocol stack of a self-organized knowledge-defined network

Architectural evolution & SO-KDN Protocol Stack

5G beyond technology aims to design and develop a scalable architecture compliant with Open-RAN [189] and ng-SDN [190] (Figure 12), handling a massive amount of data generated at the edge supporting different verticals (e.g., surveillance, public safety, autonomous vehicles, etc.). Ubiquitous connectivity at the Edge (edge cloud, backhauling), Big Data management, and AI/ML form a virtuous cycle for next-generation connected computing. The combination of AI and an optimized edge architecture can reduce cloud data and backhauling costs. The introduction of federated learning can reduce the complexity of AI/ML engines when they are deployed in a distributed environment. Additionally, cellular operators can provide a platform for innovation and open edge services to relevant partners, allowing them to develop and create

applications that support consumers, enterprises, and multiple verticals while adding significant value to their business. 6G architectures foresee facilitate the following characteristics:

- **Ultra-high data requirements at the edge using new radio architectures:** Cell-free massive MIMO networking can potentially resolve many of the interference issues that plague current cellular networks. In cell-free mMIMO networks, cooperative signal processing is usually prohibitively costly due to the considerable amount of data, which may involve a high computational cost of joint processing. Full-scale cooperation also requires the estimation of channel coefficients from all devices to all radio units (RUs), or access points (APs), resulting in significant channel estimation overhead, thus fundamentally limiting the gain achieved through cooperation.
- **Automation**: AI/ML is vital for making decisions across different layers, from physical to application. MEC can support AI processing, decomposing applications in processes that run in parallel at any location, including vehicles, drones, or machinery, and collect and process data from smart devices as they consume or generate data. Additionally, local AI may need to update ML models from the collected data.
- **Multi-Tier Edge Cloud Architecture:** 6G aims to design a multi-tier edge cloud architecture allowing the establishment of collaboration among edge-cloud instances. Such tiers are essential to optimize the different types of caching within the multi-tier architecture (e.g., Video caching, Face recognition, Positioning) and apply different types of caching policing by considering the multi-tier level caching popularity prediction and backhauling costs, and capabilities.
- **Application Decomposition:** Both academia and industry have focused on Edge Computing by providing software (e.g., Google Lite TensorFlow) and hardware (e.g., NVIDIA Jetson AGX Xavier, AWS DeepLens) solutions suitable for edge processing. The

latest trend is decomposing QoS demanding AI-based applications to be deployed across heterogeneous distributed edge cloud infrastructure. It aims to design and develop a decomposed model considering each decomposed service's computation requirements and QoS constraints.

- **Overlay Plane:** To achieve transport-agnostic design, 6G communication provides dynamic end-to-end Tunnelling technologies (e.g., IPSec[3] over DMVPN[4] and OMP[5]). It allows Enterprise and service providers to use any transport mode of their choice (e.g., MPLS, LTE, 5G, and Internet), deploying a virtual network on top. The Overlay-plane provides a platform to virtualize, manage and orchestrate a physical network. Additionally, it abstracts the data-plane infrastructure to the control plane and presents a unified interface to automate and configure policies on it.
- **Big Data Management:** The powerful data processing and analytics capabilities traditionally lived in the heart of the centralized data centre must be strategically placed closer and closer to the data generating and consuming endpoints. By expanding the powerful capabilities of the edge data centre, service and network providers can deliver more effective services, reduce application latency by processing more data closer to the edge, and optimize TCO.

Knowledge -Defined Self-Organization in beyond 5G

The 6G Self-Organized Network inherits the Self-Optimization, Self-Configuration, and Self-Healing from its predecessor 5G, which also includes Self-Learning. It shows an apparent convergence of KDN and SON to achieve this. Centralized policy-based routing with SDN

[3] Security Architecture for the Internet Protocol RFC (https://bit.ly/3h6PTYA)
[4] Dynamic Multipoint VPN Configuration Guide (https://bit.ly/3ttN4Wv)
[5] Cisco-Viptella Overlay Management Protocol (https://bit.ly/3yW5sZ3)

accomplishes the Self-Optimization, Self-Configuration is leveraged by Network automation and programmability and rapid convergence provides Self-Healing and route-reliability prediction using deep-Leaning make the self-learning possible. It results in an intelligent network architecture that inherits all benefits of SDN and extends its capability with centralized intelligence. In this research work, we have tried to exploit SDN-routing as an optimization problem. The KP accumulates the network behaviour and predicts the most reliable route, which the network can converge in a constant time. The rapid switch-over guarantees Self-Healing, having met the URLLC criteria.

2.5.2. State of the art in KDN

The inception of the KDN comes from Clark *et al.* [44], who proposes a unified KP that takes decisions based on partial and conflicting information accumulated from a distributed cognitive framework. KP has been considered in solving the Optimal Route-Preference problem by learning network behaviour over time. However, the article lacks real-world network types such as ISP, Enterprise, Cellular, etc., and does not include working principles in a heterogeneous network. These issues are addressed by Strassner et. al.[191] by their extension of KDN with an Interface-Plane, which offers a clearer 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, aiming to achieve the KDN. Fadlullah *et al.* present a classification of various ML/DL algorithms and their application to intelligent network traffic control systems [50]. Chen *et al.* focus on the application of DL into several cognitive wireless communication systems such as the Internet of Things (IoT), Multi-Access Edge Computing (MEC), Unmanned Aerial Vehicle (UAV) networks, etc. [192]. Zhao *et al.* [193] review the specific applications of ML to SDN problems such as defence mechanisms against Distributed Denial of Service (DDoS) attacks, 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 (HA) are the two widely used approaches that solve routing-optimization problems [194]. Among several alternatives, Artificial Neural Networks (ANN), Reinforcement Learning (RL), Deep RL (DRL), and Lazy Learning (LL) are the four learning models primarily used to address Routing Optimization. Yanjun *et al.* [120] propose an ML-Meta later-based approach where an ML model is trained by the calculated traffic parameters of a heuristic algorithm and its corresponding network state as input. The proposed framework maps the input and output of the HA that reduces its exponential run-time to a constant one. NeoRoute [195] 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. for high-density WANs [127]. The previous research papers 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. Sandra *et al.* [196] propose a DRL framework, which trains 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 [123] *et al.* apply DRL with a Random Neural Network 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 [125].

2.6. Chapter Summary

The goal of this chapter is to outline the state-of-the-art of the various disciplines in the current literature that actively motivates this research. The thesis represents the design development of a Routing Framework that is QoS aware, runs on hybrid SDN infrastructure, leverages Machine Learning to intelligently choose a route, and complies with the 5G self-organized networking philosophy. Therefore, the chapter covers a comprehensive survey and summarization of five respective domains namely, QoS aware routing in SDN, Hybrid SDN architectures, Application of machine learning in Routing, Self-Healing technologies, and 5G-SON. Each section explores the state-of-the-art algorithms and modelling techniques for the above topics.

The study shows the following. For QoS aware routing algorithm we used a metric that gives a measure of efficiency leveraging the Cost Inefficiency and Runtime Ratio, which shows Lagrange Relaxation method is the most efficient complete but non-optimal algorithm. The optimal hybrid SDN design uses ILP optimization in SDN node deployment, virtualized controller, and config automation in managing data plane devices. For routing optimization, RNN and RL are efficient solutions in terms of computational complexity, however, if prediction efficiency is concerned, DRL techniques give optimal results with a higher computational cost. A basic framework of Self-Healing involves three stages namely, Failure detection, diagnostic and compensation. Detection algorithms dominate unsupervised algorithms, whereas Diagnostic algorithms leverage supervised ones. Compensation algorithms also use Supervised methods but with no exclusive dominance. This chapter also presents the adaptability of service migration as a tool to accomplish Self-Healing. Finally, the chapter presents the concept of KDN and its compliance with next-generation self-organized networking, especially in the context of beyond 5G networks and Industry 5.0.

Chapter 3: Self Optimization

The self-Optimization method aims to seamlessly execute the optimization functions (e.g., QoS, QoE, Routing convergence, etc.) in a network with minimum human intervention. In SDN, the controller is responsible for hosting the SO module. The controller receives several control inputs from the underlying network and computes the optimal configuration parameters, eventually programmed into the devices using remote configuration protocols such as NETCONF[6], RESTCONF[7], and SSH[8]. The latter part is called Self-Configuration if automated with a device automation tool such as Ansible, Puppet, Chef, Salt, etc. In the context of this chapter, the optimization problem is Routing, i.e., computing the best path between a pair of nodes in a network topology.

The chapter discusses the concept of Policy Based Routing (PBR) first, which allows overriding a Router's default routing behaviour by programming it with a set of custom rules, called Policy. Further, a novel cost-relaxation technique, Stochastic Temporal Edge Normalization (STEN), is introduced. STEN is a pre-processing algorithm that results in an isomorphic transformation of the network topology by fusing the node costs into the link cost. This is followed by a routing framework, Cognitive Routing as a Service (CoRoS) which leverages STEN and converges in constant time, providing the most reliable route.

---

[6] https://datatracker.ietf.org/doc/html/rfc6241
[7] https://datatracker.ietf.org/doc/html/rfc8040
[8] https://datatracker.ietf.org/doc/html/rfc4253

3.1. Modeling a novel Policy-Based-Routing (PBR) model

In the classic PBR, a policy is written in the form of route maps. A route map is a series of conditions and action pairs applied on ingress interfaces. When traffic appears on a PBR-enabled interface, its header is examined and matched against the policy. If it matches a condition, the router's control plane invokes the corresponding action (e.g., altering Next-Hop address, TTL value, egress interface ID, etc.). SDN uses the same concept through the South-Bound protocol (SBI), e.g., OpenFlow, OMP, etc. ASICs of the DP devices maintain a match-action table populated and altered by the Controller via OpenFlow. However, the OpenFlow protocol is limited to controlling forwarding traffic only. It does not provide robustness as Route-Maps, and it does not configure devices. There are other protocols such as OVSDB, and SNMP that could collaborate with OpenFlow. However, it results in a complex design.

The proposed hybrid PBR model leverages Route-Maps' robustness but has a central controller injecting the policy into the routers using remote configuration. The SO module computes the optimal parameters for the Policy, which is then plugged with the native PBR modules of the router. The following chapters discuss the methods used by the SO module.

3.2. Stochastic Temporal Edge Normalization (STEN)

In graph theory, Shortest Path Algorithm (SPA) is a class of optimization algorithms that find the best path between a pair of vertices. A fundamental criterion for a SPA is, that the graph must be simple (i.e., no self-loop and no parallel edges must exist in the Graph). In SDN routing, each underlying router informs the controller about its directly connected networks. The controller uses the link-state approach to build the topology and applies SPA to calculate the optimal path. The primary logic is, for $G(V, E)$ the network topology that v_i has an optimal path to a non-adjacent vertex v_j via its neighbour v_k. Then, the routing process installs reachability to all the networks v_j with a v_k. The cost calculation for most of the SPA uses link

parameters only; however, with the rise of NFV and network Softwarization, a significant amount of heterogeneity in computing resource distribution can be evident. A virtual network appliance (e.g., Cisco CSR, Cumulus router, etc.) shares a shared pool of hardware resources through Hypervisors (e.g., VMWare ESXi, Citrix XEN, etc.), where the FIB is virtualized. All flows are processed by software.

Therefore, the processor and memory utilization (node cost) affect the overall processing delay. For instance, a sub-optimal path with an inferior link-state could deliver a packet faster than an optimal path as it has a sufficiently higher processing delay. The traditional routing protocols can't detect it, as node costs are not considered in SPA calculation. Hence, including node cost in path calculation is an obvious solution. However, there are two issues. Firstly, there exists no standard model for relaxing node costs into link costs. Secondly, the node costs appear in the graph as a finite-self loop if the problem is modelled as a Finite-State Machine (FSM). In the latter option, the graph is not simple as it contains self-loops and is not compatible with SPA. The novel technique (STEN) leverages queuing theory-based approximation to relax the node cost into links, making the graph simple to be SPA compatible while preserving the node cost information.

3.2.1. Problem Formulation

This section formulates the STEN problem. Consider $G(V, E)$ is a directed connected graph that represents the network topology, where the vertex set $V = \{v_i | 1 < i < n\}$ contains a programmable forwarding device (e.g., Router, L3 Switch, etc.) and the edge set $E = \{e_{i,j} | adj(v_i, v_j), \forall v_i, v_j \in V, i \neq j\}$ contains the WAN links (Figure 13). The magnitude of an edge $|e_{I,j}|$ is calculated as Eq. *6*, $\mathrm{C_e}$ and $\mathrm{C_n}$ are multivariate real functions calculating the edge

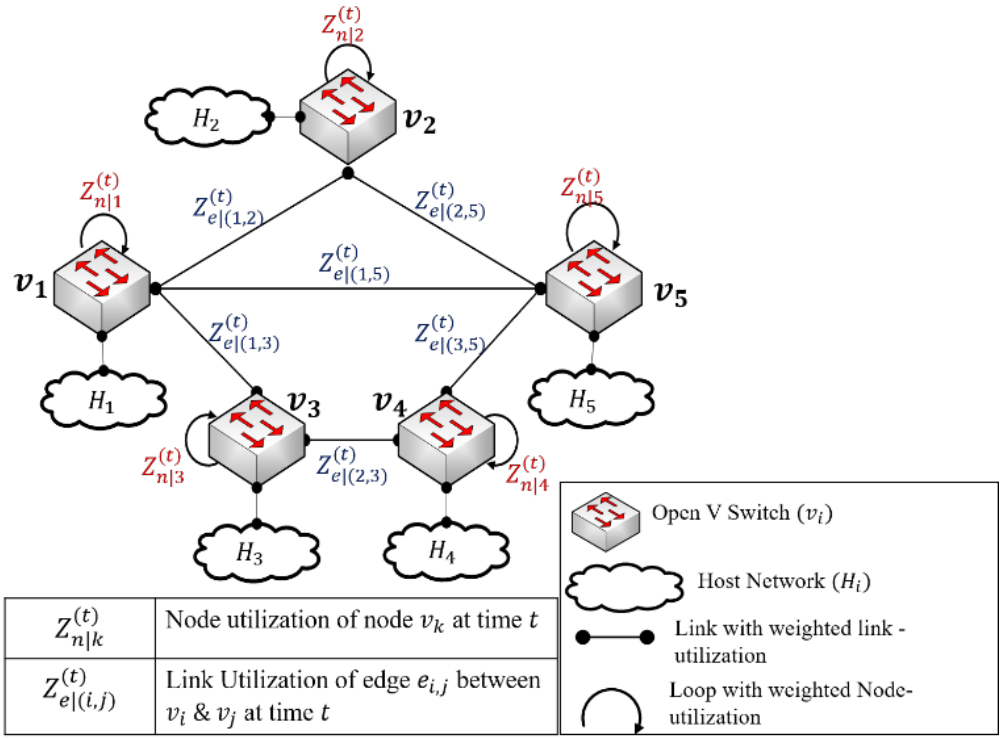

Figure 13: Reference Topology

and node cost respectively using the node parameters X_n (CPU and memory utilization) and edge parameters X_e (bandwidth, delay, load, and reliability).

$$|e_{i,j}| = \begin{cases} Z_{e|(ij)} = C_e(X_e) : i \neq j \\ Z_{n|i} = C_n(X_n) \quad : i = j \end{cases} \qquad \text{Eq. 6}$$

Each node $v_i \in V$ points to a set of directly connected networks H_i, and a set of neighbours $\mathcal{N}(v_i) = \{v_j \in V\}$. Therefore, the routing problem can be summarized as, if v_i has the best path $v_k \in V$ $v_j \in \mathcal{N}(v_i)$ v_i installs routes for all prefixes H_j and v_k as the next hop. The values of X_n & X_e varies over time, the calculated costs generate time series for both node and edge utilization as $Z^{(t)}_{n|i}, Z^{(t)}_{e|(i,j)}$ $\forall v_i, v_j \in V$. The problem is to $\aleph: G(V,E) \rightarrow G'(V,E')$ such that $Z^{(t)}_{e'|(i,j)} \in E'(G'), Z^{(t)}_{n|i} \in V(G),\ Z^{(t)}_{e|(i,j)} \in E(G)$ for any t instance. It diminishes the $Z_{n|i}$ of G' which makes G' a simple graph and thus it becomes compatible for SPA to run on it. The granularity of the transformation, i.e., node and link parameters used, and cost calculation function, is discussed later in this chapter. This section focuses on the STEN relaxation process.

3.2.2. Relationship between energy consumption and Routing

Assume that an application requires a total of E amount of energy to run locally. Without loss of generality, it is assumed that part of the application runs locally, and the rest is offloaded to it remotely. Then, E can be expressed as a sum of the energy consumed for local execution (E^l), remote execution (E^r)) and data transfer (E^t). From the source's perspective $E^r = 0$ as it is not utilizing the source's energy resources. Hence, the actual energy saved by offloading the application partially is $E^l - E^t$, as the energy spent for data transfer acts as a penalty for the saved energy. Initially proposed by Microsoft in their article on the MAUI framework[35], it formulates the optimal saved energy for a call graph in a distributed application with a constrained latency.

The proposed solution is a $0-1$ IIPP problem. The objective function maximizes the energy saved by executing a method remotely. The saved energy is the difference in the total energy cost of local execution ($E_v^l \mid v \in V$) and the total data transfer cost for executing the method, ($C_{u,v} \mid u, v \in V \; and \; e_{u,v} \in E$). There are two constraints for the above objective function. First, the total time for the execution $T_v^l + T_v^r$ must be within a certain latency. T_v^l & T_v^r are referred to as the local and remote execution time of $v \in V$. Second, only Remote methods can be offloaded for remote execution. The formal representation is given below (Eq. 7):

$$\begin{gathered} maximize \sum_{v \in V} I_v \times E_v^l - \sum_{e_{u,v} \in E} |I_u - I_v| \times C_{u,v} \\ subjct\ to, \left(\sum_{v \in V} (1 - I_v) \times T_v^l + I_v \times T_v^r \right) + \sum_{e_{u,v \in E}} \left(|I_u - I_v| \times B_{u,v} \right) \leq L \\ and, I_v \leq r_v \; \forall v \in V \end{gathered} \qquad \text{Eq. 7}$$

where, I_v is an integer that is equal to 0 for local execution and 1 for the remote. R_v represents methods marked as remote, and $B_{u,v}$ is the state transfer time from u to v. It can be inferred from equation Eq. 7 that the latency constraint is linearly dependent on the execution time

T_v^l & T_v^r and state transfer time $B_{u,v}$. Further, remote execution and state transfer time are proportional to the network delay. Hence, a routing protocol that guarantees to choose a path reactively that costs the least latency, with every altering network condition, would meet the latency satisfiability constraint with the highest probability. Eventually optimizing the saved energy, defined as the equation Eq. 7. The following section discusses our proposed algorithm's design cost and design, stating the relationship between the routing protocol and energy savings.

3.2.2. The Queuing model of a Stochastic Network

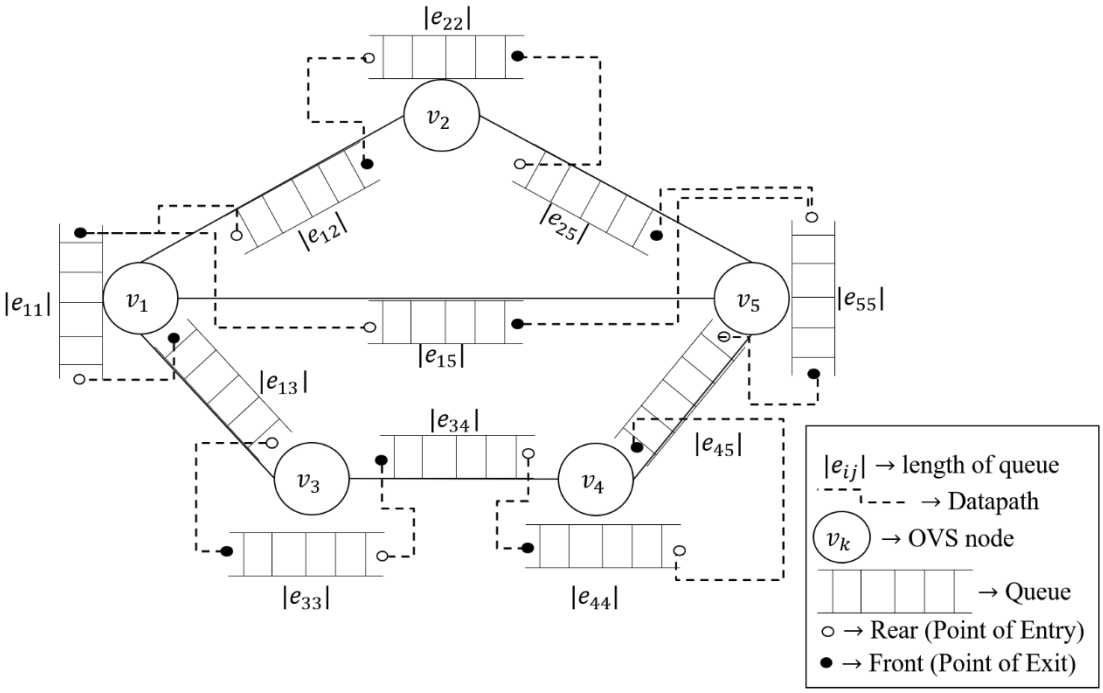

Figure 14 Queuing model of the network with service queues at nodes

STEN uses Stochastic Network Calculus (SNC) theory, which renders a network as a collection of queues, i.e., each node and the edge are replaced by their queuing functions. Traffic flowing along a path a $P_{i,j}$ migrates through a sequence of node and edge queues. Knowing the distribution of them could approximate the flow delivery time. Additionally, if each queue's delay is computed, both node (with service and processing delay) and edge (with transmission and propagation delay) queues become operationally compatible. Therefore, an additive operation between all the Queues along $P_{i,j}$ provides the end-to-end delay along the same path;

twice the RTT measured from either end. If a packet appears to an ingress interface at the time t_0 and leaves at t_k from an egress interface[9], the queuing time = service time + processing time (Eq. 8)

$$(t_k - t_0) = t_q. \quad \text{Eq. 8}$$

As the system load increases, more packets gather in the service queue, resulting in a longer queue size which is proportional to T_q; hence, $Z_{n|i} \propto t_q$. For the sake of simplicity, we do not consider the situation of link congestion, i.e., the egress interface pumping more packets into a link than its Bandwidth-Delay Product (BDP[10]). Part of the reason is that all modern network devices implement a QoS mechanism (RSVP or DSCP) that prevents this from happening. Figure 14 depicts the queuing model of the topology shown in Figure 13, where the weights of each edge and self-loop become the length of the corresponding queues. Each queue has a point of entry and exit called rear and front, denoted as hollow and solid circles respectively on the figure. For depiction simplicity, we assume that the links are simplex, i.e., $e_{I,j}$ can only carry data from v_I to v_j not vice versa. The queuing system can be heterogeneous, i.e., each queue may run a different scheduling mechanism. Therefore, it is obvious to generalize it. As mentioned earlier, the queue size is proportional to the processing load for the nodes and traffic load for the edges. The queue size is also proportional to the QT; the mean of QT is also called Average Waiting Time (AWT). Hence, choosing the least time-consuming path can also be a sequence of queues. The sum of AWT is the least among the possible alternatives, which inherently select nodes and edges comparatively underloaded.

---

[9] Due to the Split-Horizon rule used as a default loop-prevention technique in Distance Vector Routing, ingress and egress ports are generally non-identical, except special cases like NBMA or DMVPN networks.

[10] The BDP defines the maximum number of bits that can fit into a channel without having a collision.

Average Waiting Time (AWT) of a Node ($\mathbf{W_{node}}$)

A queue is mathematically expressed as $A/B/c/K$ where A is the distribution of inter-arrival time, B is the service time, c is the number of servers, and K is the capacity. Since the packets arrive from many sources and the service time depends on the system load, which depends on several random causes, A & B have been chosen as distribution agnostic. Also, we assume the problem as an unbounded buffer problem with a single server; hence $k = \infty$ and $c = 1$. This makes the queuing model $G/G/1$. From Little's theorem (Eq. *9*) [197].

$$W = W_q + \frac{1}{\mu} = \left(\frac{L_q}{\lambda} + \frac{1}{\mu}\right) = O(L_q) \quad \text{Eq. 9}$$

Where,

W : AWT of the system

W_q: AWT of the queue

L_q : mean number of requests in the queue

λ : mean rate of interval

μ : mean service rate

From the approximated value of L_q for $G/G/1$ queues derived by Marchal (Eq. 11)[197],

$$L_q = O(\rho^2, \sigma_s^2, \sigma_a^2, \mu^2, \lambda^2) \quad \text{Eq. 10}$$

where ρ: utilization of the server and σ_s^2, σ_a^2 : variance of the service & inter-arrival time, respectively; Hence, from equations Eq. 9 & Eq. 10 Z_n is the node utilization.

$$W_{node} = O(L_q) = O(\rho^2) = O(z_n^2) \quad \text{Eq. 11}$$

Therefore, as the system goes busy Z_n decreases and W_{node} (AWT[11]) increases quadratically (Eq. 11).

Average Waiting Time of an Edge W_{edge}

The AWT of edges is relatively simpler to calculate since the channel is First In First Out (FIFO); we consider the mean round trip time (RTT) as AWT, which is inversely proportional to the edge cost. Therefore,

$$W_{edge} = O(RTT) = O\left(\frac{1}{Z_e}\right) \qquad \text{Eq. 12}$$

Figure 15 shows the normalized version of Figure 14. All the loops $|e_{ii}|$ are set to zero; instead, their values are distributed among the adjacent edges of the node v_i. The coefficient α_j^I

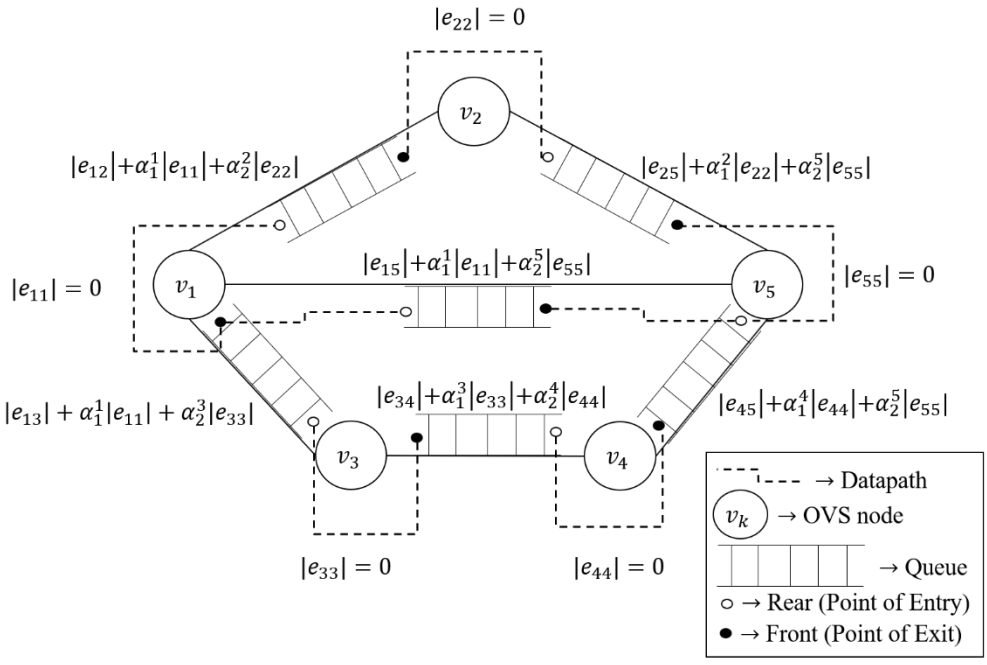

Figure 15 Queuing model after relaxation of the service queues from nodes to links

[11] Average Waiting Time (AWT) is a measure of delay a process has to face before it is served. AWT is statistically calculated by averaging the individual waiting time of all processes in a queue.

is a rational number between [0,1] that denotes a fraction of $|e_{ii}|$, such that $\sum_j \alpha_j^i = 1$. It specifies the next-hop probability of a switch v_I distributed over its incident edges. This edge normalization process is temporal as it changes time-to-time and stochastic because the fraction is probabilistic and distribution agnostic. Once normalized, the graph realigns, the busy nodes move farther, and the free nodes come closer. Consequently, running any shortest path algorithm will choose a path with minimum path length, which comprises freer nodes than the busy ones. The normalization function ℵ transforms a graph with a self-loop into one with a normalized edge. ℵ is defined formally below (Eq. 13),

$$\aleph\big(G(V,E)\big) \rightarrow G'(V,E') \qquad \text{Eq. 13}$$
$$Such\ that,\ |e_{ij}| \rightarrow |e_{ij}| + \alpha_k^i|e_{ii}| + \alpha_k^j|e_{jj}|\ and, |e_{ii}| = 0\ \forall e \in E|$$

3.2.4. Numerical Example of STEN

Step 1: Building the Binary Adjacency Matrix: The controller receives HELLO messages from downstream routers and locally connected networks. Using Link-State logic (i.e., if two routers have a common local network, then they are neighbours), the controller builds the topology and stores it as a binary adjacency matrix Adj_b. (Figure 16).

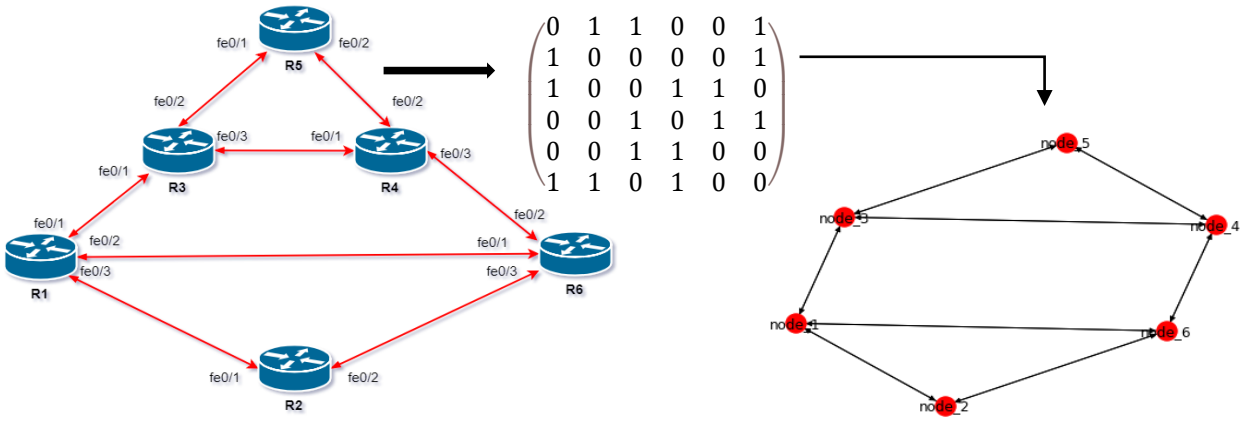

Figure 16 Building a Graph from the adjacency matrix of the topology

Step 2: Calculating Cost matrix: For each time instance, the controller fuses the node and edge costs $Z_{n|i}^{(t)}$, $Z_{e|(i,j)}^{(t)}$ to Adj_b using the following transformation (Eq. 14),

$$Adj_c^{(t)} = \left[Z_{e|(i,j)}^{(t)}\right]_{n\times n} \circ Adj_b + \left(\left[Z_{n|i}^{(t)}\right]_{1\times n} . I_n\right) \quad \text{Eq. 14}$$

Firstly, the Hadamard product $\left[Z_{e|(i,j)}^{(t)}\right]_{n\times n} \circ Adj_b$ copies the edge costs $Z_{e|(i,j)}^{(t)}$ at the (i,j) position of the Adj_c matrix. Secondly, the dot product $\left(\left[Z_{n|i}^{(t)}\right]_{1\times n} . I_n\right)$ is summed to place $Z_{n|i}^{(t)}(i,i)$ position along the diagonal of Adj_c in $O(1)$ with $O(n^2)$ number of threads and keeps both the node and link costs within the same data structure $O(n)$ space. However, Adj_c cannot be used for SPA as the diagonal needs to be zeroed out first. Adj_c generates a times series of matrices. Its value can be any arbitrary $Adj_c^{(t)}$.

Step 3: Relaxing the node costs into edges: The controller uses the mean load share of the interfaces to distribute the node cost into the edges. It measures the mean load share as the moving average of loads from the connected interfaces then normalizing into a $[0,1]$ scale over a defined window. Therefore, it yields the probability of a packet being forwarded to an egress interface. The Adj_l keeps a record of it, the row-wise sum is always 1 or 0. The Adj_l matrix is not symmetric as the load at two ends could be different. The relaxation function is as follows (Eq. 15)

$$Adj_s^{(t)}[i,j] = Adj_c^{(t)}[i,i]^2 Adj_l^{(t)}[i,j] + Adj_c^{(t)}[i,j] \quad \text{Eq. 15}$$

First, the node cost at $Adj_c^{(t)}[i,i]$ is squared to calculate the AWT_n for the i^{th} node. The faction of the AWT corresponding to an edge $e_{(i,j)}$ is calculated by multiplying it with a load of edge $e_{(i,j)}$ which then adds up to the corresponding edge cost got minimization. The node costs are negated for simplifying the following step. The resultant matrix is the Affinity matrix Adj_a.

The sum $Adj_a + Adj_c = Adj_s$ (STENed matrix) relaxes the node costs into the edges making the resulting graph a simple graph, isomorphic to the one represented by Adj_b. (Figure 17)

$$\boldsymbol{Adj_l} = \begin{pmatrix} 0 & 0.268 & 0.325 & 0 & 0 & 0.407 \\ 0.458 & 0 & 0 & 0 & 0 & 0.542 \\ 0.109 & 0 & 0 & 0.188 & 0.703 & 0 \\ 0 & 0 & 0.211 & 0 & 0.326 & 0.463 \\ 0 & 0 & 0.364 & 0.636 & 0 & 0 \\ 0.297 & 0.293 & 0 & 0.428 & 0 & 0. \end{pmatrix}$$

$$\boldsymbol{Adj_a + Adj_c} = \begin{pmatrix} -0.649 & 0.248 & 0.322 & 0 & 0 & 0.079 \\ 0.254 & -0.357 & 0 & 0 & 0 & 0.102 \\ 0.128 & 0 & -0.713 & 0.584 & 0.703 & 0 \\ 0 & 0 & 0.306 & -0.79 & 0.292 & 0.193 \\ 0 & 0 & 0.037 & 0.18 & -0.216 & 0 \\ 0.157 & 0.396 & 0 & 0.308 & 0 & -0.861. \end{pmatrix} +$$

$$\begin{pmatrix} 0.649 & 0.933 & 0.863 & 0 & 0 & 0.382 \\ 0.933 & 0.357 & 0 & 0 & 0 & 0.480 \\ 0.863 & 0 & 0.713 & 0.598 & 0.022 & 0 \\ 0 & 0 & 0.598 & 0.790 & 0.082 & 0.345 \\ 0 & 0 & 0.022 & 0.082 & 0.216 & 0 \\ 0.382 & 0.480 & 0 & 0.345 & 0 & 0.861 \end{pmatrix}$$

$$= \begin{pmatrix} 0 & 1.182 & 1.185 & 0 & 0 & 0.462 \\ 0.519 & 0 & 0 & 0 & 0 & 0.582 \\ 0.171 & 0 & 0 & 1.183 & 0.221 & 0 \\ 0 & 0 & 0.874 & 0 & 0.374 & 0.538 \\ 0 & 0 & 0.115 & 0.31 & 0 & 0 \\ 0.235 & 1.218 & 0 & 1.092 & 0 & 0 \end{pmatrix} = \boldsymbol{Adj_s}$$

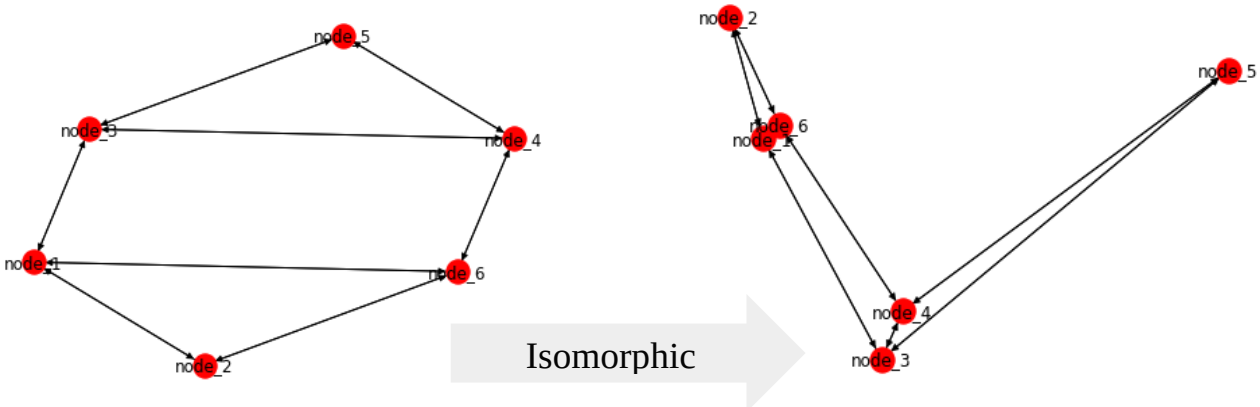

Figure 17 Transforming the topology graph to an isomorphic graph after STEN transformation

3.2.5. Experimental Validation

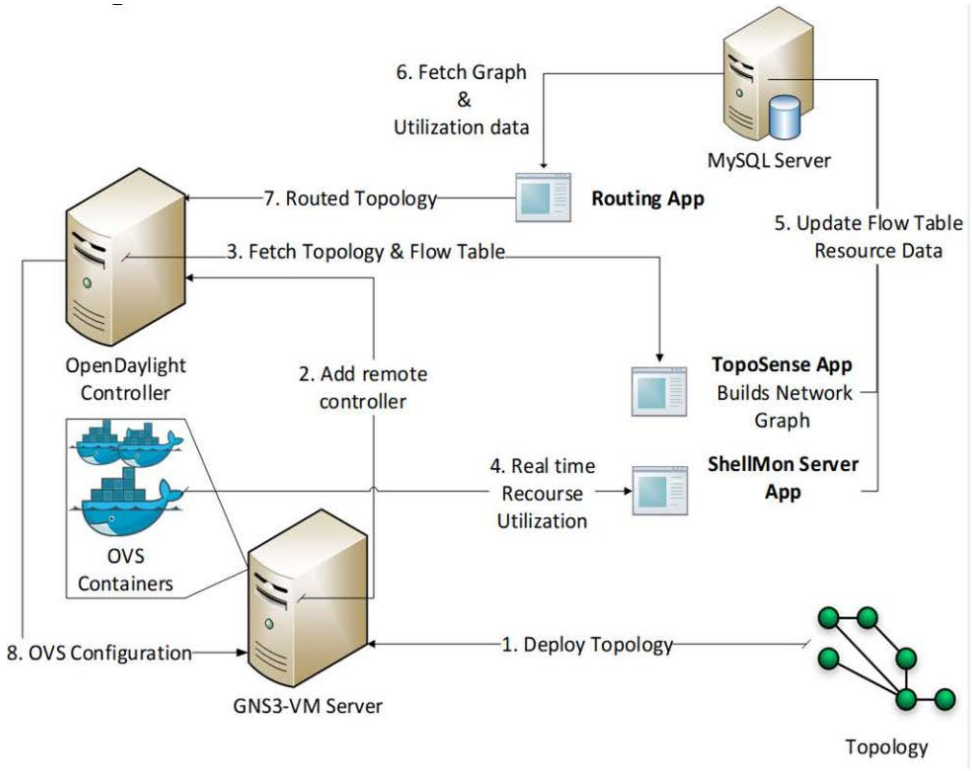

Figure 18 Experimental Setup and Dataflow Architecture

We implemented a testbed (Figure 18) using the GNS3 network emulator and OVSs hosted using Docker containers, OpenDaylight (ODL) beryllium SR4 as the SDN Controller and MySQL Server is used for middleware & database management. Three bespoke apps (*ShellMon*, *TopoBuild*, *TopoRoute*, and *TopoSense*) run in the application layer to apply STEN to downstream IP flows. (Explained in Chapter 4).

A. **Experimental Setup:** Each OVS runs the *ShellMon* client and sends event-driven resource updates to *ShellMon* Server. The *TopoSense* app retrieves topology and flow table information from ODL using RESTConf protocol from nodes/topology and nodes/inventory resources, respectively, and updates the database. Route-App fetches data from the database, runs *Algorithm 1* generates a graph with resource information and the shortest path for eligible edges. Each shortest path then gets configured to the OVS using OpenFlow packet out messages from the controller. Figure 18 depicts the complete data flow.

B. Methodology: Since the router OS is mostly monolithic, altering the code is a complex task. Therefore, the system has been tested using traditional routing, and OVSs with STEN flows with Quagga software routers. To emulate the routing behaviour, the controller pushes the flow instruction to the switch. The routers run both RIP and OSPF to generate a dataset for Distance-Vector and Link-State routing. Two nodes are selected as Client and Server and placed at the end along the network diameter to simulate traffic flow. After the routing protocol determines the optimal path, we choose an intermediate node as Victim. A Linux tool, Stress, progressively overload the victim, and the time to deliver a data burst using *Iperf*[12] is observed to simulate the bottleneck behaviour. The rate of increasing time is the measurement of efficiency.

C. Experimental Results & Discussion:

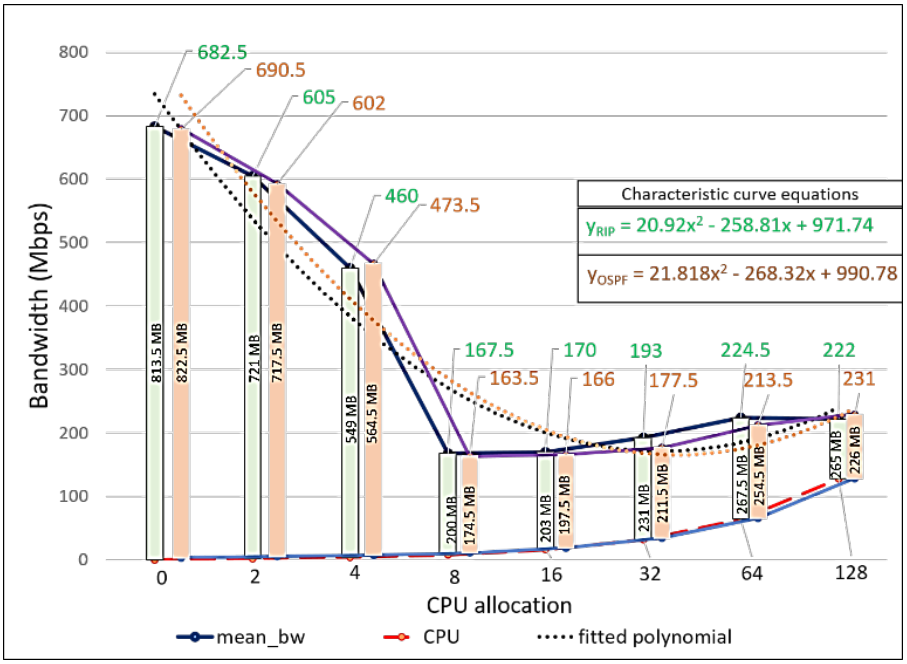

Figure 19 Effect of CPU utilization in Ent-to-End throughput

[12] Iperf is a link benchmarking tool that tests the utilization based on TCP and UDP flows. For more information visit https://iperf.fr/ .

Algorithm 1**: Stochastic Temporal Relaxation Routing Algorithm (STR-RA)**

Input: Graph $G(V, E)$ - Topology from SDN Controller

$Z_{n|v}^{(t)}$ & $Z_{e|l}^{(t)}$ - Utilization $\forall v \in V \; and \; \forall l \in E$ at the time, t.

Output: Set of Routes $R_{ij}^{(t)}$

Steps:

```
While (true) {
    Set route ← r_temp ← ϕ
    Normalize G : G' = ℵ(G) // Apply STEN.
    For all vertex pair (v_i, v_j) ∈ V(G') × V(G') {
        If (v_i, v_j) ∈ E'{
            If min(e'_ik) + min(e'_kj) + min(E'''') < |e'_ij|
                route ← route ∪ e'_ij
            Δe'_ij : change in edge weight.
            If Δe'_ij > min(e'_ik) + min(e'_kj)
                route ← route ∪ e'_ij
        }
        Else r_temp ← r_temp ∪ e'_ij
    }
    If route ≠ ϕ
        route ← route ∪ r_temp
    For all (v_i, v_j) ∈ route
        R_ij ← R_ij U dijekstra(v_i, v_j)
    For all r_ij ∈ R_ij call Flow Modifier
    Sleep (Timeout)
}
```

The first set of experiments (Figure 19) measures the dropping end-to-end throughput over CPU load. With no change in link capacity, the experiment shows that the throughput falls from 700 Mbps to 250 Mbps when additional threads increase from 0 to 128. A fitted polynomial comes as a quadratic one which further validates the claimed relationship between utilization and delay.

In the following experiment (Figure 20), a bespoke application, *TopoRoute*, programs the ODL controller, which writes the flow entries to the downstream switches using OpenFlow.

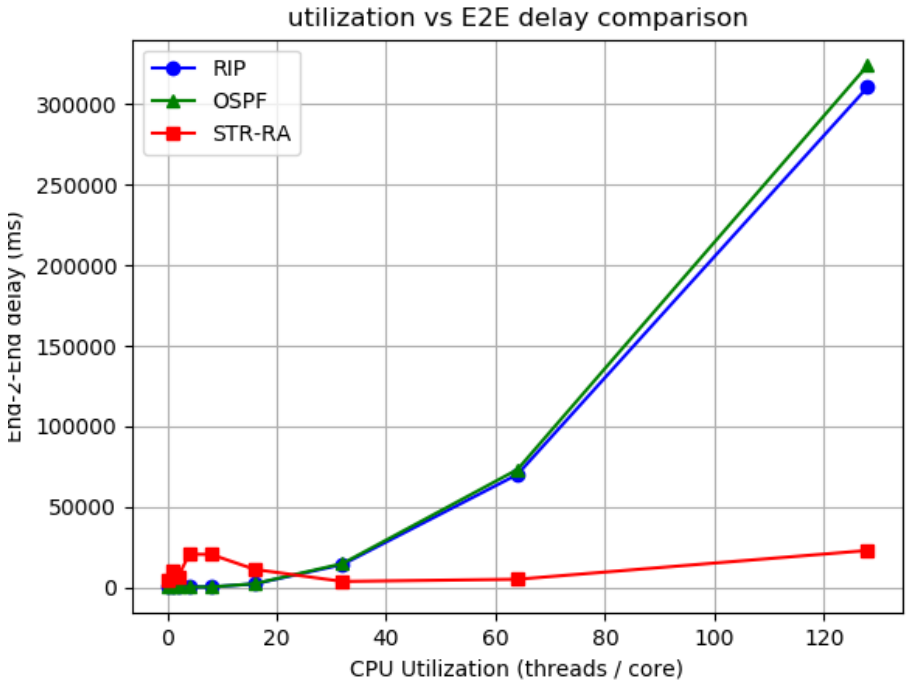

Figure 20 Comparison between RIP, OSPF & proposed STR-RA in Utilization vs Delay characteristic

Programming the switches simulates the routing behaviour. The *TopoRoute* application hosts a reactive algorithm named Stochastic Temporal Relaxation Routing Algorithm (STR-RA) (Algorithm 1), which computes STEN on a given graph and generates routes.

Running Dijkstra's algorithm for all parts of vertices would cost $O(|V|^4)$. To reduce it, Algorithm 1 chooses only those pairs of vertices which are eligible. This means that they have a possibility of replacement by an alternate path. The eligibility criteria are listed below,

a. If e_{ij} is an edge between two adjacent vertices (v_i, v_j) and the sum of minimum weighing incident edges of the subjected vertices and the minimum weighing edge of the entire graph is less than $|e_{ij}|$, i.e.

$$\min_{(i,k) \in E'}(|e'_{ik}|) + \min_{(k,j) \in E'}(|e'_{kj}|) + \min_{e \in E'}(E''') < |e'_{ij}| \qquad \text{Eq. 16}$$

b. If the change in the value of a direct edge e'_{ij} is denoted as $\Delta e'_{ij}$, exceeds the sum of minimum weighing incident edges of the subjected vertices.

$$\Delta e'_{ij} > \min_{i,k \in E'}(|e'_{ik}|) + \min_{j,k \in E'}(|e'_{kj}|) \quad \text{Eq. 17}$$

c. All indirect vertex pairs, i.e., $(v_i, v_j)\ |e_{ij} \notin E$ are eligible.

This doesn't reduce the asymptotic upper bound of the runtime but the lower bound significantly when the eligible edges are few. STR-RA calls the flow modifier program that encodes the flow rules using OpenFlow.

Algorithm 2: **Flow Modifier**

Input: Route $r_{ij} \in R_{ij}$

Output: Flow entry F_{ij}

Steps:

```
1. For all v_k in r_ij{
2.     If succ(v_k) ≠ ϕ{
3.         ovs ← v_k
4.         sip ← H_i = {h_i}
5.         dip ← H_j = {h_j}
6.         port ← p(succ(v_k))
7.         ovs.addFlow(
               nw_src = sip
               nw_dst = dip
               action = output:port )
       }

   }
```

When comparing the results of Experiment 1 and Experiment 2 by plotting the end-to-end delay, STR-RA shows to maintains a fixed delay bracket while RIP and OSPF climb quadratically. This is since when the intermediate node's utilization exceeds a threshold, STEN virtually stretches all its incident edges. The shortest path algorithm reconverges, and an alternate path is discovered via a less stressed node, and the traffic gets steered through it. A small bump can be noticed during STR-RA's convergence when the threshold was yet to be met. A slight rise

in the latter section is since the simulation was carried out on the same system. Thus, stressing one node also affects the physical processor load, hence other nodes.

3.3. Rapid Convergence in Multi-Path Routing (*MRoute*)

The key to achieving rapid convergence in a data network is to cache pre-computed forwarding information using some $O(f(n))$ time, where n is the number of nodes in the topology and used on-demand in $O(1)$ time without needing to re-calculate. NetFlow or the Fast-Switching model is the earliest such approach which is debuted as an industry standard replacing its predecessor, the process-switching. Although the "*Route once and switch many*" philosophy gave an initial boost to the L2 and L3 operations by offloading the lookups from the route/switch processor to the hardware; however, it failed to keep up with the scalability. The switching processor eventually gets involved with a limited cache memory once a cache replacement is needed. Cisco solved this issue using their proprietary *Cisco Express Forwarding* (CEF) technique which dumps the entire *Routing Information Base* (RIB) into hardware *Forward Information Base* (FIB) and Adjacency Table. Any traffic which does not match any *Access Control List* (ACL) or QoS policy does not involve the control plane processing.

Routing protocols support multipathing natively, OSPF provides equal path load-balancing between multi-path routes using round-robin and EIGRP offers unequal cost load-balancing using a variance multiplier[13] along a primary path or Successor route (S) and a backup path or *Feasible Successor* route (FS). If S fails, EIGRP switches to FS in constant time,

[13] EIGRP Load balances between $S \cup FS$ routes if routes $metric(r_i) \leq variecnce \times metric(s)\ \forall r_i \in FS$ for each $\left\lceil \frac{metric(S)}{metric(r_i)} \right\rceil$ packet sent along S route, 1 packet is sent along FS route.

proving rapid convergence with single-successor fault tolerance. However, if the FS route fails too, the router needs to defuse the convergence process across the network using the *Query/Reply* and *SIA-Query/SIA-Reply* control packets. During this phase, all routers put the lost routes in an active state, and all control processing freezes for the subjected routing protocol instance. This results in an $O(d)$ communication complexity, where d is the network diameter. OSPF is a Link-State routing protocol, that updates the rest of the peers within the same area about the lost route at an infinite cost. This triggers the SPF convergence on every node with $O(n^2)$ time for pathfinding and $O(d)$ Time for control message propagation.

For an SDN, the controller possesses a birds-ey view of the network, and thus, with link-state logic, it can realize the underlying topology in $O(1)$ time. Every downstream router advertises its locally connected routes towards the controller, which is ideally one-hop away[14]. *MRoute* takes the topology graph $G(V, E)$ as input and generates all possible paths between every pair of nodes. The controller stores the result into a data structure called Route Forest ($RF = \{\mathrm{RT}_{i,j}\}$), which is a collection of Route-Trees $RT_{i,j}$. Every $RT_{i,j} \in RF \mid \forall(i,j) \in V^2, i \neq j$ is an n-ary tree that stores all paths between $\mathrm{v_I}, \mathrm{v_j} \in \mathrm{V}$. A Route-Tree is finite with a maximum depth and width as the diameter of the network (d) and $|V| - 1$, respectively. Further discussion is provided by explaining that generating an $RT_{i,j}$ It is computationally independent with only a read operation on the shared adjacency matrix. Therefore, the controller can simultaneously compute $\mathrm{RT_{I,j}}$ for each node pair $(i,j) \in V^2$ using multithreading. Moreover, as the number of edges is fixed, and the edges are distributed across all the Route-Trees as branches; therefore, a dynamic programming approach with memorization would lower the computation complexity of several orders. That said, STEN

[14] In practice, an SDN controller establishes a secure tunnel (e.g., OMP on IPsec, OpenFlow on TLS/SSL)

periodically recalculates the updated link cost, a Hash-Map with $O(|E|)$ space stores the updates and revises the route-forest in $O(1)$ time with $O(|E|)$ Threads.

3.3.1. System modelling

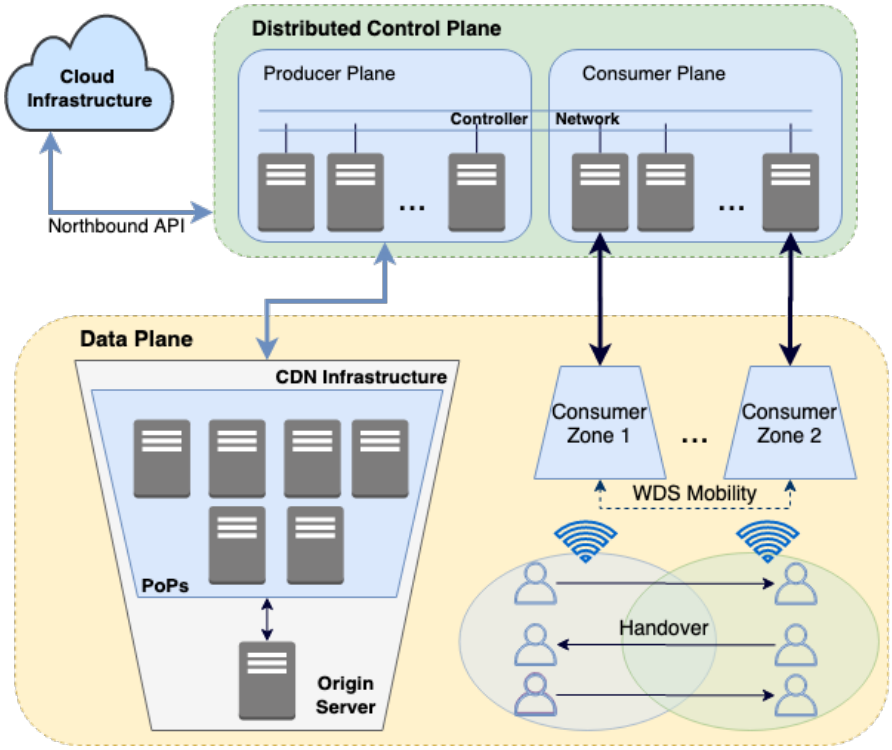

Figure 21 Reference Architecture of an SD-WAN with a CDN use case

To explain the *MRoute* algorithm, this section instantiates an SD-WAN[198] implementation as a use-case related to CDN provisioning. Figure 20 depicts the system architecture. The DP constitutes two groups of servers, first, that originates the traffic, such as a Point-of-Present (PoP) and a CDN infrastructure, and second the hosts the consumers. Controllers that manage the producer side optimize the egress traffic, whereas the consumer plane optimizes the ingress. An overlay network logically segregates the producer and consumer plane and maintains connectivity with respective edge devices. The edge devices manage any mobility management such as handovers of devices. The consumer side segments its user base into several zones to facilitate hierarchical routing. Interzonal communication takes place via the controller. However, a dynamic multi-point VPN (DMVPN phase-3) can provide site-to-site on-demand

connectivity with summarized routes. Each controller aggregates partial topology information from downstream edge routes in a link-state manner and generates a topology graph. Each controller shares complete topology information to its direct neighbours and summarizes 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 neighbour. Generally, all routing protocols follow a four-step process in execution. First, neighbour discovery, secondly topology synchronization, followed by the shortest pathfinding and finally, when the routing table converges, it stays idle in the control plane until a primary route fails and a reconvergence is needed. We assume the controller and edge network topology is unvarying. The following steps describe the process in detail.

Phase 1: Neighbor Discovery:

Controllers create end-to-end tunnels to form adjacencies. Edge server registers themselves to a policy server and gets tunnel parameters for establishing two-way communication. However, for the sake of simplicity, we consider simple GRE tunnels among the controllers. In other words, the neighborship may be either static or dynamic. Every controller maintains a neighbour table to keep track of the neighbours' activity. Figure 22 shows a sample topology of a controller network with flow vectors after the neighbour discovery.

Phase 2: Initial controller advertisement:

In traditional routing protocols like RIPv2, OSPFv2, and EIGRP, control packets are exchanged using both multicast and unicast methods. However, SD-WAN depends on overlay networking, where P2P or P2MP VPN tunnels connect the edge devices to the controllers. Each controller $\mathrm{CON_I}$ generates a topology $G_i(V_i, E_i)$ for its underlying edge network, where V_i and E_i are the set of downstream edge routers and their corresponding links, respectively. CON_i computes the

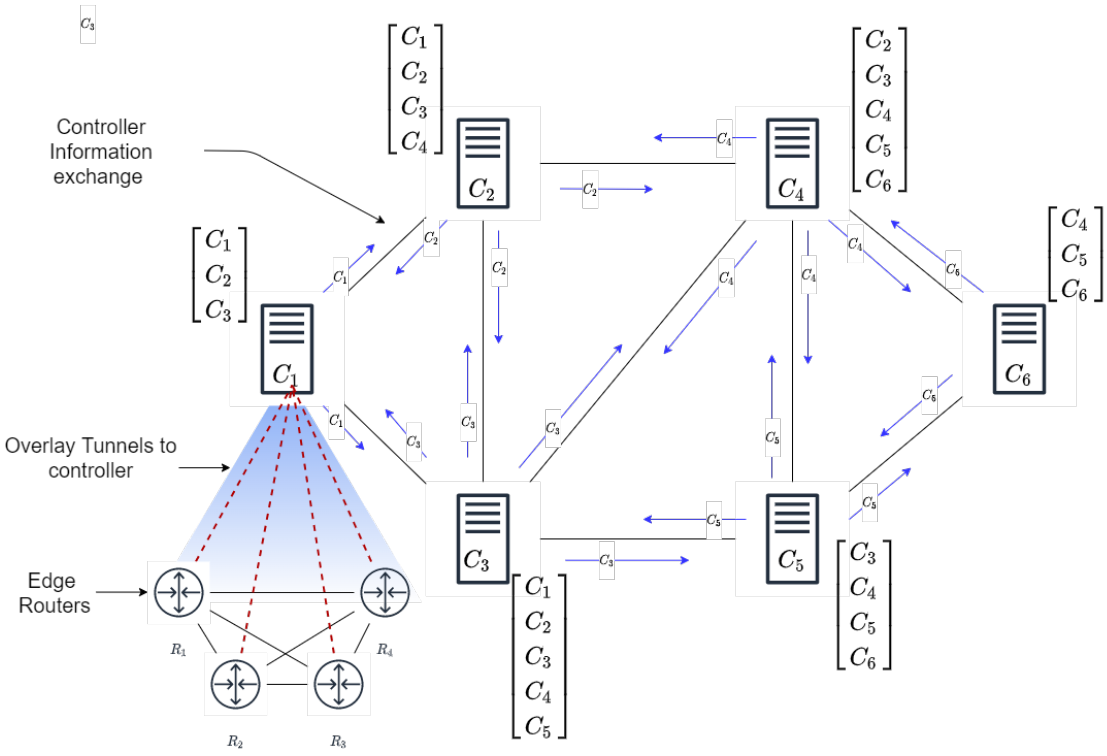

Figure 22 A use-case model of CDN implemented over an SD-WAN

topology matrix $C_i = [c_{i,j}]_{V\times V} \mid \forall e_{i,j} \in E$, members of which are a set of possible costs between the pair of nodes in G_i. For static neighborship, the neighbour database populates entries with indefinite ageing time while adding neighbours. For dynamic neighbour discovery, a pair of neighbours connect on-demand and any tunnel that ages above a limit (typically 2 hours) are removed. A CON_i advertises only to its full C_i to its direct neighbours $\mathcal{N}(\mathrm{CON_i})$. When a controller acts as a transit node, it forwards its neighbour's cost matrix to another neighbour, by sending a list of reachable network prefixes.

Phase 3: Vertex set augmentation:

After the neighbour discovery and initial advertisements, all controllers become aware of their neighbours' topology and far-end networks accessible by the non-neighbour controllers. The Vertex-Set Augmentation (VSA) process augments the producer side, which comprises origin servers from the consumer and end-users. The segregation enables easy policy maintenance,

especially for QoS and PBR. 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 mean processing load (CPU and memory) and communication load (bandwidth utilization and congestion) using a moving average with a given window size. A heavily loaded router maintains a longer service queue, which results in a delay in packet processing. In this phase, the controller ranks routers based on their load.

Phase 4: Route Calculation

Every controller computes a full-mesh graph $\mathcal{G}_i(V_i, \mathcal{E}_i)$ from the underlying topology $G_i(V_i, E_i)$ To maintain the database of all possible paths between all-pair of nodes. It tags the path with a unique Route_ID. The process starts with a controller $\mathrm{CON_I}$ computing a Route Forest $\mathrm{RF_I}$, comprises a collection of Route Tree $RT_{s,d} \mid \forall (s,d) \in V^2, s \neq d$. An $RT_{s,d}$ keeps all possible paths souring from s towards d. Each of its branches is a unique path connecting $\mathrm{v_s} \in \mathrm{V}$ and $v_d \in V$. RT is a ternary tree of exponential, i.e., $O(2^n)$ space complexity. Moreover, in case of no topology change, it serves no purpose but to occupy the space. Therefore, a compression method would solve the problem by transforming the RF_i into an optimal data structure with polynomial complexity. The proposed approach leverages the FSM that compresses the $O(2^n)$ sized Route-Forest into a graph of size $O(n)$. Details of this compression are discussed in a later section.

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

Finally, the full-mesh graph $\mathcal{G}(\mathrm{V}, \mathcal{E})$ is generated by aggregating the Router_IDs of $T_{s,d}$ and mapped as $e_{s,d} \in \mathcal{E}$. Thus, the FSM stores the node sequence for every path and □ 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 is trivial to represent graphs in their adjacency-matrix form with $O(n^2)$ space or with $O(n)$ Space if linked representation is used. The FSM matrices are exchanged during database synchronization between controllers. Immediate neighbours exchange the full matrix so each controller can aggregate its topology to its neighbours. However, being a transit controller suppresses most details and only advertises Router_IDs learned from a remote controller. The primary reason is efficient space management. Figure 22 depicts the complete process of the controller generating the full mesh graph from their underlying topology.

3.3.2. Computing all-possible paths

The *MRoute* Algorithm

Controllers run *MRoute* (Algorithm 3) 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 leaf-to-root traversal of $t_{s,d}$ is a possible path between v_s and v_d Figure 18 RouteTree generated by *MRoute* w.r.t. for $\mathrm{r}_{1,3}$depicts the generation of Route-Tree $T_{1.3}$ regarding the topology shown in Figure 22.

Figure 24 shows the route tree corresponding to $\mathrm{R}_{1,3}$. The algorithm uses the backtracking principle to enumerate all possible routes between source and destination vertices, in this context, (v_1, v_3). The following paragraph explains the working principle of *MRoute*.

MRoute has two phases; during the Grow Phase the route tree grows by recursively expanding its branches, the Shrink Phase runs intermittently with the Grow phase, where invalid paths are pruned out from the tree.

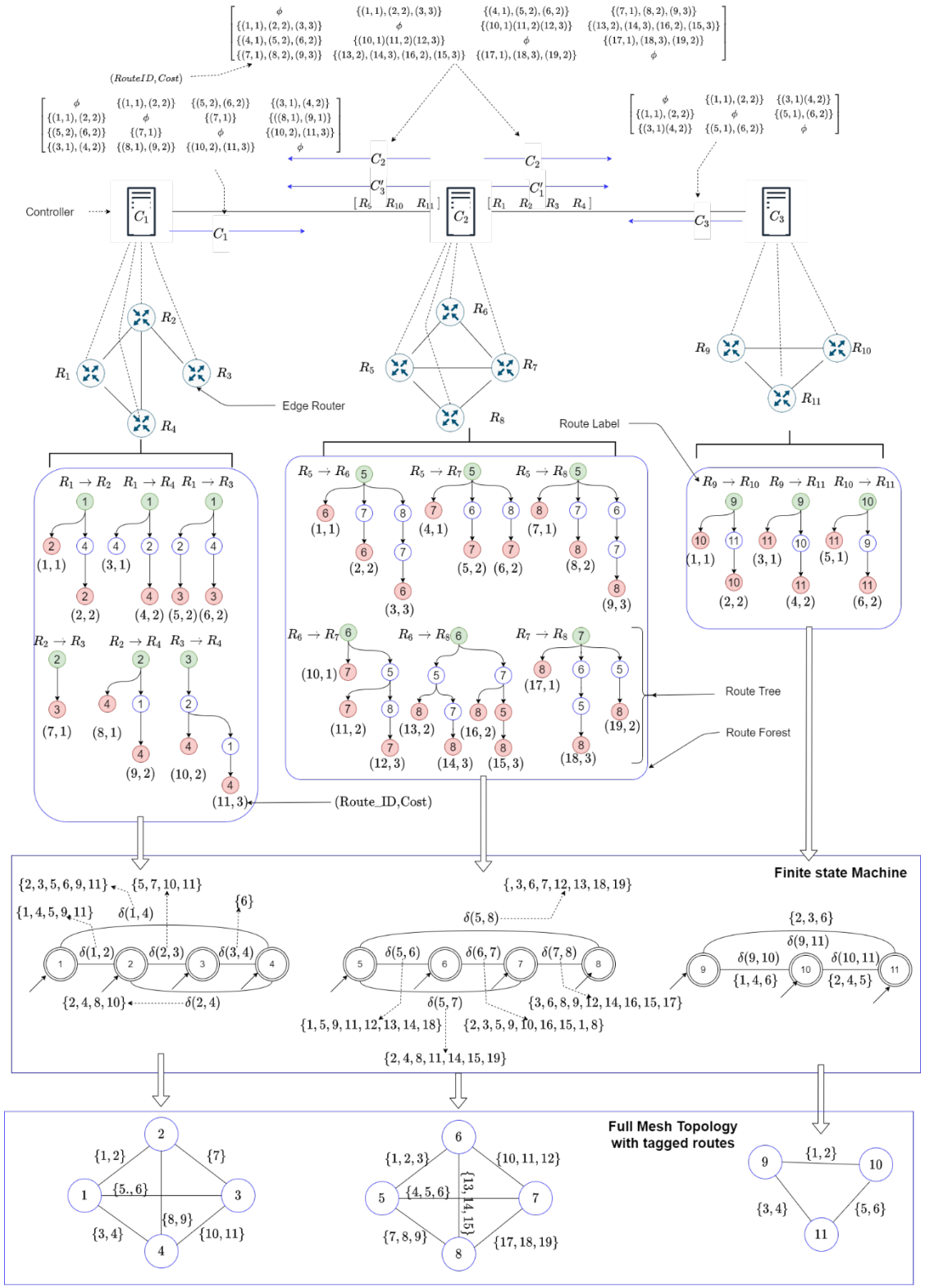

Figure 23 Complete process of computing all-paths for all-pair of nodes. First MRoute generates route trees $T_{s,d}$ for all pair of vertices that results route forest RF_i for every controller CON_i. Next, FSM compresses RF_i preserving the path information using Route_ID and finally full mesh graph $\mathcal{G}(V, \mathcal{E})$ is generated that maps RouteIDs into edge-set $\mathcal{E}$

- **Initialization:** *MRoute* initiates the process by creating an n-ary tree with v_d as root.
- **Recursion:** The $ADJ(v_k)$ the function returns the adjacent vertices of a node $v_k \in V$, and $ANSC(v_k)$ returns the list of ancestors that have already been visited along the branch where the intermediate node v_k belongs. That said, the recursive function could be described as follows (Eq. *18*).

$$MRoute(v_k, ANSC) = \begin{cases} \phi : v_k = v_s \\ \phi : ADJ(v_k) - ANSC(v_k) = \phi \\ MRoute(x, ANSC \cup \{v_k\}) \ \forall x \in ADJ(v_k) - ANSC(v_k) \end{cases} \quad \text{Eq. 18}$$

  Condition 1: Terminate with success if the source vertex is visited.

  Condition 2: Terminate with failure if no neighbour is left to visit.

  Condition 3: Recursively visit all the neighbours that are not visited, keeping the current node as an ancestor along the trail.

- **Termination:** The recursion terminates if one of the following cases is valid.
  - If the source vertex v_s is found on any of the branches, the branch satisfies the criteria of a path that starts with v_s and ends with v_d. As v_d is the root, backtracking the branch gives a unique path. Thus, the recursion is terminated.
  - If all adjacent vertices appear as ancestors, this means no more un-visited neighbours have left along with the subjected recursion. This condition terminates the recursion to prevent any loop or duplicate path discovery.
- **Optimization:** All descendent nodes leading to a non-source leaf are pruned out to optimize the space of the tree. This recursive removal is called Shrink-Phase.
- **Result:** Backtracking each branch of the route tree returns a set of paths between v_s and v_d, which *MRoute* (Algorithm *3*) returns.

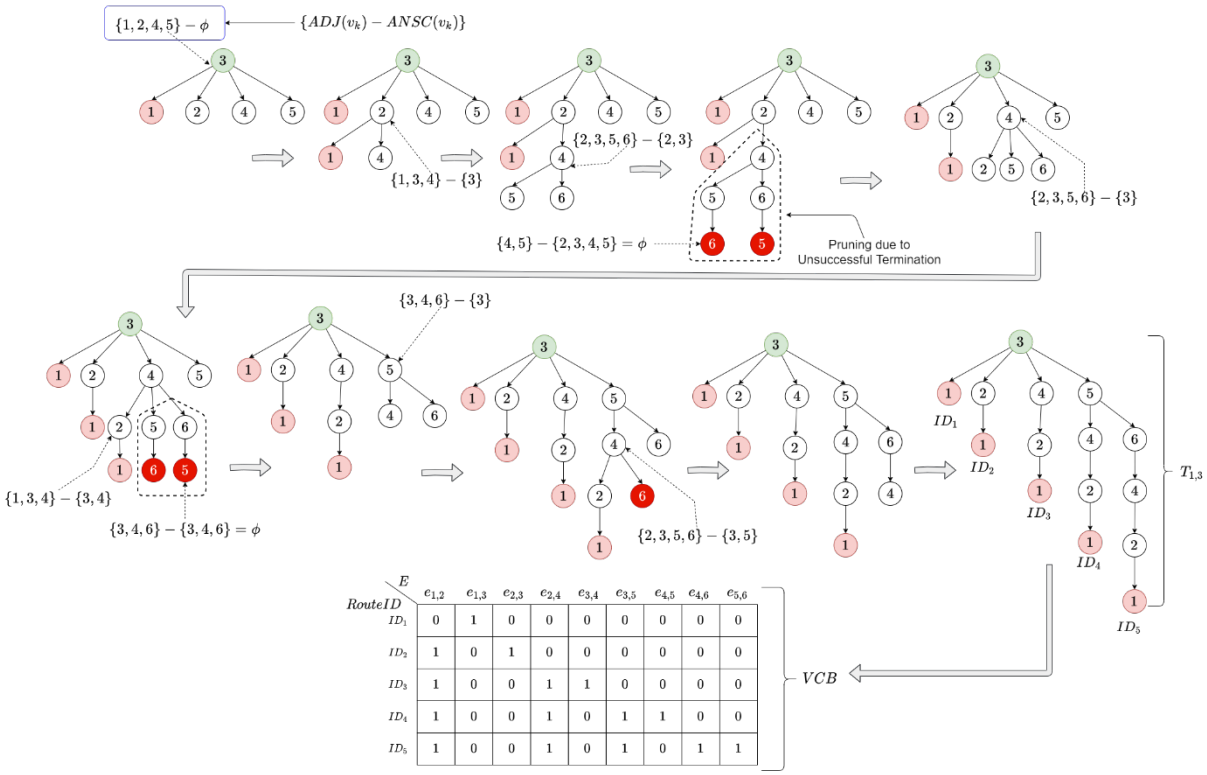

Figure 24 RouteTree generated by MRoute w.r.t. for $r_{1,3}$

3.3.3 Route Tree

The Route tree $RT_{s,d}$ is an m-way search tree that represents all paths between $v_s, v_d \in V$ with the following properties.

1. The destination vertex v_d is the root.
2. All leaves are the source vertex v_s
3. Every branch has a positive weight ($NCOST$) assigned as a sum of individual edge costs along the branch. STEN periodically calculates the edge cost.
4. For any intermediate vertex v_k, the function $ANSC(v_k)$ and $ADJ(v_k)$ return the ancestors along the branch and the adjacent nodes, respectively, for v_k. The base case for the recursion is $ANSC(v_k) \cap ADJ(v_k) = \phi$.

Algorithm 3**: *MRoute***

Purpose: Finds all possible paths between $(v_s, v_d) \in V^2$
Local Input: $v_s, v_d, v_k \in V$
Global Input: $ADJ, NCOST$
Output: $RT_{s,d}$
Data Structure: n-ary tree
Implementation: Dynamic Array, Implicit Stack
Strategy: Recursion, Backtracking

```
Begin
    if root = ϕ then
        root ← v_k
        if v_k = v_s 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
                    Call Update_Ancestors()
                    // Recur
                    MRoute(v_k, v_s, v_d)
                end loop
            end if
        end if
    end if
end
```

3.3.4. Topology Synchronization

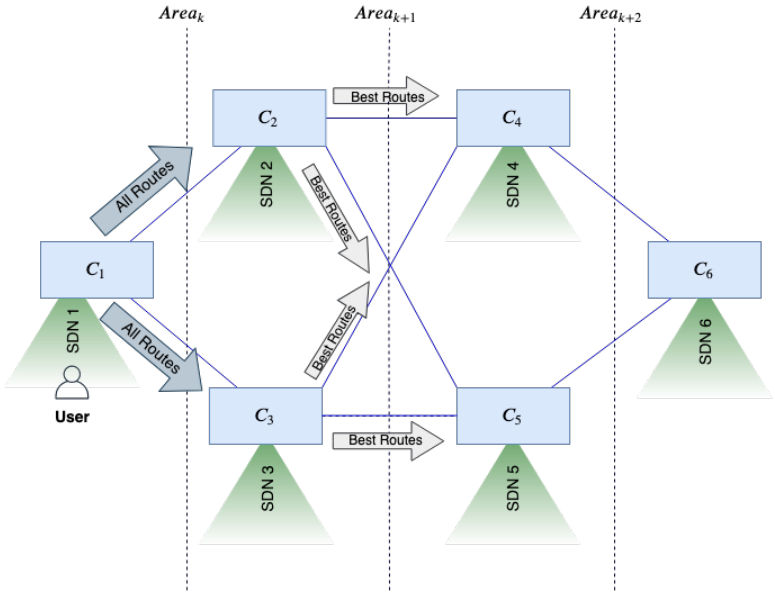

Figure 25 Controller network of distributed SDN

The previous section discusses the principles of *MRoute* that generates all possible parts between all pairs of vertices. The source and destination vertices belong to the source and destination classes, respectively. An SDN controller does this calculation for its local network. Controllers then share this information with their neighbours. In large distributed SDNs, this process may cause an overflow of controllers' memory. However, routes learned from remote controllers are only significant to routers with proximity. Therefore, we propose that the advertisement of locally known routes is restricted to the neighbouring controllers, and only the best path is advertised further. This results in limited flooding in the controller network and prevents overflow from the controllers' routing table.

Figure 25 depicts the controlled advertising of local routes across a distributed SDN. The user belongs to a consumer network managed by the controller by CON_1. Thus, CON_1 OBJ's neighbors CON_2 OBJ CON_3 OBJ. T $Area_k$ the SDN is partitioned into adjacent areas based on proximity. In General, OBJ $rea_{k-1}, Area_{k+1}$ A OBJ and $Area_{k-1}$ only the best route learned from $Area_{k+1}$ to OBJ and vice-versa.

3.3.5. Benchmarking

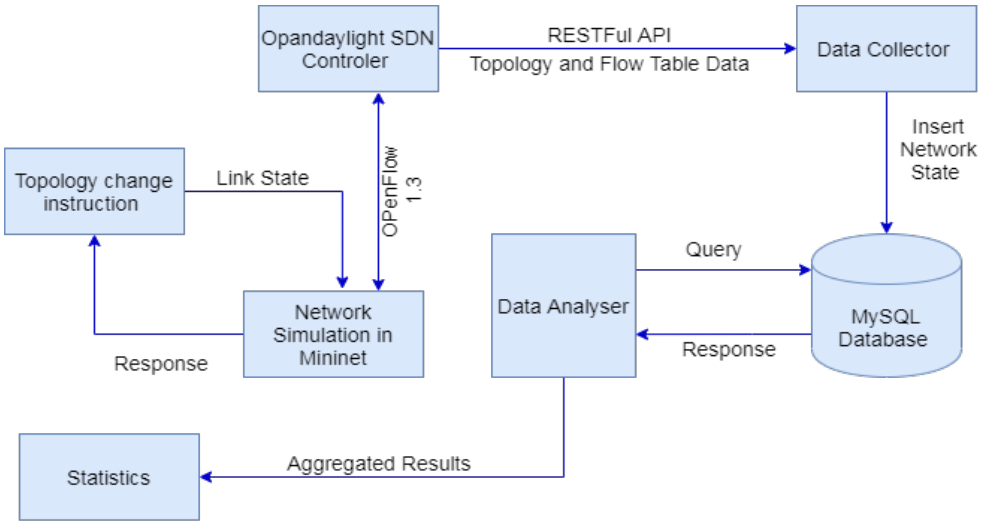

Figure 26 Workflow of the testbed

Figure 26 depicts the architecture and workflow using various open-source tools to develop the testbed. Table 8 lists them with their purpose and brief usage description. The workflow of the testbed is as follows. The testbed 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 an increasing number of nodes $[0 - 100]$ is fed into the emulator. Mininet 'talks' with a controller-cluster using OpenFlow. The controllers discover their downstream topology and feedback OpenFlow rules to the respective switches. OpenFlow rules are generated by translating the routes calculated by the routing engine. The script then starts disconnecting random links $[0 - 10000]$, from the topology, which invokes network re-convergence. Eventually, switches contact their upstream controller for a new rule. The controller contacts the routing engine for a new route. In the case of the proposed model, routes are pre-computed and ranked. This diminishes the need to enter the convergence process instead; it gives the following best Route on demand. The rapid-convergence feature of *MRoute* gives it an edge over its competitors. Several parameters (listed in the next section) are collected during this process, and further used for comparison and benchmarking.

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	• It Interfaces with an SDN network, simulated in Mininet using OpenFlow 1.3. • It Provides the topology and flow table information using RESTful API.
MySQL	Opensource Database	• Stores Node and link states varying over time

Table 8 Lists of open-source tools used to develop the testbed

3.3.6. Comparative Parameters

The experiment compares *MRoute* against OSPF and EIGRP, considering their wide acceptance in the enterprise networks with their respective routing 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 it compares with the proposed algorithm. The comparison benchmarks *MRoute* uses six parameters, namely,

1. **Discovery Time:** The average time 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, it is $O(n^2)$.
2. **Convergence Time:** The average time the algorithm takes to calculate an alternative path if the primary path fails. Since *MRoute* proactively pre-calculates all possible routes and maintains their dynamic rank, it is always guaranteed that the controller will reinforce it to the network instantly until there is at least one valid route. As a result, the network will converge in a constant order of time.
3. **Communication cost for Discovery:** Routing protocols use distributed computing models. To discover and monitor neighbours, they use the "Hello" protocol over Multicast. The number of control packets in *MRoute* is constant as all edge devices send 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 following best active route. Thus, no control messaging is needed.
5. **Space Consumption:** The amount of memory needed to maintain the topology information, including the data structures and look-up tables.
6. **Route Tree size:** The algorithm is inherent, exponential, yet deterministic. During the growing phase, the tree adds children and removes the invalid paths during the shrinking phase. The tree size growth is also tested to examine the temporal space complexity of the n-ary tree data structure.

3.3.6. Experimental Setup

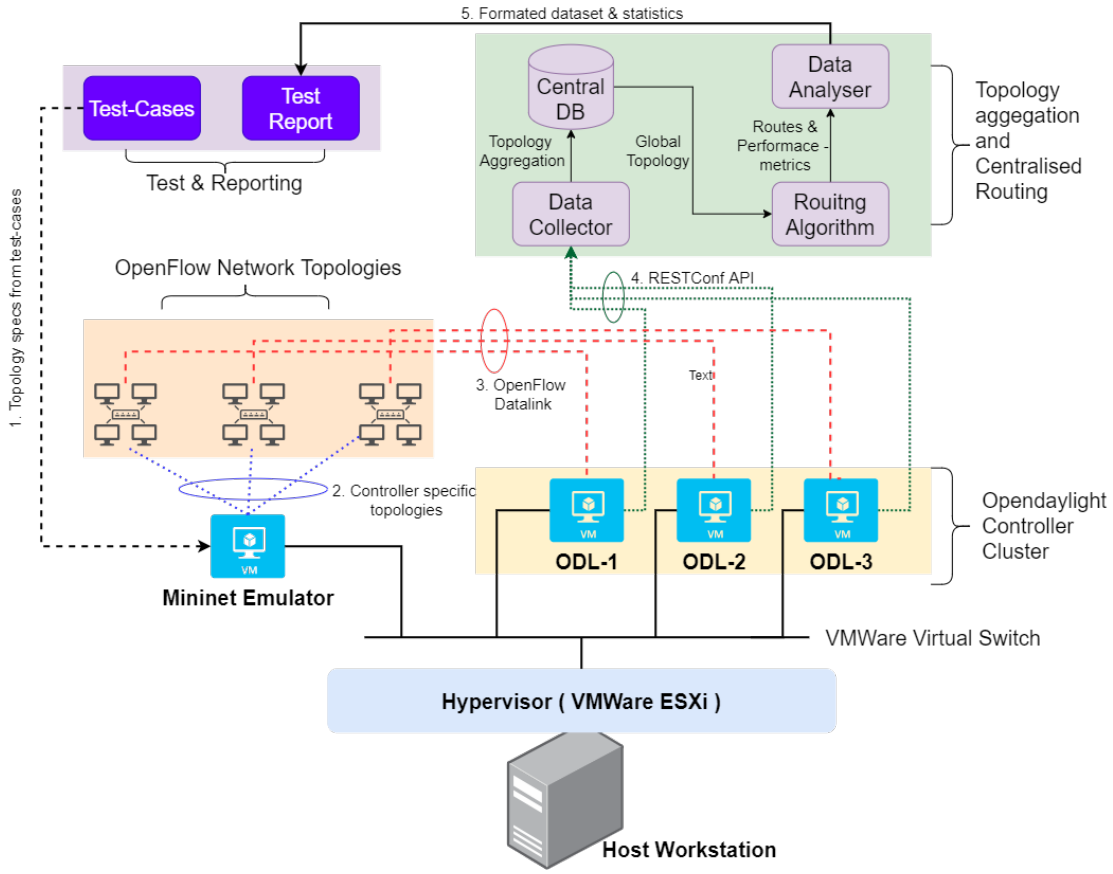

Figure 27 Deployment diagram of the experiment

Figure 27 depicts the detailed setup of the experiment and implementation of the testbed; the numbered events denote the workflow. The testbed comprises several virtual machines

designated for the Mininet 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 pushed into the Mininet instance. One iteration comprises five phases that terminate with a report summarizing parameters listed in the previous section. Each topology configuration creates 3 OpenFlow LAN networks; each represents an edge segment and is 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) correspond to their downstream topologies established with the OpenFlow datalink. Each controller maintains its downstream topology map and flow tables and exposes them using RESTConf API to the northbound using TCP port 8181. The data-collector module accumulates topologies flow tables from individual controllers in the application plane, which are then fused into a global topology (as described in Figure 27). The routing algorithm module executes SPF, DUAL and *MRoute* on the topology and returns benchmark information to the data-analyzer, which finally formats the comparison information in a CSV file and streams it to the Reporting module. We limit the benchmarking with a 3-Controller (each with 4-vCPUs & 8GB RAM) configuration. However, the same process is scalable to a larger configuration with adequate resources given. A clarification for the readers' comprehension of Controller-Cluster, The cluster configuration does not yield a controller aggregation (e.g., Akka [15] clustering) but rather a collection of multiple autonomous controllers.

[15] For more information about Akka clustering, visit https://bit.ly/3qshdWK

3.3.7. Experimental Results

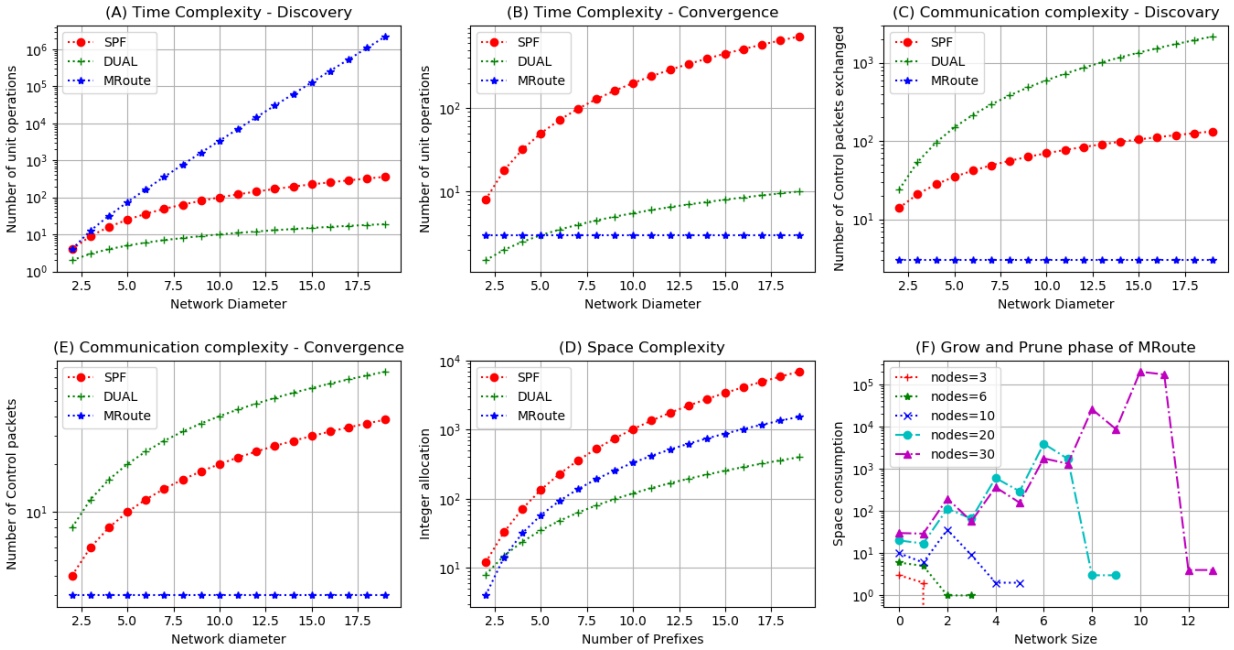

Figure 28 Experimental Results and Comparison MRoute against SPF and DUAL using the following parameters (A)Time Consumption to computing paths, (B) Time consumption to converge, (C) Control traffic for topology synchronization, (D) Space consumption for topology maintenance (E) Control traffic for convergence, (F) Route-Tree size.

The comparative analysis between *MRoute*, DUAL, and SPF (Figure 28) benchmarks algorithms using six parameters as discussed in section 3.3.5. Subplot (A) compares the time complexity concerning the size of the network. These results are plotted on a log scale. Therefore, *MRoute* shows exponential growth in comparison with DUAL and SPF, which are bounded above by $O(n^2)$. Due to the diffusion-computation model and the presence of a feasible successor. DUAL goes less deep into the convergence state than SPF. We tuned the SPF to run on each downstream topology in parallel, simulating a multi-area OSPF network. Although it seems initially that DUAL is the optimum than its competitors, 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 will never enter a re-convergence process, which is not the case with DUAL and SPF. This is shown in the subplot (B), where the random link failure causes SPF to re-converge every time. DUAL shows a better result as, in some cases, a feasible successor exists or a neighbour replies with a route much before the

query reaches the network boundary. However, *MRoute* shows a constant reading, as it is a $O(1)$ that requiring a fixed number of operations that involve querying and getting a reply for the following best route. The process can be thought of 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 the 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' messages. Since OSPF uses a link-state model, the total number of packets exchanged is higher than that of distance-vector-based EIGRP. *MRoute* is designed as a centralized routing algorithm; therefore, it does not exchange any discovery or update messages with other nodes. Instead, it updates only the controller, which is logically one hop away. This is justified by the subplot (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 exponentially. Although the pruning phase releases some memory, the overall growth remains exponential. The state model is built after the complete forest is generated, which compresses them into tables and relinquishes the memory (subplot (F)). The space complexity of *MRoute* sits between SPF and DUAL as OSPF maintains an identical link-state database for all nodes and EIGRP topology tables list the successor and feasible successors for each destination prefix depicted in subplot (F).

3.4. Most Reliable Route First (MRRF)

3.4.1. Problem formulation of Cognitive Routing

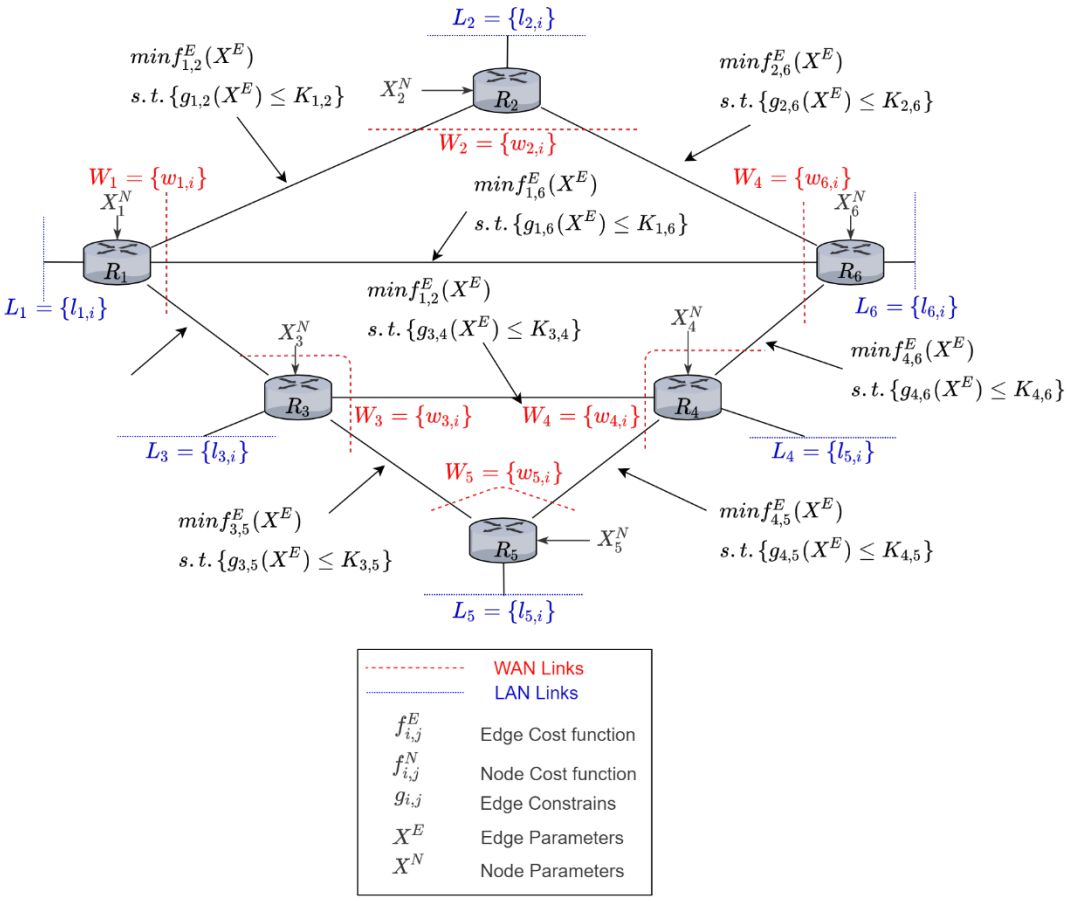

Figure 29 Reference topology with route-policies

The controller sees the topology of the SDN as a simple, finite, and connected graph. The network consists of programmable routers and switches connected to the Controller via a secure and reliable SBI. The controller treats both the router and switch as a generic 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.

1. EN doesn't exchange SP traffic among each other but only with the controller over the SBI.
2. There exists no Neighbor Discovery mechanism. EN shares their local information and keep-alive packets with the controller only.

3. The controller can monitor information about various resource utilization of the EN such as Memory, CPU, Network interface, etc.
4. The network topology does not change frequently.

Each EN_i maintains a local Routing Table (RT_i) comprising 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 learned from the controller. 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 Node ID like Router ID in OSPF and EIGRP and maps it with their corresponding CR set. When an EN receives a packet with an unknown destination address, it forwards it to the controller. The controller then resolves the destination node's ID from a map, finds a route between the source and destination router, and replies to it back to the source node.

Figure 29 depicts a reference topology of 6 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 RFC-1918 [199]. The controller uses the Link-State Routing (LSR) approach to build a topology from this information, i.e., nodes with a shared WAN network are adjacent. However, for the sake of simplicity, we did not include topology with Broadcast segments as it requires additional Designated node placement. 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, reliability, etc. The WAN links are constrained and heterogeneous, i.e., their attributes are bounded above by some pre-defined values specific to that link. These values generally depend on the network policy or the media type; hence we leave it user-defined. We propose the formulation of link-cost as a set of linear programming Problems, for individual edges, with a

linear cost-function $f_{i,j}^{E}: X^{E} \rightarrow \mathbb{R}^{+}$, between R_i and R_j, such that its linear constraints $g_{i,j}(X^{E}) \leq K_{i,j}$ are met. This is to overcome the limitation of OSPF's sub-optimal 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[24]. However, as the metrics are calculated locally to the controller, it diminishes the need to exchange update packets between edge nodes, thus eliminating the cause of route-flapping. Like the edge cost, the node cost also contributes to the calculation of the final metric. The node-cost function $f_i^N : X^N \rightarrow \mathbb{R}^{+}$ computes a cost based on the node attributes (X^N).

The controller generates a Graph structure isomorphic to the network topology and weighs its edges by relaxing the f_i^N and f_j^N into $f_{i,j}^{E}$ for all adjacent for all R_i, R_j using STEN. As the X^E and X^N varies over time, but the topology remains the same; hence the subjected Graph is a dynamic isomorphic Graph, which we refer to as Meta-Graph.

The proposed algorithm performs the following steps to meet the rapid-convergence criteria.

1. Efficiently computes all possible paths between all pairs of nodes from the meta-graph using *MRoute*. This step is invoked whenever the topology changes.
2. Computes the reliability of the links by profiling their cost variation over time using an RNN using LSTM; This is a periodic step. As the LSTM estimates a time series by autoregression.
3. 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.
4. Returns the most reliable routes on-demand as the primary route keeping the rest in a backup. In case the primary Route fails, the following best Route is served instantly. Hence rapid convergence is achieved.

The Simple, undirected and connected graph $G(V, E)$ represents the topology of the underlying network; where, $V = \{v_i\}$ and $E = \{e_{i.j} \,|adj\,(v_i, v_j)\}$ be the Vertex and Edge set, respectively. V and E are finite and non-empty, $adj(v_i, v_j) = 1$ if v_i, v_j are adjacent, and 0 otherwise. The graph is simple (No self-loop, no parallel edge) to fit 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:** $ADJ(G) = [adj(i,j) \in \{0,1\}]^{n\times n}$ is a symmetric binary matric represents the adjacency of the graph $G(V, E)$ and $|V| = n$.
2. **Policy Set:** A finite non-empty set of policy tuples that includes $f_{i,j}^{E}$ and $\{g_{i,j} \leq K\}$. The policy set (Eq. *19*)

$$PLC = \{< f_{i,j}^{E}(X^E), \{g_{i,j}(X^E) \leq K_{i,j}\} > \forall (i,j) \in V^2\} \quad \text{Eq. 19}$$

3. **Variable Cost Matrix (VCOST):** $VCOST(t) = \left[c_{i,j}(t) \in \mathbb{R}^+\right]_{n\times n\times t}$ is a tensor representing the cost matrix at time instance t. All the n diagonal values $c_{i,i}$ represents corresponding node-costs f_i^N and the non-diagonal ones represent the edge-cost $f_{i,j}^{E}$ for all valid edges i.e. $(i,j) \in E$.(Eq. *20*)

$$\{c_{i,j}(t)\} = \begin{cases} min\, f_{i,j}^{E}(X^E, t), & if\ i \neq j\ \forall (i,j) \in E \\ f_i^N(X^n, t), & Otherwise \end{cases} \quad \text{Eq. 20}$$

4. **Normalized Cost Matrix (NCOST)**: As the diagonal elements of $VCOST\ (t)$ represent weighted self-loops, it violates the "simple-graph" criteria. Therefore, a normalization is needed that relaxes the self-loops but preserves their effects on the resultant "Simple-

Graph." We use the Stochastic Temporal Edge Normalization (STEN) technique to do so, which results. $NCOST(t) = \left[\{c'_{i,j}(t)\} \in [0,1]\right]$.

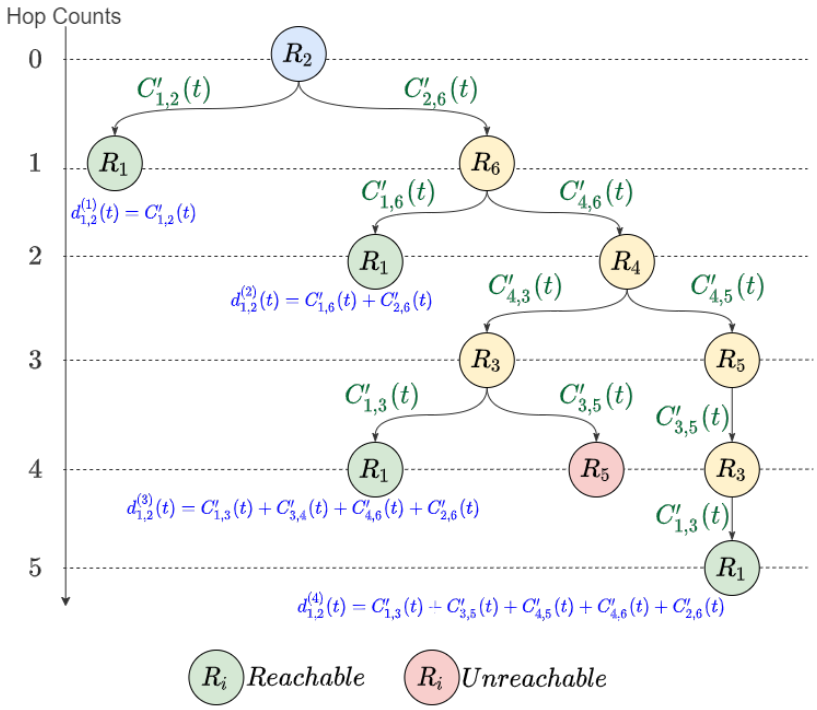

Figure 30 RouteTree of $RT_{1,2}$, rooted at R_2 all the reachable paths terminate with R1 and unreachable node R_5.

5. **Route Tree**: The *MRoute* algorithm generates the tree and is discussed in section 3.3. Figure 29 depicts the Route-Tree $RT_{1,2}$ w.r.t. the reference topology Figure 29 shows the hop counts and cumulative costs for each valid route (terminating at source vertex v_1). At $hopcount = 5$, R_3 has two children, R_1 and R_5 and R_1 is the source, 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 (Grow Phase), where the tree grows recursively, registers several unsuccessful terminations, and Phase-2 (Shrink Phase) eliminates all such branches.
6. **Route Forest:** For an n-node graph, there exists $\binom{n}{2}$ possible pairs (from the Handshaking theorem in Graph Theory) of nodes. Each node produces a Route Tree. A collection of such trees forms a Route-Forest. 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.

3.4.2. Metric formulation

The proposed composite metric for *MRoute* constitutes the node cost $C_i^N(t)$ and edge costs $C_{i,j}^N(t)$. The node and edge parameters are listed in Table 9, followed by the formulation of costs.

Node Parameters ($\mathbf{X^N}$)	**CPU**	Parameter	Core Count (n_c)	frequency (f_c)	Utilization (u_c)	
		Units	Integer	MHz	[0,1]	
	Memory	Parameters	Volume (v_m)	Frequency (f_m)	Utilization (u_m)	
		Units	MB	MHz	[0,1]	
Link Parameters ($\mathbf{X^E}$)	Parameters	Bandwidth (BW)	Delay (DLY)	Load (LD)	Reliability (RLY)	MTU
	Units	Mbps	ms	[0,1]	[0,1]	[0,1500]

Table 9: Link and Node Parameters, Monitored By CP

Formulation of the Node Cost: The node cost uses CPU and memory utilization as parameters. However, CPU & memory utilization can solely determine performance (i.e., a 20% utilized 8-core CPU processes more operations than an 80% single-core CPU, which 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, yielding 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. (Eq. 21)

$$C_i^N(t) = f_i^N(X_i^N, t)$$
$$= \left[\alpha_c\left(f_{c_i}(t) * n_{c_i}(t) * u_{c_i}(t)\right) + \alpha_m\left(f_{m_i}(t) * v_{m_i}(t) * u_{m_i}(t)\right)\right] \quad \text{Eq. 21}$$
$$such\ that, \alpha_c + \alpha_m = 1;\ given, \alpha_c = \alpha_d = 1\ as\ default\ \forall \alpha_c, \alpha_m \in [0,1]$$

Formulation of the Link Cost: The link cost function uses the same parameters as EIGRP. 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. The topology is built inside the controllers' memory, and no control packets are flooded to make a neighborship. The SDN paradigm unifies the benefits of OSPF and EIGRP as it creates a complete topology view like OSPF, uses all parameters of a more robust composite metric and supports unequal-cost load balancing like EIGRP.

The formulation in Eq. 22 has three components.

1. **BDP**: The Bandwidth Delay Product $BDP(t) = BW(t) \times DLY(t)$ measures the instantaneous end-to-end throughput.
2. **Load**: The BDP is scaled by the mean load $\left(occupancy = BDP(t) \times MLD(t)\right)$ and measures the amount of occupancy in the link.
3. **Reliability**: The occupied capacity is scaled with the additive inverse of reliability $\left(occupency \times \left(1 - RLY(t)\right)\right)$ to measure the unreliability of the occupied capacity.

$$C_{i,j}^{E}(t) = f_{i,j}^{E}\left(X_{i,j}^{E}, t\right) = \left[\beta_L MLD_{i,j}(t) \times \left\{\beta_B Min_{i \in path(i,j)}\left(BW_i(t)\right) \times \beta_D \sum_{i \in path(i,j)} DLY_i(t)\right\} \times \beta_r\left(1 - RLY_{i,j}(t)\right)\right] \quad \textit{Eq. 22}$$

Such that,
$\sum \beta_k = 1; given\ \beta_k = 0.25\ as\ default\ \forall k \in \{B, D, R, L\}\ and\ \forall \beta_k \in [0,1]$
$MLD_{i,j} = \frac{TLD_{i,j} + RDL_{i,j}}{2}$ *is the mean load across the end-to-end link*
$Min_{i \in path(s,d)}\left(BW_i(t)\right)$ *is the effective throughput of the bottleneck link*
$\sum_{i \in path(s,d)} DLY_i(t)$ *is the cumulative delay along the path.*

Formulation of Normalized Metric: Concerning equations Eq. 20 and Eq. 21, the cumulative metric for a link $c'_{I,j}(t)$ is obtained by relaxing the node costs of both endpoints $\left(C_i^N(t), C_j^N(t)\right)$ and scaling them by their corresponding load-share $\left(P_{i,j}(t), P_{j,i}(t)\right)$ into the link cost $C_{i,j}^E(t)$ as shown in Figure 31 The parameters γ_N, γ_E are the weighing factors set by the user. The load-share of an interface is a proportion of the number of packets passed through that interface over the total Packet exchanged. The value is expressed in [0,1]. (Eq. 23)

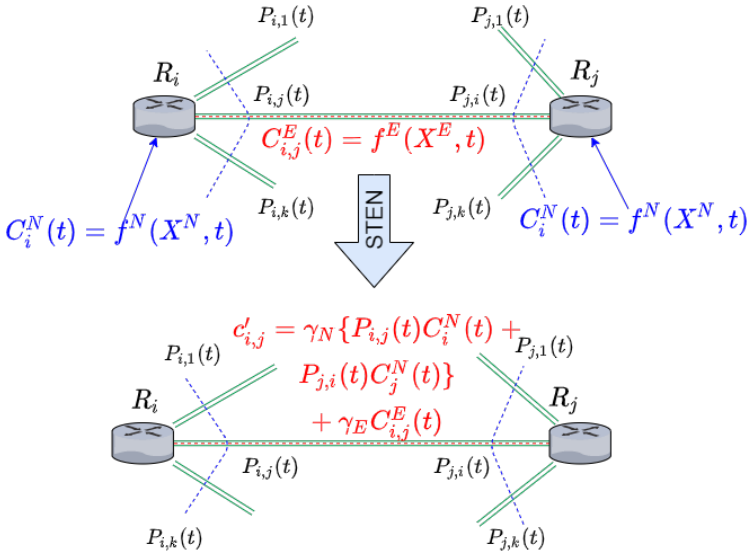

Figure 31 Relaxation of Node costs into Edge using STEN

$$c'_{i,j}(t) = \left[\gamma_N\left(P_{i,j}C_i^N(t) + P_{j,i}\,C_j^N(t)\right) + \gamma_E\left(C_{i,j}^E(t)\right)\right]$$
$$such\ that, \gamma_N + \gamma_E = 1;$$
$$with\ \gamma_N = \gamma_E = 0.5\ as\ defalt\ \forall \gamma_N, \gamma_E \in [0,1] \qquad \text{Eq. 23}$$

Figure 31 depicts the relaxation process to calculate the variable cost metric $c'_{i,j}(t) \in VCOST$ at time t.

3.4.3. Analysis and Optimization of *MRoute* algorithm

MRoute takes to source and destination vertex (v_s, v_d) as input, looks up to global structures ADJ and $NCOST$ during its recursive run-time and returns a route tree $RT_{s,d}$. The n-ary tree is stored in a hashed-dynamic array structure. It finds all possible paths between a pair of vertices using the 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 are further explained in the later section of this chapter.

Optimizing the Route-Tree Data structure

MRoute adds nodes recursively into the Route-Tree, the algorithm assumes $ANS(v_k \in V)$ is of $O(1)$. Generally, an n-ary tree can be stored using either a linked (non-contiguous) 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 is the network mesh. Additionally, the recurrence decreases monotonically as more neighbours are visited. They would not appear as children. Therefore, the number of children decreases as the tree gets deeper and choosing an n-ary tree structure is not space-optimal.

We propose an optimal data structure to accommodate such a sparse array. Furthermore, when a graph is converted into a 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 (Figure 32).

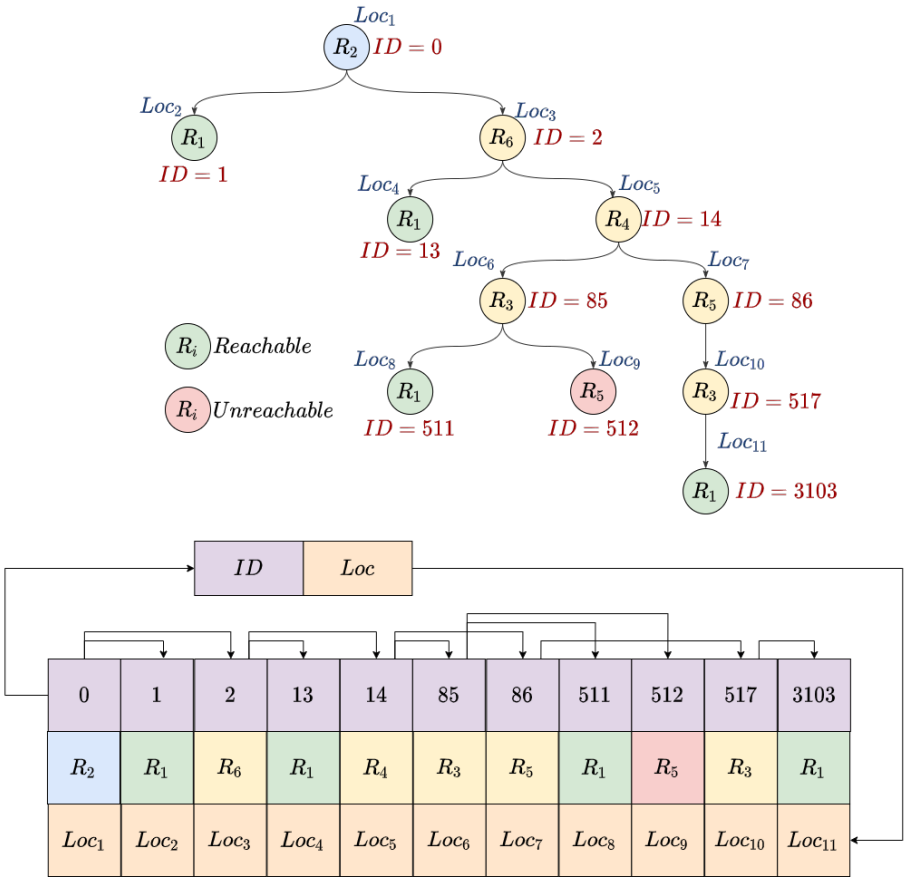

Figure 32 Dynamic Array-list with hash-table organization for fast searching. Loc_i is the virtual memory location, that holds the router object R_j with ID k. Hash table maps an ID to its location

An efficient and light index generation method is needed. For an n-ary tree, the following Eq. 24 generalized heap-indexing rule is adapted for this purpose.

$$(index(v_k) = i) \Rightarrow \begin{cases} parant(v_k) = \left\lfloor \frac{i}{n} \right\rfloor \\ child_j(v_j) = ni + j \end{cases} \quad given, index(root) = 0 \; and \; |V| = n \qquad \text{Eq. 24}$$

A non-contiguous data structure stores the nodes for better scalability to avoid any segmentation error while using large topologies. Nodes are kept in random memory location Loc_k. The ID is calculated using rules in Eq. 24 and is kept along with the nodes' data. A hash table maps the index to location; thus, the search time is reduced to $O(1)$; Figure 24 depicts the process.

Optimizing Route-Forest Formation

MRoute is a costly algorithm in terms of space consumption while generating a Route-Forest. The algorithm is invoked $O(n^2)$ Times. The calculation of the route tree for any arbitrary pair of nodes is computationally independent since they share a common data structure ADJ and $NCOST$. This satisfies the criteria to execute them in parallel without any race condition (as no write operation on global structures occurs). Therefore, each $RT_{i,j}$ $\forall (i,j) \in V^2$ is computed parallelly in their threads. Also, $T_{i,j}$ can also be realized by reversing $T_{j,i}$ with $O(n^2)$.

FSM model and Route-tagging

As the Route Tree grows exponentially, a compression algorithm is necessary to keep it scalable. We propose a novel approach to achieve such by using a Finite State Machine (FSM) or Type-3 automata, which eventually generates regular expressions to identify a route uniquely. That said, an n-node FSM consumes $O(n)$ space for storage using a matrix format; thus, it also leads to an $O(1)$ access tine. Hence, we chose to leverage the FSM model for compressing Route-Trees.

Let, $\mathcal{M}(\mathcal{Q}, \mathcal{T}, \delta, q_0, \mathcal{F})$ be a pentuple describing an FSM such that,

- $\mathcal{Q}$ is a finite, non-empty set of states ($\mathcal{Q} = V$)
- $\mathcal{T}$ is a finite, non-empty set of unique route identifiers (Route-Tags), ($\mathcal{Q} \cap \mathcal{T} = \phi$)
- δ is a transition function, such that.? $\delta: \mathcal{Q} \times \mathcal{T} \rightarrow \mathcal{Q}$
- q_0 is the initial state, $q_0 \in Q = v_s \in V$
- $\mathcal{F}$ is a finite, non-empty set of Final stets(s), $\mathcal{F} = \{v_d\} \subseteq Q$

Any Route Tree has unique paths between roots and leaves. An identifier called Route-Tag tags each Path uniquely. This compresses the exponentially large Route-Tree into a state machine of size $O(|V|)$. We term this transformation $\mathcal{F}: RT_{s,d} \rightarrow \mathcal{M}_{s,d}$ as Route State-Transformation Function (RSTF) and $\mathcal{M}_{s,d}$ as Route State Graph (RSG), depicts the transformation with changes in the data structures. Table 10 shows the transition function from the state machine (Figure 33).

$\mathcal{Q} \times \mathcal{T}$	**1**	**2**	**2**	**4**
R_1	ϕ	ϕ	ϕ	ϕ
R_2	R_1	R_6	R_6	R_6
R_3	ϕ	ϕ	R_1	R_1
R_4	ϕ	ϕ	R_3	R_5
R_5	ϕ	ϕ	ϕ	R_3
R_6	ϕ	R_1	R_4	R_4

Table 10: *Transition table of $\mathcal{M}_{1,2}$ rows represent routers receiving packets with route-tag represented by columns, cells represent the corresponding next hop and ϕ means empty set.*

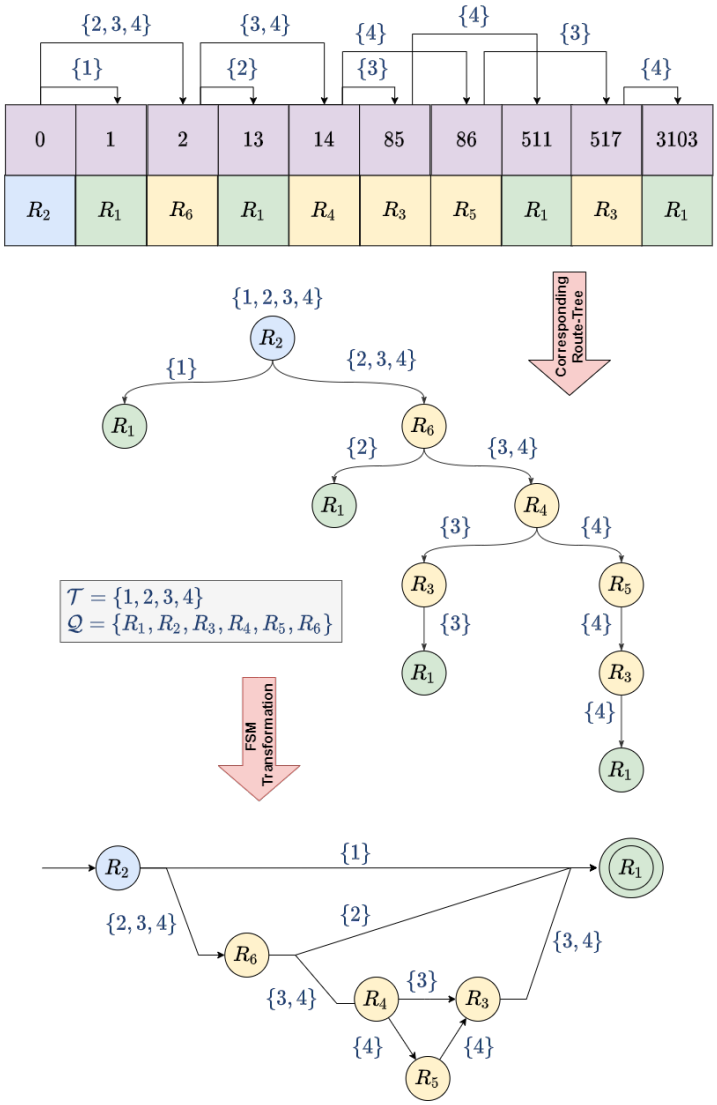

Figure 33 Implementation of Route-Tag and generating FSM form route tree. The process depicts the transformation of data-structures from the Route-Tree to Route State Graph

Path Matrix

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

1. **Encoding as Grammar:** In this approach, the state machine is encoded into a set of production rules called grammar $\mathcal{G}(\mathcal{V}, \mathcal{T}, \mathcal{P}, \mathrm{s})$. 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 were,
   - $\mathcal{V}$ is it a set of non-terminals?
   - $\mathcal{T}$ is a set of the terminal (Route-Tags)
   - $\mathcal{P}$ is a set of Regular production rules.
   - s is the start symbol.

As an example, $\mathcal{G}_{1,2}$, be the grammar corresponding to $\mathcal{M}_{1,2}$which is expressed in Eq. 25 .

$$
\begin{aligned}
\mathcal{P} = \{ & \mathrm{R}_1 \rightarrow \epsilon, \\
& \mathrm{R}_2 \rightarrow \tau_{1,2}\mathrm{R}_1 \mid \tau_{2,6}\mathrm{R}_6 \mid \tau_{1,2}, \\
& \mathrm{R}_3 \rightarrow \tau_{1,3}\mathrm{R}_1 \mid \tau_{1,3}, \\
& \mathrm{R}_4 \rightarrow \tau_{3,4}\mathrm{R}_3 \mid \tau_{4,5}\mathrm{R}_5 , \\
& \mathrm{R}_5 \rightarrow \tau_{3,5}\mathrm{R}_3, \\
& \mathrm{R}_6 \rightarrow \tau_{1,6}\mathrm{R}_1 \mid \tau_{4,6}\mathrm{R}_4 \mid \tau_{1,6}, \\
& \tau_{1,2} \rightarrow 1, \\
& \tau_{1,3} \rightarrow 3 \mid 4, \\
& \tau_{1,6} \rightarrow 2, \\
& \tau_{2,6} \rightarrow 2 \mid 3 \mid 4, \\
& \tau_{3,4} \rightarrow 3, \\
& \tau_{3,5} \rightarrow 4, \\
& \tau_{4,5} \rightarrow 4, \\
& \tau_{4,6} \rightarrow 3 \mid 4 \}
\end{aligned}
\qquad \text{Eq. 25}
$$

Encoding RSG into its grammar summarizes the routes, and parsing-ability is enforced using regular expressions.

2. **Encoding as The-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 it is the corresponding edge set. The column-sum tells how many route tags are sharing a given edge (typically used for load-balancing). The Tag-Cost function is formulated in Eq. *26* and the Tag-Cost table in Table 11. 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.

$$c_{s,d}^{(t_k \in \mathcal{T})}(t) = \sum_{(i,j) \in E} \left(c_{i,j}(t) \times TCT[t_k]\right) \qquad \text{Eq. 26}$$

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

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	$c_{1,2}^{(1)}(t)$
2	0	0	1	1	0	0	0	0	$c_{1,2}^{(2)}(t)$
3	0	1	0	1	1	0	0	1	$c_{1,2}^{(3)}(t)$
4	0	1	0	1	1	1	1	0	$c_{1,2}^{(4)}(t)$
Share	1	2	1	3	2	1	1	1	

Table 11 tag-cost-table for $\mathcal{M}_{1,2}$

3.4.4. Estimation of the Reliability using Recurrent Neural Networks (RNN)

As the normalized costs matrix ($NCOST$) varies over time (due to 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 separate time series. Then, unlike predicting the traffic pattern or load, we focus more on predicting the trend. One of the challenges regards online training in a dynamic environment. A trained neural network often rejects to adapt to sudden

changes as the 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 [200] is a widely used metric in portfolio management that measures the volatility of a stock and estimates the risk associated with it[201]. 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 (Eq. *27*).

$$\mathcal{V}_{i,j}(t) = \frac{\overline{C'_{i,j}([t-w:t])}}{SD\big(C_{i,j}([t-w:t])\big)} \forall (i,j) \in E \qquad \text{Eq. 27}$$

2. Estimate the edgewise hypothesis functions $h_{I,j} \in \mathcal{H}$ as an auto-regressive function using an RNN with a period W[16]. (Eq. *28*)

$$\mathcal{V}_{i,j}(t+1) = h_{i,j}\big(\mathcal{V}_{i,j}[t-w:t]\big) \qquad \text{Eq. 28}$$

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 then uses online training to update it. We define a cutoff value $\epsilon > 30\%$.

[16] In the theory of time series analysis, the optimal window size is when the Partial Auto Correlation Function (PACF) intersects the lag axis; after which the data at the time series does not contribute to the approximation.

3.4.5. Implementation

Figure 34 depicts the deployment diagram of our testbed. A multi-tier approach is conceived for operational and functional segregation. The SDN philosophy of decoupling control and data plane has been the proposed architecture's core design principle. However, the Knowledge plane has been integrated on top to support SON capabilities.

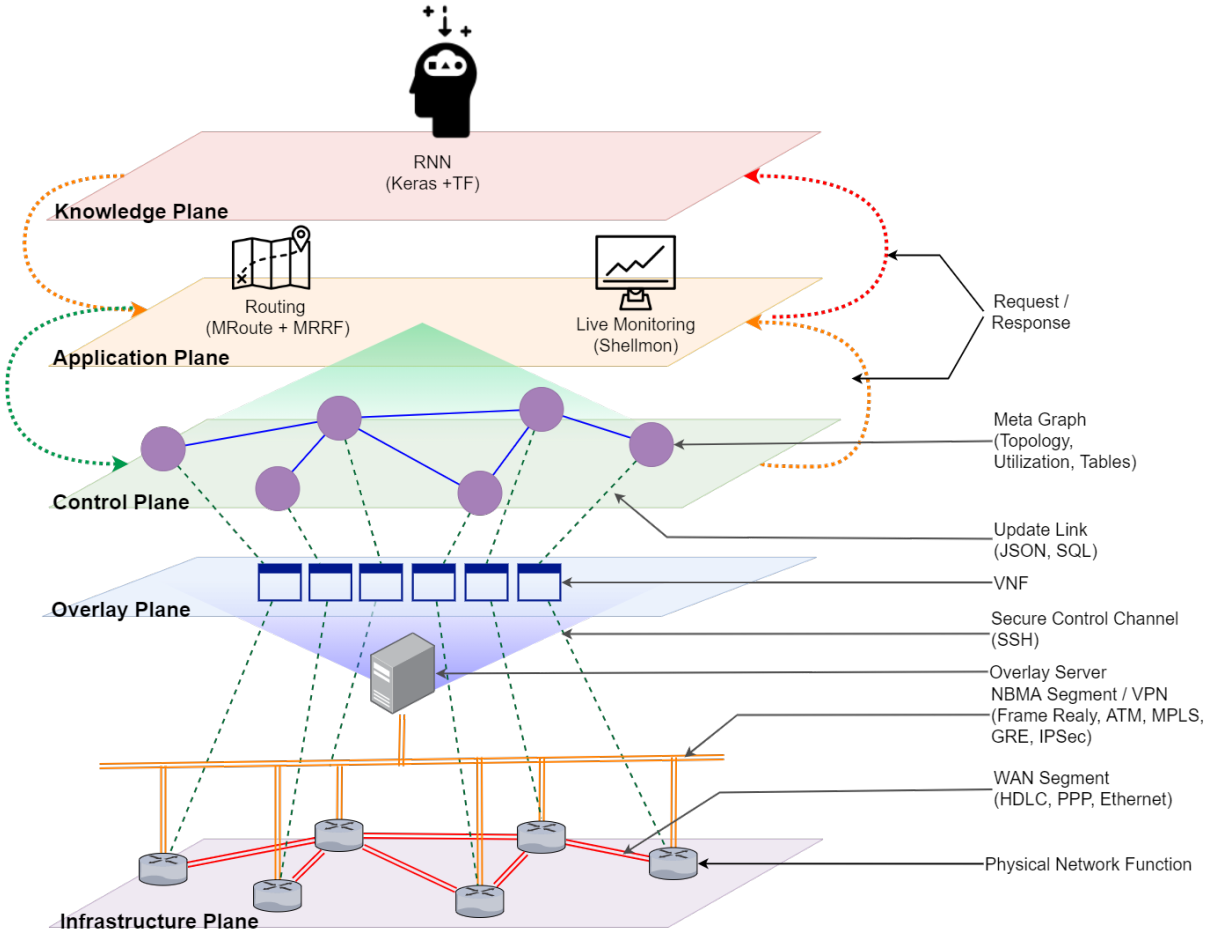

Figure 34 Deployment diagram of the Testbed. Infrastructure plane holds routers, overlay server receives monitoring information and spawns VNFs per Router. Control planes discover topology and application plane operates on it. Knowledge plane is for self-learning however beyond the scope of the context.

A. Self-Organized Knowledge Defined Network (SO-KDN) testbed

The architecture supports network automation and SON. It finds the 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 keeping updating them over time (Self-Healing). Thus, it meets all three criteria of SON. The following explains the working of the layers. Please refer to the implementation details, including connection API and algorithm code, for further information [46].

1. **Infrastructure Plane:** This layer hosts physical and emulated network nodes (e.g., routers, L2/L3 switches, etc.). In this experiment, Cisco IOU-L3 routers are simulated on GNS3. Routers are also connected to Overlay-plane securely using IPsec-DMVPN to exchange control traffic, like the OpenFlow channel.
2. **Overlay Plane:** This layer interfaces between the infrastructure and control plane. A VNF process (agent) is spawned for each router underneath, which maintains a secure link (using SSH) to monitor the resource utilization. Additionally, it also injects configuration commands. We use Napalm[17], library to automate the routes. Remote routes are injected as floating-static routes, and their priorities are controlled with respective administrative distances.
3. **Control Plane:** Resource and topology information are fused to generate the meta-graph in the control plane. REST-Conf is used to interface with the overlay plane below and the application plane above.
4. **Application Plane:** The application plane brings modularity to the architecture, hosting various apps that govern the use-cases' functional characteristics. *MRoute* is one such

[17] For more information on Napalm visit https://napalm.readthedocs.io/en/latest/

application. However, there are other functions such as migration and monitoring beyond the scope of the context of this chapter.

5. **Knowledge Plane:** The knowledge plane leverages the KDN paradigm. This component takes care of all data pre-processing, and 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.
    a. **Pre-processing:** This acts as a staging area for the model of the training units. It performs data acquisition, data quality checks, and validations, imputing and standardization. Typically, 70% of the overall process time is spent on this phase.
    b. **Offline training:** After the pre-processing tasks, the offline activity starts by dividing the data into training, validation, and testing for the machine learning (ML) model. It utilizes the historical data from the repository to train the model and predicts the networking characteristics to produce decisions such as VNF placement and state prediction.
    c. **Online Training:** It is used when the data is generated in a sequence (such as time series). Network resource utilization is a form of a time series. The Topology is represented as a matrix. Each element of the matrix represents a normalized link cost between a pair of nodes. Over time, a sequence of such matrices is received, making it a $n \times n \times t$ tensor. Where n be the number of nodes and t be the time.
    d. **Modelling:** The learning algorithm learns from the fed dataset and generates a prediction model. Since the problem can be classified as a time series prediction type, RNN is chosen as the base architecture.

B. RNN Architecture

In this section, the design of the machine learning function is presented. We also introduce a few techniques used, like hyper-parameters, fine-tuning, and choosing the best optimization algorithm.

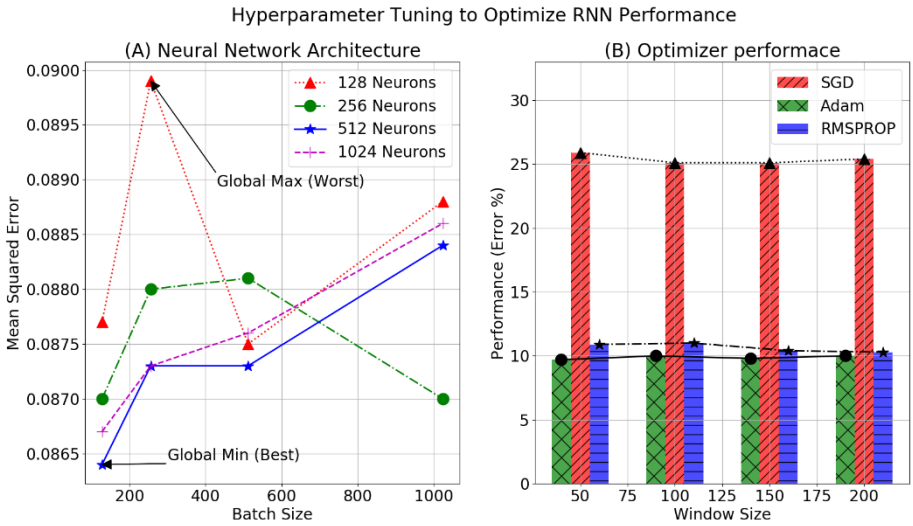

Figure 35 (A) Comparison of accuracy (by mean squared error) with four network setups (128, 256,512 & 1024), the Global optima is reached with 128 Neuron at a batch size of 512. (B) compares three optimizer algorithms (SGD, Adam & RMSPROP), over a varying window size of [20 − 200], on which Adam gives best result on average

1. **Hyper-Parameter Tuning:** In this phase, the Hyper-parameters such as Batch-size and number of neurons are tuned from experimental data. Figure 35 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 datasets; the error rate is measured using Mean Squared Error (MSE). As highlighted in bold, the optimum hyper-parameters have been 128 neurons and 512 batch sizes at 0.08 MSE.

2. **Optimization Algorithm:** Figure 33 compares the various optimizers. For the LSTM model, different sets of window sizes are tested. Three principal variants of Gradient Descent (SGD, ADAM & RMSPROP) are compared. As a proof of concept, results

show that predicting a 200ms 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 reliability can be estimated with the appropriate hyper-parameters with a mean of 90% accuracy.

C. Online Learning

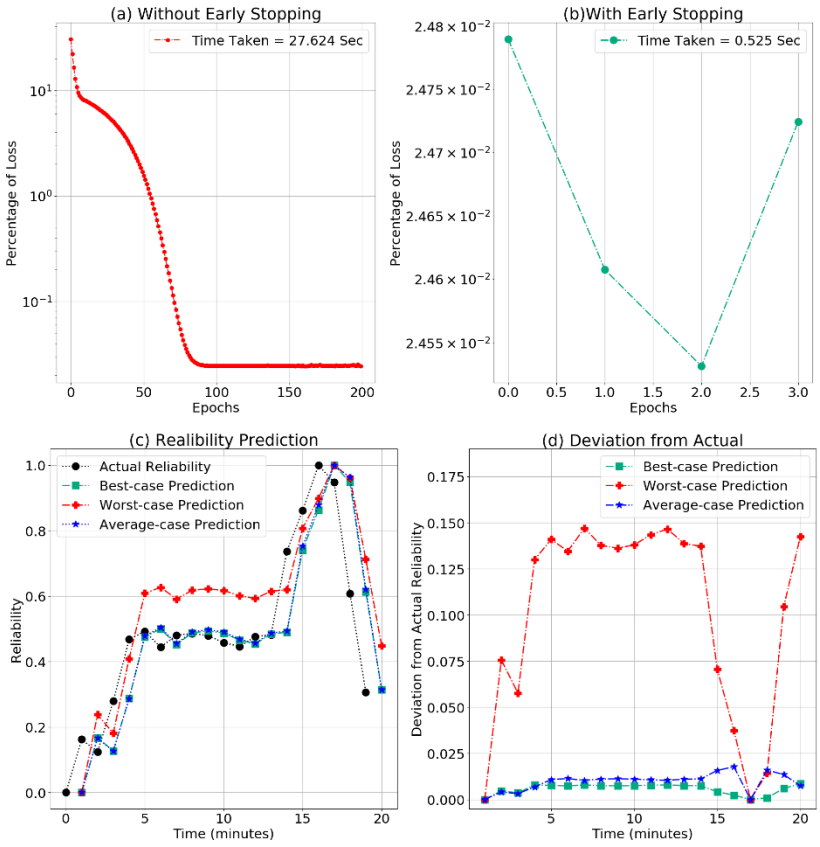

Figure 36 Evaluation of the Online-Learning, (a) Learning time with 200 epochs, (b) Accelerated learning with Early-Stopping enabled (c) Comparing time-series prediction of reliability in Best, Average and Worst-case scenario (d) compares the deviation in log-scale, also shows the comparison is distinctive when there is less fluctuation

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 re-learning process takes place for multiple edges simultaneously.

Hence, the tuning needs to be optimized. 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, the learning process doesn't need to run for all the epochs. Figure 36(a) shows the loss function's characteristics spanning 200 epochs which took 27.6 Seconds to complete learning. The function settles around 55'th epoch and has stayed constant since then. Figure 36(b) depicts the effect of the Early Stopping that brings the training time to just 0.53 seconds. Thus, it exponentially reduces the time consumption of re-training the RNN, making it feasible for online training.

With several trials of online training, a more comprehensive comparison between the actual and predicted reliability is shown in Figure 36(c, d). The first compares the best, worst and average cases, sampling them down to a set of 20 instances, collected over 20 minutes of online learning. The results show discrimination is prominent when there is less fluctuation in the data sets; it's more comprehensive when the deviation is plotted on a log scale (Figure 36 (d)).

D. Rapid Convergence and Co-relation to Sharpe Ratio

Figure 37 depicts the varying reliability of five edge nodes over a period of 350 stamps each of 10 seconds. The log scale is used to magnify the variation. Over time, three nodes 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 emulated these dynamics by randomly altering various node and edge attributes. This causes the network to be highly chaotic and the routing protocols to re-converge frequently. An effect that appears in Figure 24(A, 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. The relative dotted boxes also draw a clear correlation between the learned reliability and the Sharpe ratio. As the Sharpe ratio measures the degree of volatility

every time, it meets a rapid depression. The corresponding router is chosen as the most reliable. During the training, the RNN captures this trend and predicts accordingly. We set the window size of 100 timestamps; thus, an offset of 100 can be seen on the time axis of the two plots.

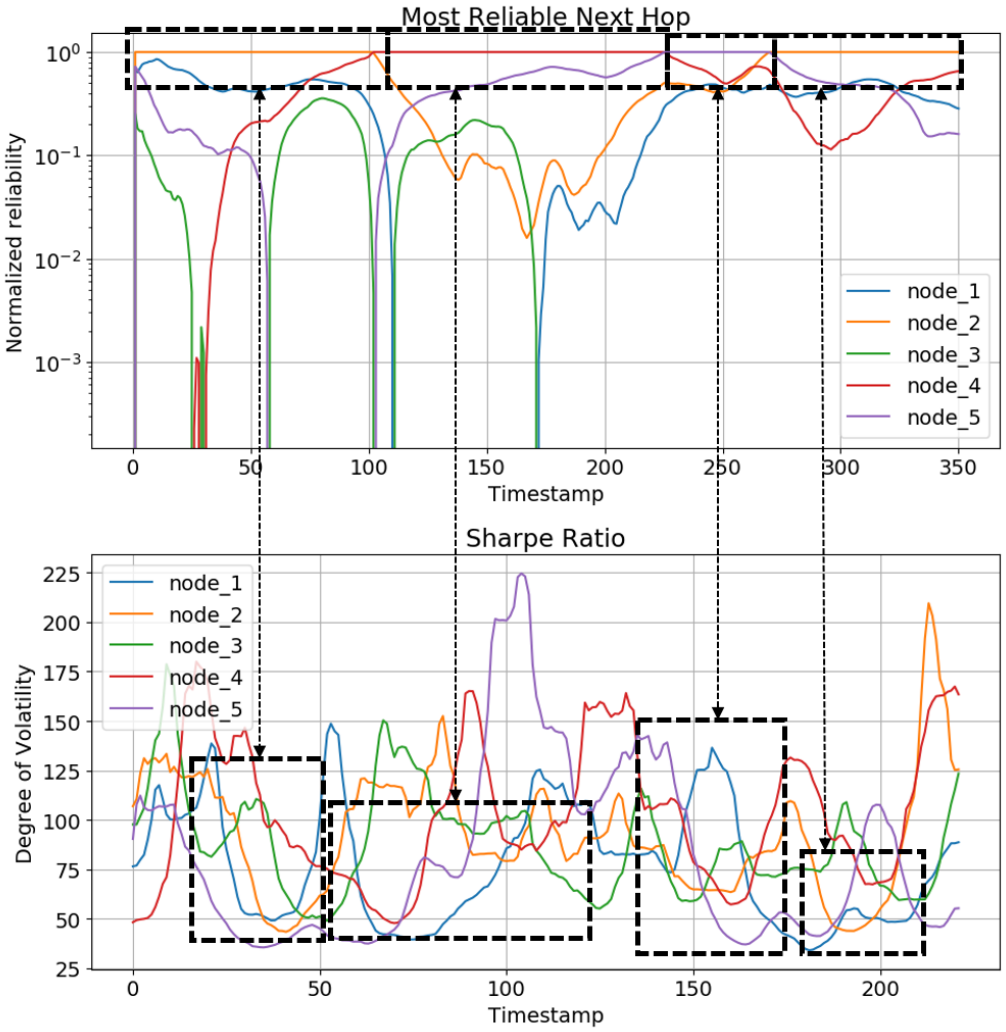

Figure 37 Demonstration of Self-Healing through rapid-convergence: At timestamp $[0-100]$ $Node_2$ *is most reliable as the corresponding rolling Sharpe-Ratio has maximum descending gradient calculated on 100 timestamps. Similar pattern can be noticed for* $Node_4$ *during [100-240],* $Node_5$ *during* $[240-270]$ *and* $Node_2$ *during* $[270-350]$. *The correlation is analytical however the RNN learns it.*

Chapter Summary

This chapter summarizes the contributions to the Self-Optimization problem of 5G and beyond. The optimization problem it addresses is the Routing as-a-service for a knowledge-defined network (KDN). It covers a three-part discussion, first, the STEN method takes both the link and node utilization into account for costing the communication cost, which makes the routing decision more robust as it can include convergence scenarios based on service overload. Second, the pathfinding algorithm *MRoute* which proactively finds all possible paths between all pairs of nodes and stores them efficiently to the controller for a reactive constant time convergence. Third, the MRRF technique uses RNN to estimate the reliability as uses it for metric calculation, which is then used by *MRoute* to update the cost matrix periodically.

The experiments benchmark the optimization process in each of its aforementioned stages against both distance vector and link-state routing algorithms. Test results validate the claims of constant time convergence in dynamic network situations by verifying several stress scenarios.

In summary, the chapter discusses the compliance of the Cognitive Routing model to the Self-Optimization feature in 5G and beyond architectures. It also explains the role of cognitive routing in pursuing the URLLC by choosing a more reliable path to transport than the cheapest one.

Chapter 4: Self Configuration

The self-configuration property leverages the Softwarization property of Software Defined Networking (SDN). Chapter 3 discusses the Self-optimization property, where it describes how the optimization problem calculated the optimal decision variables subjected to the given constraints. The self-configuration property is responsible to inject them into an underlying Control Plane (CP) using North-Bound Interfaces (NBI) which the CP exposes as Application Programming Interfaces (APIs).

This chapter introduces the elementary self-configuration concept from a software engineering perspective with a historical coverage of several SDN programming paradigms. Later, it presents a list of contributions that leverages the self-configuration framework in the scope of this thesis.

4.1. Introduction

Over the stretch of several decades, Internet architecture has evolved enormously. The core philosophy that gave birth to the Internet was to provide a generic fabric that connects several vendor-specific and platform-dependent networks, hence the term '*Inter-Network*'. Therefore, the design philosophy of the Internet is more skewed towards stability than dynamics, e.g., the working principle of Exterior Border Gateway Protocol (eBGP) deviates significantly from the rest of Interior Gateway Routing Protocols (IGPs) in that very context. Eventually, the Internet architecture has become static and hard to change as the core design prefers stability and accuracy over dynamics. This phenomenon is called *Internet Ossification* [202], which elongates the innovation lifecycle in network engineering compared to its software counterpart.

The SDN architecture results in a paradigm shift in dealing with the stringent requirements by segregating the network's functional constituents. It offers rapid programmability on the overlay network keeping the underlaying networking untouched. Arguably, this has been the key factor behind the decoupling of the Control and Data planes which outstand SDN over its likely predecessors such as Open Signaling (OpenSig) [203], Active Networks (AN) [204], and Ethane [205].

Although AN is not similar to what SDN offers, before exploring deep into SDN programmability, it is essential to address two questions, first, *why did OpenSig fail?* and *why did SDN win?*. The OpenSig model addressed the issues faced by AN by segregating the CP and DP, but its static programming interface results in a tight couple between the programming language and programmable hardware. The SDN model blends the best of both the aforementioned models. First, it removes the platform dependence using flow modification and loosens the coupling between the programming language and the programable hardware using standard APIs. Second, it applies CP-DP segregation by isolating the forwarding logic from the forwarding hardware (e.g., switch, router, firewall) and placing it in the central controller. This results in the forwarding hardware autonomously controlling its forwarding circuits (i.e., Application Specific Integrated Circuit or ASIC) using a local software; where the forwarding rules reside in a local data structure called Flow Table (FT) which the controller populates.

Currently, three major organizations namely: The Open Networking Foundation (ONF), Internet Engineering Task Force (IETF) and Open Networking Research Center (ONRC) are responsible for SDN standardization. However, it is legit to mention, that there have been several flow management protocols developed like OpenFlow [206], NetOpen [207], OpFlex [208], POF [209], ForCES [210]; but OpenFlow has become the *de facto* protocol for SDN due to its robustness and wider industry acceptance.

The remainder of this section will delve deep into the SDN programmability to cement the rationale behind modelling the contributions to knowledge made by this thesis.

4.1.1. SDN use cases

A study of the recent literature [211][212] suggests seven use case scenarios that leverage SDN programmability. Table 12 lists the suggested use cases and their objectives.

Use Case	Objective
Routing	Migrating from the traditional decentralized routing model to a logically centralized mechanism that accommodates Virtualization, Orchestration, Automation and Programmability on networks.
Cloudification	It is the most important use case in SDN [213]. Manages interconnection and interaction between Datacenter and transport networks (e.g., bandwidth allocation, policy enforcement, traffic engineering and network telemetry). Cloud platforms offer network orchestration as a part of their code modules, e.g., Neutron in OpenStack, Network-Node in Open Nebula, Network manager in Cloud Stack, etc.
Load Balancing	SDN provides load-balancing as a part of its control logic. There are claims that SDN could replace dedicated load balancers due to its native support [211].
Network Management	Centralized control eases the implementation policy-based network. Several automation frameworks such as Ansible, Puppet, Chef, Salt etc. enable the controller to automate the policy injection. Cisco's Intent-Based Networking [214] is a great example. The Cisco DNA centre and Campus Fabric for Cisco SD-Access (SD-LAN) and in SD-WAN with VManage (management plane), VBond (Orchestration), VSmart (Control Plane) and VEdge (Data-Plane).
Application-Centric Networking	SDN controller exposes Northbound-APIs for exchanging valuable information between the CP and AP and East/Westbound-APIs for inter-controller information exchange. External applications can leverage these APIs to inject application-specific policies such as Security policies, QoS/QoE policies etc. This reduces the deployment complexity significantly [215]
Security	The Authentication, Authorization, and accounting (AAA) architecture is the *de-facto* standard for enterprise networking. However, there has been several evidence of attacks in the AAA model, especially through Man in the Middle (MITM) and Distributed Denial of Service (DDoS) attacks.

Table 12 SDN use cases

4.1.2. The programming language taxonomy and SDN adaptability

Attribute	Classification	Description	Ref
Programming Paradigm	*Declarative programming*	• Formal language expresses the intent primarily using logic rather than arithmetic • The developer focuses on 'What' than 'How' • The interpreter translates the 'What' clauses into 'How' methods using Frenetic Notation [216]	[216] [217] [218]
	Functional Reactive Programming (FRP)	• Suitable for event-driven programming • Method invocations depend on event occurrence (Signals) hence reactive	[219] [220]
Language Specification	*Formal*	• Uses mathematical notation to express instruction • suitable for SDN programming	[221] [222] [223]
	Informal	Uses graphical modelling language such as UML to express the instructions	[224]
Network Programmability	*Domain-Specific Language (DSL)*	• Tailored to cater to a specific domain of application • Suitable for SDN programming • Abstracts the development complexity of common network procedures (e.g., Sockets, Tunnel, Flow-Injections etc.) • There are two classes of DSL, Textual and Visual aka *Domain-Specific Modelling Language (DSML)* • DSML uses the concept of *Model-Driven Engineering* (MDE) to speed up the deployment process by hiding the implementation details and reducing the complexity of programming mundane tasks. • The MDE framework achieves the above by segregating its specifications into MD-Architecture (MDA) which uses UML to define the overall deployment architecture and MD-Development which leverages the model defined in MDA and Interoperability to translate into target-specific instructions. The Object Management Group (OMG) defines the translation standards for MDE using a concept called *Meta-modelling*.	[225] [226] [227] [228] [229] [230] [231] [228]
	General Purpose (GPL)	• Domain agnostic (e.g., C, C++, Java Python etc.) • Suitable for developing generic Network applications (Traditional and SDN)	

Table 13 Language Taxonomy of SDN programming languages

Table 13 summarizes the several industry-standard programming languages based on their class as prescribed in the standards [222][216]. These apply to implementing the controller logic and higher-level ones can express policies better. From the above table, we can conclude that a model SDN programming language should be Declarative or Functional reactive, Formal and Domain-Specific. *Table 14* compares the benefits and demerits of each programming language paradigm defined above.

Paradigm	**Pros**	**Cons**
FRP	• Efficient event-driven programming. • Enable modelling of delays and state, • Implicit caching and multicast	• Performance • Complexity in creating data structures, memory, and space leaks
DSL	• High abstraction level • Fewer lines of code • Flexible • Enables verification and validation of application • Higher productivity in the specific problem domain • Layering that can lead to language-independent from the underlying infrastructure	• Performance • Language design is hard • Useless for application outside the domain
Imperative	• Flexibility • A high degree of abstraction • The developer defines ‘how’ he wants a network to behave	• Complexity in creating structures • Low abstraction level
Declarative	• High abstraction level • Fewer lines of code • Simplicity, in focusing on ‘what’ a developer wants a network to behave	• Inflexibility • Hard to express conditions
Logic programming	• Flexibility and reliability • The network architecture of protocol (e.g., OpenFlow) can be changed without changing the program or their underlying code	• Performance • Lack of arithmetic, event, and datatype support • Complexity in creating structures

Table 14 Comparison of different programming paradigms

4.1.3 State of the Art in SDN programming languages

The initial SDN language developments could be traced back to 2009 when the standard was incepted. Lopes *et al.* in their survey [232] present a comprehensive comparison of the state-of-the-art initial SDN programming languages given in Table 15.

Language	Paradigm	Objective	Year	Limitation
FML [233]	Declarative + DSL	High-level abstraction of network behaviour to replace specific configuration	2009	No arithmetic operation, No dynamic policy, No rule conflict resolution, No explicit negation rule
Nettle [234]	Declarative + FRP + DSL	Declarative programming of OpenFlow	2011	No rule conflict resolution
Procera [227]	Declarative + FRP + DSL	Express reactive dynamic policies in a declarative way	2012	No direct support for events or external queries
Flog [235]	Declarative + DSL	Event driven and forward-chaining language	2012	No explicit negation rule
NetCore [236]	Declarative + FRP + DSL	Intent-based network programming, the programmer defines the ‘what’ clauses and the interpreter translates them into ‘How’.	2012	No support for stateful control
Frenetic [216]	Declarative + DSL	High level of abstraction for programming state and forwarding policy to the controller	2013	Consistency only to single switch per flow.
FatTire [217]	Declarative + FRP + DSL	Write a program in terms of paths through the network and explicit fault tolerance requirements	2013	No inbuild Fault detection and recovery, and no QoS support
Pyretic [237]	Declarative + DSL	Specify network policies with a high level of abstraction	2013	Consistency only to single switch per flow.

Language	Paradigm	Objective	Year	Limitation
Nlog [238]	Declarative + DSL	Compute the network forwarding state separating the logic specification at the controller	2013	No flow verification method, no explicit negation rule
Flowlog [239]	Declarative + DSL	Abstract DP and CP behaviour	2014	No abstraction for queries
Merlin [240]	Declarative	Express high-level policies that provision network resources	2014	Does not provide consistency between policies
Kinetic [241]	DSL	Provide abstraction for automating changes in network policies	2015	Does not provide consistent updates by itself

Table 15 Comparison of SDN programming languages [232]

The above summary shows the various SDN programming languages developed during the period of 2009 – 15 with their corresponding merits and limitations. The study shows that there isn't any language for "one size fits all", due to some inherited limitations such as static policies, no configuration conflict resolution, no explicit rule generation, lack of event-driven programming support, lack of external query support, lack of simultaneous configuration of multiple switches with consistent configuration, lack of fault tolerance and high availability, and lack of rule verification. Therefore, after 2015 when SDN standardization was stated by ONF, the number of languages started to converge into more API driven implementation. The SDN controller exposes a standard RESTful API such as RESTCONF to allow applications to leverage it. This addresses all the limitations mentioned above to be resolved by the controller itself before it pushes the translate forwarding entry to the data plane. Recently ONF has launched a new programming language called P4[242] that provides the ease of declarative programming and Stratum[243] a switch operating system that unifies several platform-specific deployments. The following sections list the contributions to the self-configuration domain primarily using the API manipulation using a GPL (Python3.x).

4.2. System-Level Simulator integration with SDN (SDN-SIM)

Design and structural complexity are skyrocketing with the introduction of diverse technology paradigms in next-generation cellular and vehicular networks. The beyond- 5G use cases such as time-critical application, 5G-V2X, and UAV communications require ultra-low latency and high throughput and reliability with limited operational complexity and cost. These use cases are being explored in 3GPP Releases 16 and 17. To facilitate end-to-end performance evaluation for these applications, we propose SDN-Sim-integration of a System Level Simulator (SLS) with a Software Defined Network (SDN) infrastructure. While the SLS models the communication channel and evaluates system performance on the physical and data link layers, the SDN performs network and application tasks such as routing, load balancing, etc. The proposed architecture replicates the SLS-defined topology into an SDN emulator for offloading control operations. It uses link and node information calculated by the SLS to compute routes in SDN and feeds the results back to the SLS. Along with the architecture, data modelling and processing, replication, and route calculation frameworks are proposed.

4.2.1. Preliminaries

Towards 5G/B5G, the third-generation partnership project (3GPP) finalizes release and defines Release 171. In the area of vehicular networks, the 3GPP, in partnership with the Fifth Generation Automotive Association (5GAA), is driving the efforts on the 5G-based vehicle-to-everything (V2X) paradigm, which adds advanced features to the LTE-V2X from Release 14, particularly in the areas of support for ultra-reliable and low-latency communication (URLLC) applications for the future intelligent transport systems (ITS) [29],[30], [31]. In the evolution path from LTE-V2X to 5G-V2X, the authors in [29] advocated the incorporation of SDN in the architecture to enhance the system performance through SDN’s capabilities in facilitating

intelligent multi-hop routing, dynamic resource allocation, and advanced mobility support, among others.

To evaluate the performance of proposed algorithms, techniques, and frameworks for any new era of communication networks, numerical simulations, mathematical analyses, and field trials are the three main approaches being employed. Though analytically tractable, mathematical methods (e.g., stochastic geometry tools) are often constrained by simplifying assumptions that potentially limit their use in modelling large-scale, highly complex, and dynamic networks. Realistic performance can be measured in live operating environments. However, the financial and operational requirements are costly and practically infeasible for the early design and development stages. Hence, in the past few decades, simulations have become essential tools for the assessment of network performance due to the apparent cost and implementation advantages [32].

Depending on the performance metrics under investigation, simulators can be categorized into three: Link Level Simulator (LLS), System Level Simulator (SLS), and Network Level Simulator (NLS). The LLS examines detailed, bit-level physical (PHY) layer functionalities of a single link. The SLS evaluates the performance of links involving many Base Stations (BSs) and User Equipment (UEs) at the Medium Access Control (MAC) layer (with the PHY abstracted). It focuses on the radio access network/air interface and facilitates analyses of resource allocation, capacity, coverage, spectral and energy efficiencies, amongst others. The NLS, however, assesses the performance of protocols across all layers of the network, including control signalling and backhaul/fronthaul issues. Performance is characterized using metrics such as latency, packet loss, etc. [33].

Besides metric-based classification, simulators can also be grouped based on radio access technologies supported (cellular, vehicular, Wi-Fi, etc.), programming language environment (MATLAB, Python, C++, etc.), licensing option (open source, proprietary, free of

charge for academic use) or network scenario capabilities (LTE, 5G, B5G, etc.) [33][18]. While the SLS does not simulate beyond the MAC layer, the NLS simulates networks up to the application layer. However, the implementation and computational complexity of NLS become very high when many nodes are involved [32].

Another significant paradigm shift in network design takes place with the advent of SDN [34]. It decouples the control (signalling) plane from the data (forwarding) plane and runs applications in the AP to manage the network. This brings transparency to network design and lets software developers write applications for managing the networks, keeping the internal design in abstraction. Each layer uses several interfaces to communicate with each other. The CP communicates with both AP and DP using North and Southbound interfaces, respectively. In the case of a cluster of controllers, East and Westbound interfaces are used for communicating among them.

The default southbound protocol for SDN, OpenFlow uses FT to perform packet forwarding. Each FT entry is a forwarding rule determined by the controller. A forwarding rule has mainly three significant fields, a “match,” an “action,” and a “priority.” A “match” is some criteria for an inbound packet to be checked. A packet that satisfies the criteria is termed a “table hit”; otherwise, it is a “table miss.” For each case, an action is defined such that the OpenFlow switch executes on the subjected packet. If a packet satisfies matches from multiple flow rules, priority is used to break the tie. The SDN Controllers populate flow entries. The OpenFlow switch requests the controller for every table miss and the controller replies with a flow entry. If the controller cannot resolve an action, it is set as a “drop,” The switch does not process the packet. The decoupled control plane reduces computational cost on forwarding

[18] Representative simulators include the Vienna LTE-A and 5G simulators for LLS and SLS (https://www.nt.tuwien.ac.at/research/mobile-communications/vccs/), and the 5G-K Simulators for the NLS (http://5gopenplatform.org/main/index.php).

devices by offloading the control packet processing tasks to the controller. Therefore, SDN offers better modularity, programmability, agility, automation, and load balancing capability than traditional networks. Also, the SDN-based approach is used in network design practices for cloud computing and 5G.

This section presents a novel SDN-based System Level Simulator (SDN-Sim) platform where the SLS-Stage runs in MATLAB, and the NLS stage is based on python3. By inheriting all the benefits of SDN, the architecture considerably reduces the overall computational complexity of the system. The computationally demanding upper layer network functions (e.g., inter-cellular routing) are offloaded to the virtualized cloud infrastructure. The controller maps Low-level network information (e.g., Channel model, topology, etc.) from SLS to SDN. Python-based application development and virtual infrastructure using Type-1 hypervisor (VMWare ESXi) servers. OpenFlow and RESTCONF are the south and northbound protocols, respectively. OpenDaylight is used as the SDN controller, while GNS3 and Mininet-wifi emulate the data plane emulation.

4.2.2. System Architecture and Implementation

Figure 38 depicts the system architecture of SDN-Sim, with the SLS at the bottom and the SDN infrastructure running on the top. When the SLS does the channel modelling and scheduling, the SDN takes care of the upper layer functionalists such as IP routing and traffic control, described as follows.

A. System Level Simulator

The tasks of the SLS run in loops of transmission time intervals (TTIs), and the results are averaged over several simulation runs or channel realizations [244].

- **Scenario Setting:** The layout depicts a Vehicular Network of BSs/Roadside Units (RSU) and vehicle-mounted radio/UEs configured with 3D locations of the nodes. UEs or

vehicles are mobile and attached to their serving RSU, which further interfaces with the cellular networks through BSs with parameters such as line-of-sight probability, distance, and Signal-to-Noise Ratio (SNR).

- **Channel Modeling:** For all links, the path loss (PL), shadow fading (SF), transmit and receive, and antenna gains and fast fading are calculated to estimate the channel of each user for both desired and interfering links.
- **Scheduling**: Radio resources are allocated to users based on the scheduling algorithm. Resource blocks (bandwidth) and power are allocated either in a quasi-random fashion (for open Loop configuration) or based on feedback from the users in closed-loop systems. The channel state information (CSI) feedback and other factors such as the traffic type of users, link adaptation strategy employed, and quality of service (QoS) demands are used as decision determinants at the scheduling stage.
- **Link Quality and Performance Estimation:** The links' signal-to-noise and interference ratio (SINR) are then estimated. The users ‘throughput and the cells’ capacities are calculated using the SINRs and the link abstraction model (for the block error rate (BLER)).

Recent developments in SLSs are mainly focused on solving problems on channel modelling [245], high-frequency communication [246], [247], coexistence and performance optimization [248], energy efficiency, latency, scheduling, and load balancing over a heterogeneous network [249], among others. The 5G public-private partnership project (5GPPP) has described several aspects of Softwarization, service management, and orchestration in their architectural reference [250], including SDN, cloud computing, virtualization, etc. Therefore, cloud computing is being employed to enhance the scalability and computational efficiency of 5G SLSs by offloading the computational load [251].

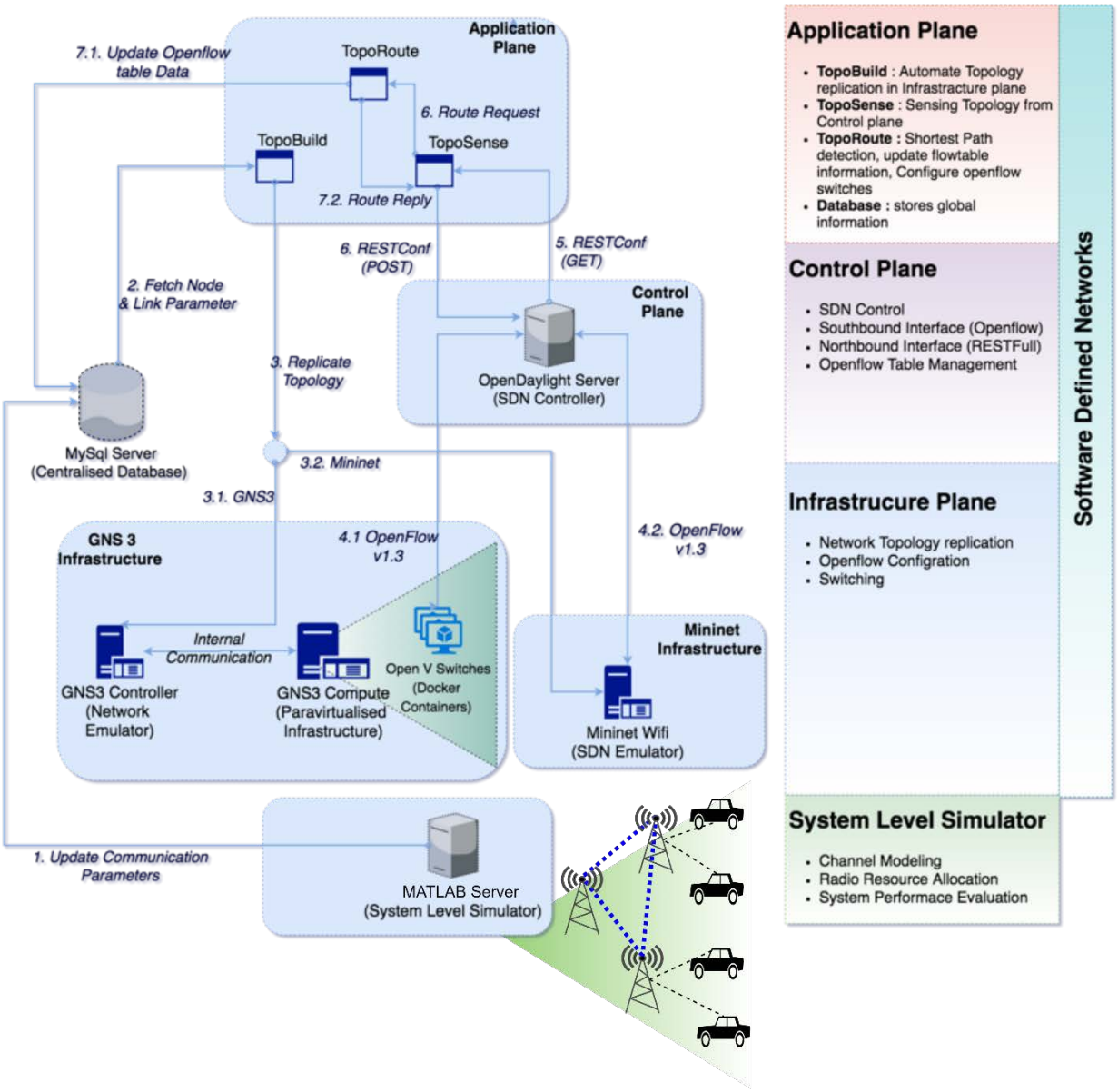

Figure 38 Schematic system architecture of SDN-Sim with Full stack setup along with their core functions

B. Software-Defined Network

The task of the SDN extension is described as follows.

- **Channel State Monitoring:** The SLS running in a MATLAB server with Open Database Connector (ODBC) driver updates the topology and channel modelling parameters to a centralized Database.

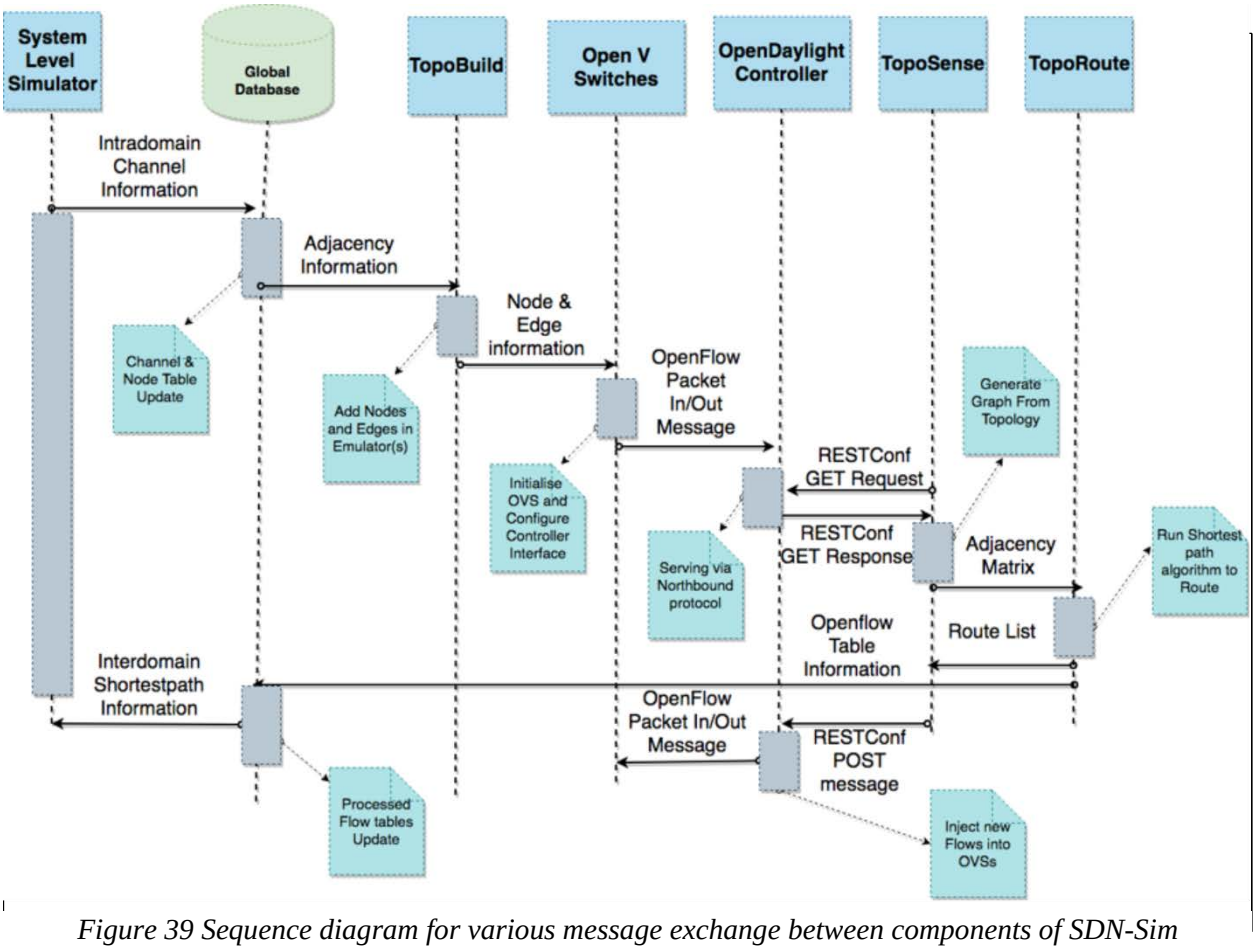

Figure 39 Sequence diagram for various message exchange between components of SDN-Sim

- **SDN Emulation:** The topology information is reactively fetched and emulates using the *TopoBuild* module in the SDN DP through GNS3 (wired core networks) and Mininet-wifi (wireless edge network). The CP starts communicating with Open-Flow.
- **Route Calculation:** The application plane fetches topology from the control plane, translates it into a graph using the *TopoSense* module and finds the shortest paths between node pairs using the *TopoRoute* module. These paths are fed back to the database and the controllers.

The workflow and communication sequence taking place between the various elements of the architecture are as given in Figure 39

1. **Update communication parameters:** The SLS performs channel modelling, radio resource allocation, and system performance evaluation. After optimizing the model for a given scenario generates several channel parameters (bandwidth, distance, path-loss,

latency, and delay) and BS parameters (position, range, RSSI, transmission power, etc.). Since parameters are calculated per BS, each BS updates its local dataset to the centralized database.

2. **Fetch node and link parameters:** *TopoBuild* is a bespoke program to fetch SLS parameters from the central database and relay them to the SDN Data plane to replicate the topology. BSs are placed as wireless access points, running OpenFlow protocol. Links, depending on types (i.e., fronthaul or backhaul), are assigned to BSs. Wireless fronthaul connection carries several radio parameters such as path loss, frequency band, RSSI, etc.

3. **Topology replication:** *TopoBuild* translates parameters obtained from the central DB into a series of commands. There are two possible infrastructures (GNS3 and Mininet-Wifi) in the proposed architecture with three possible deployment options (full GNS3, full Mininet-wifi, and hybrid). *TopoBuild* generates a script to replicate the topology and injects it into the specific engines (GNS3 and/or Mininet-Wifi) based on the deployment type and connection specifications. A feature comparison between GNS3 and Mininet-Wifi is given in Table 16 and briefly described as follows.

	GNS3	Mininet-Wifi
Similarities	• Both are open source and offer GUI-based network design. • Both support API based Interfacing with the external environment • Both have methods to access the physical interface • Both can host OVSs and dummy workstations for data plane	
Pros	• It is not Limited to SDNs only. • Use of actual OVS images. • Better Test-result accuracy • Realistic test cases. • Supports scalability via Clustering. • Native Docker Support. • Extensibility with apps from the GNS3 market- place	• Rapid Implementation, no SDN / OpenFlow configuration needed link configuration • wireless support • native SDN support • Inbuild mobility & propagation models
Cons	• No wireless support • Manual configuration for SDN / OpenFlow and other supporting devices No link configuration • Requires more physical resources to set up the test environment	• Limited to SDNs • Test-result accuracy • Scalability via multi-threading • No clustering supports • No paravirtualization • No app-based functional extensibility

Table 16 A comparative study of GNS3 and Mininet Wi-Fi as Data Plane Engine

a. **GNS3 Infrastructure:** GNS3 is an open-source network emulation software. It comprises the controller VNF that interfaces through RESTful APIs, manages topology, etc., and a cluster of para-virtualized compute nodes that host VNFs such as routers, switches, firewalls as containers, or VMs. In SDN-Sim, *TopoBuild* communicates with GNS3-Controller using RESTful API, and the Open-V-Switches (OVS) run as a docker container within the GNS3-Compute cluster. Therefore, GNS3 offers realistic test cases with a testbed and produces more authentic results than a simulation. On a downside, it lacks support for wireless networking emulation; however, it allows access to a physical wireless card.

b. **Mininet-Wifi Infrastructure:** With a Wi-Fi extension, inheriting all the features of Mininet, it can now emulate wireless SDNs. With the support of its python API, programming and configuring Mininet-Wifi is more user-friendly. Mininet-Wifi

runs in a single sandbox with its interactive command line. Thus, *TopoBuild* uses raw sockets to inject the Mininet-wifi commands for deployment and parameters updates. It also offers four mobility and five propagation models. Since Mininet-Wifi uses HWSIM drivers to simulate wireless networking, emulating an extensive network is constrained. Also, it does not support clustering,

4. **Accessing Control Plane with OpenFlow:** Both GNS3 and Mininet-Wifi host OVSs; thus, they communicate with the controller using OpenFlow v1.3 (OF1.3) as southbound protocol. OF1.3 uses bidirectional messaging to communicate with the switches. A switch requests a controller with a Packet-In message, and the controller replies with a Packet-Out. SDN-Sim uses Open-Daylight-Beryllium SR4 (ODL) as a controller that runs as VM. Both GNS3 and Mininet-Wifi use TCP ports 6633 and 6635 for communication. The CP supports the binding of several controller nodes with clustering to maintain scalability, high availability, and persistence in the data plane. ODL supports 'Akka' clustering for this purpose.

5. **Interfacing with Application Plane with RESTCONF:** SDN-Controller interacts with the application plane using Northbound APIs. ODL uses RESTCONF (RFC-8040). It is a RESTful version of NETCONF (RFC- 6241) protocol and uses JSON (RFC-7159) format to transfer data among REST-enabled devices. In SDN-Sim, we use two RESTCONF resources (Inventory and Topology). "Operational" resources are to read, and "Config" are to write. The inventory resource provides node-wise OpenFlow tables, and the Topology resource provides the topology of the network. The App *TopoSense* makes use of the resources to model a graph by fusing topology and flow-table information. *TopoSense* invokes *TopoRoute* to calculate the route for a given topology.

6. **Route Calculation:** *TopoRoute* uses Stochastic Temporal Edge Normalization (STEN) [1] technique (Discussed in Chapter 3) to find routes. It receives the topology and flows tables from *TopoSense*. The link parameters are already present in the database. By fusing them, a single source shortest path algorithm is run over every pair of vertices. A set of all possible routes are generated. In a traditional network, local routes are shared among neighbours to form routing tables. With the size of the network, due to propagation delay, the routing becomes significantly slow. However, in an SDN paradigm, discovery is made by the controller; hence the topology graph is mapped proactively. Therefore, the shortest paths between every pair can be calculated in parallel. This results in speeding up the routing process and allows scalability in the network.

7. **OpenFlow Tables Update:** *TopoRoute* does event-driven updates of the flow tables in the Central DB. The calculated routes are fed back to the *TopoSense,* which eventually replies to the routes back to the controller using RESTCONF Inventory-Config API.

After the central DB gets updated, the SLS picks the information, and an inter-cellular route is discovered to leverage the lower (physical and data link) layer operations. The complete operational sequence diagram is given in Figure 39.

C. Data Modelling & Design of Central Database

The central database plays the role of middleware between the SLS and the SDN. Table 17 shows the related entities and their corresponding attributes for the data model. There are three main entities: Node, Flow Table, and Channel. A node can be of either Access Point (AP) or Host. One host is associated with one AP, and one AP can associate with many hosts. Each AP is an OVS; thus, it contains a flow table(s) and a unique Data Path

Node Attributes	
Node ID	Primary key, Unique ID for each node
Type	Type of the node (AP or Host)
Range	Communication Range of the node
Position	Location of node (3D cartesian coordinate)
Channel	Operating Channel Number
Frequency	Operating Frequency (In Hz)
Mode	Operating Mode (B/G/N/AC/AX)
Tx power	Transmit Power in mW
IP Address	Nodes IP Address
MAC Address	Nodes MAC Address
Access Point Attributes	
Station Association	List of Host the AP is associated
Host Attributes	
AP Association	The ID of the AP to which the host is associated
RSSI	Relative Received Signal Strength (at host end)
Flow Table Attribute	
DPID	Unique Data path ID of the AP
Source IP	Match field for source IP in ingress packet
Destination IP	Match field for Destination IP in ingress packet
Source MAC	Match field for source MAC in ingress Frame
Destination MAC	Match field for source MAC in ingress Frame
Action	OpenFlow Action opcode
Timeout	Timeout (Dead) timer
Packet Count	Total packet count statistics
Byte Count	Total bytes count statistics
Duration	Hold time for OpenFlow Entry
Node Channel Map Attributes	
Channel ID	Foreign key to Unique Channel ID in Table
Node_1	Foreign key to the first Node ID
Node_2	Foreign key to the second Node ID
Channel Attributes	
Channel ID	Primary key, Channel Identifier
Bandwidth	Channel Bandwidth in Mbps
Distance	Distance between incident nodes in meters
Pathloss	Pathloss of the channel in dB
Latency	Average latency (in ms)
Delay	Average RTT (in ms)

Table 17 Descriptions of attributes of the data model

Identifier (DPID). A pair of nodes makes a channel with a unique channel ID (mapping is recorded at the Node Channel Map table). A channel between a Host and an AP is called fronthaul (Wireless), and between a pair of APs is a backhaul (Wired). No channel exists between two hosts. Since *TopoSense* reads information from the APs, the route calculation by *TopoRoute* takes place over the backhaul network.

D. Design of Application Plane

Three major apps *TopoBuild* (Algorithm 4), *TopoSense* (Algorithm 5), and *TopoRoute* (Algorithm 6) run in the Application Plane. In the previous sections, their usage has been mentioned, the working principles are given below.

Algorithm 4: ***TopoBuild***

Name : *TopoBuild*

Purpose : Initiate topology replication from the SLS to the SDN platform

Input : Topology Information

Output : Configuration command for the Emulation

Data structure: Di-Graph

Steps:

Do

$N = \{n_i\}$ *// read Node information*

$E = \{(i,j) | \forall\, n_i, n_j \in N\,, Adj(n_i, n_j)\}$ *// read Edge information*

$C = \{c_{i,j}\}$ *// read Channel Information*

If Not (TOPO_INIT) **then**

CMD_NODE = *{ add_node(n_i)* $\forall n_i \in N\}$ *//add node config*

CMD_EDGE = *{ add_edge(I, j)* $\forall (I,j) \in E$ *} // add edge config*

Sim = INIT_DP() *// initialize Data Plane simulation*

[end if]

// Configuring DP

For $n_i \in N$ **Do**

Perbegin(Sim.Exec(CMD_NODE)) *// concurrent injection of node config*

[end loop]

For $(I, j) \in E$ **Do**

Perbegin(Sim.Exec(CMD_EDDE)) *// concurrent injection of edge config*

[end loop]

While(TOPO_CHANGE) *//recompute, if a topology change is detected; else spinlock*

Algorithm 5 : ***Topo Sense***

Name : *TopoBuild*

Purpose : Interact with SDN Controller to update topology information

Input : Controller Information (RESTCONF client object)

Output : Flow instruction for Shortest Path injection

Data structure: Di-Graph

Steps:

```
Do
    Try     // read the OpenFlow Table and Topology using RESTCONF
        OFT  = RestConf.Operational.request( GET, Inventory )
        TOPO = RestConf.Operational.request( GET, Topology )
    [ throws COMM_EXCP ]

    Try     // get the graph G  built using TopoBuild algorithm
        G = Call( TopoBuild( TOPO ) )
    [ throws CONFIG_EXCP ]

    Try     // Create the Shortest Path vector created using TopoRoute algorithm
        SP = Call ( TopoRoute ( G ) )
    [ throws GRAPH_EXCP ]

    For (I, j) ∈ G€ do  //Serialize SP configurations into JSON for transport
        CONF(I,j) = JSON.Encode( SP(I,j) )  // for all edge
    [end loop]

    Try     // Concurrent configuration of nodes, update OFT and TOPO
        Perbegin( RestConf.Config.request(POST, CONF(I,j), Inventory) )
        OFT  = RestConf.Operational.request( GET, Inventory )
        TOPO = RestConf.Operational.request( GET, Topology )
    [ throws COMM_EXCP ]

    Catch ( COMM_EXCP )
        Reconnect( TIME_OUT )     // retry until TIME_OUT timer expires
    [ end catch]
    Catch ( GRAPH_EXCP )
        Exit( Code = 100 )        // exit with code 100
    [ end catch ]
    Catch ( CONFIG_EXCP )
        Exit ( Code = 200 )       // exit with code 200
    [ end catch ]
While( TOPO_CHANGE )              // repeat if topology changes; otherwise spinlock
```

*Algorithm 6: **TopoRoute***

Name : *TopoBuild*
Purpose : Interact with SDN Controller to update topology information
Input : Controller Information (RESTCONF client object)
Output : Flow instruction for Shortest Path injection
Data structure: Digraph
Steps

Do
- **If** (INIT == False) **Then** *// If forest is not initialized*
  - $t = 0$, LEARN = False *// initialize timestamp and Flags*
  - $R = \{R^{(t)}\}, C = \{C^{(t)}\}$ *// initialize Cost and Reliability Vector*
  - $Adj = G$.Adj() *// Read adjacency matrix of G*
  - F = **Call** (MRout(Adj)) *// Create Route Forest F using MRoute*
  - IINIT = True *// set the init flag*
- **Else**
  - RET $= \phi$ *// initialize return Vector*
  - **While** $|C| \leq W$ **Do** *// accumulate costs until window expires*
    - $Adj_n^{(t)}$ = **Call**(STEN(Adj)) *// Normalize using STEN*
    - $C^{(t)} = C^{(t-1)} \cup \left\{c_{(i,j)}^{(t)} \mid \forall (i,j) \in E\left(Adj_n^{(t)}\right)\right\}$ *// update Cost Vector*
    - $t = t + 1$ *// update time index*
    - **If** $|C| == W$ **then** *// if Overflow anticipated*
      - **Delete**(C_0) *// Slide window by 1*
      - LEARN = True *// Init Learning*
    - *[end if]*
    - **If** LEARN == True **Then**
      - **For** $(i,j) \in E(G)$ **Do**
        - *// concurrently calculate the reliability for all edges in G*
        - **Perbegin**$\left(R^{(t)} = R^{(t-1)} \cup \left\{ Rel\left(c_{(i,j)}^{([t-W:t])}\right) \mid \forall c_{(i,j)}^{(t)} \in C\right\}\right)$
      - *[end loop]*
      - RNN.train(R) *//train the neural net with reliability vector*
      - $\delta_r = r^{(t)}$ − RNN.predict$(t+1)$ *// δ_r is the deviation of the forecast*
      - **If** δ_r > CUTOFF **then** *// deviation exceeds the cutoff value*
        - NEW_RET = $\left[r_{i,j}^{(t)}\right]_{N(G)^2}$ *//Reliability for all edges*
        - **For** $(i,j) \in E(G)$ **do** *// for all edges exceeding the cutoff δ_c*
          - **If** $\left| NEW\_RET_{(i,j)} - RET_{(i,j)}\right| > \delta_c$ **Then**
            - $C_{i,j} = NEW\_RET_{(i,j)}$ *//update cost*
            - $RET = NEW\_RET$ *// update RET vector*
          - *[end if]*
        - *[end loop]*
      - *[end if]*
    - *[end if]*
  - *[end loop]*
- *[end if]*

While(TOPO_CHANGE)

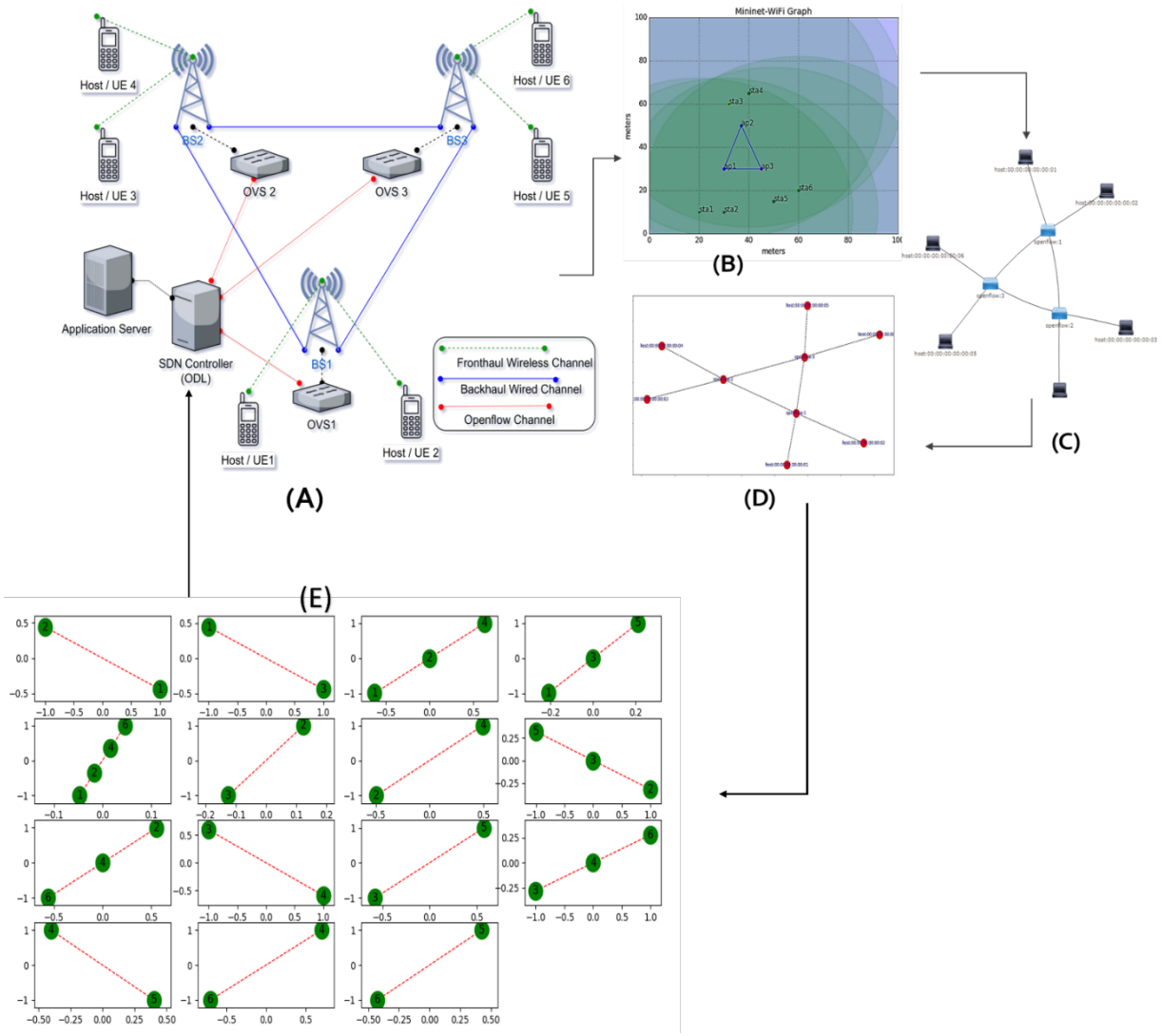

Figure 40 Closed-loop model of self-configuration

Figure 40 depicts the overall workflow resulting in a closed-loop system. In the First step, the *TopoBuild* algorithm reads the topology information and prepares the DP. Subplot B shows the emulated wireless topology using Mininet-Wifi from the Physical topology at subplot A. The OpenFlow protocol communicates with the OVS agent within the OpenDaylight controller to fetch the topology information (Subplot C). Notice, that the controller accesses the APs as a regular switch and the UEs as hosts. The *TopoSense* algorithm senses any change in the network, and *TopoRoute* calculates and injects route configuration to the infrastructure plane.

4.2.3. Experiments and Results

Figure 41 depicts the time both complexities of various phases and the latency for three use-cases (1, 7, and 19 sites and three sectors/site with each sector consisting of 1, 25, 50, 75, and 100 users) depicted in Figure 41 (A), (B) and (C), respectively. The Backhaul topology is configured wired and evaluated for Linear (minimal) and Mesh (Maximal) connectivity. The time to build the network on the SDN platform is termed as Build Time and Response Time the SDN takes to initiate traffic flow. Figure 41 (D) compares the breakdown of time consumption, number of links, and routing time; multi-threaded implementation of the shortest path algorithm limits the reactive route selection into the sub-second interval.

A. Experimental Setup

In the experiment, the following compute-node setup is as follows. The SLS runs in a MATLAB server VM with the Database tool running, and the ODBC adapter connects to MySQL Database. Mininet-wifi VM hosts BSs as APs and OVSs and UEs as stations. ODL runs in an Ubuntu 64 bit 14.04 VM and the Application server VM hosts the MySQL Database server along with *TopoSense*, *TopoBuild*, and *TopoRoute* apps. VMWare ESXi 6.5 server is used for the virtualization.

B. Latency and Time Complexity

Figure 41 depicts the latency of several stages of integration; SLS shares most of it. The setup phase consumes a significant amount of time depending on the network size and topology; this includes channel allocation and scheduling, SDN setup, Flow table population, Proactive route calculation, etc. However, the run-time is reactive and responds on a millisecond scale (Figure 41 (E)) since all the routes are pre-calculated and the network runs on a centralized virtual platform, which eliminates control packet exchange between devices to learn network topology. The time complexity of the proactive phase is of a high degree polynomial class, and the

reactive phase is constant; thus, once the SDN is deployed, response time comes down to the millisecond scale.

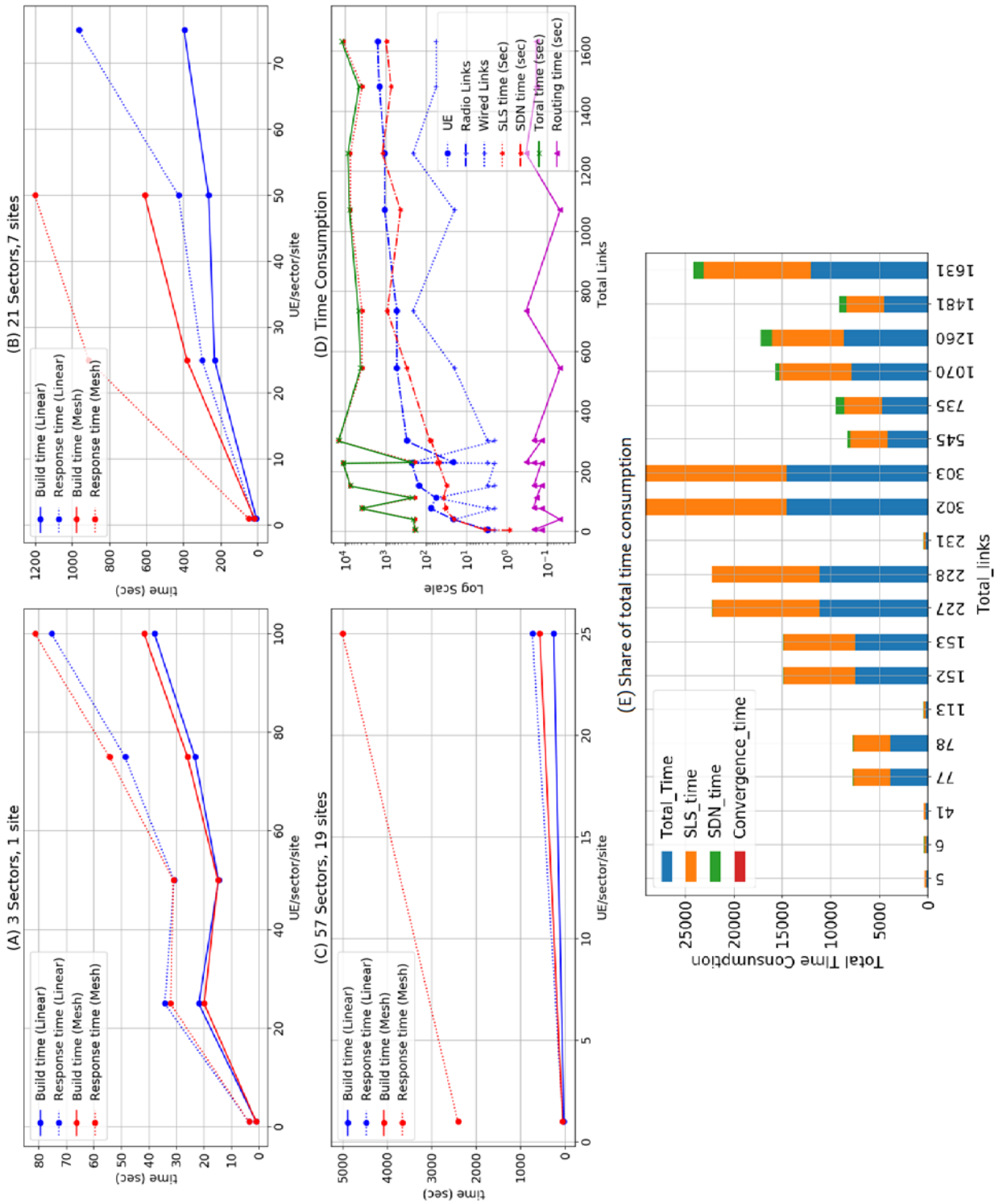

Figure 41 Sub-plots (A), (B), (C) depicts the build and response time for minimal (linear) and complete (mesh) topology of 1, 7 and 19 sites, respectively with 3 sectors per sites. (D) depicts total time consumption is predominated by the SLS channel scheduling, SDN tasks are comparatively lightweight and Routing time bounded by sub-second interval. (E) shows the total time consumption has a constant convergence time (it is too small to be visible on the staked bar chart)

4.3.*ShellMon*: Intelligent Telemetry System Architecture

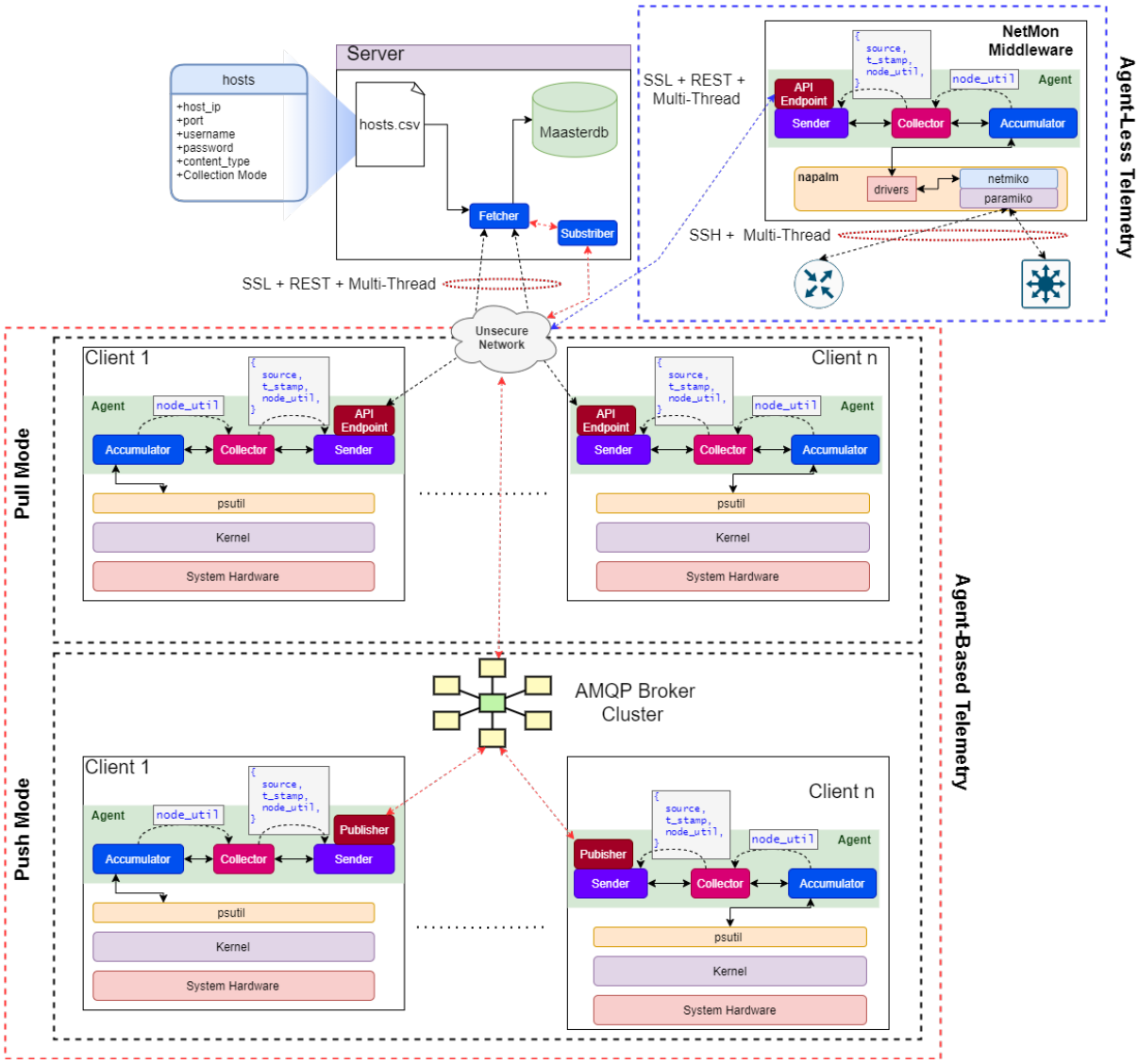

Figure 42 ShellMon API architecture

4.3.1. Architecture

The *ShellMon* API (Figure 42) provides an intelligent telemetry service to the CoRoS framework. In opposed to the classical telemetry protocol such as SNMP [252], and NetFlow[253], which are to some extent vendor dependent (e.g., SNMP MIBs and NetFlow versions), *ShellMon* provides a blend of two. Additionally, it supports RESTful RPC[254] style as well as Message Queuing communication for more robust implementation. Following is the

list of features from existing telemetry protocols that have inspired the architecting of the *ShellMon* API.

1. **Custom data modelling:** A Data-model is a serializable data structure that a telemetry agent populates when polled from the Collector. In Classic SNMP, the Management Information Base (MIB) is a vendor-specific hierarchical data structure. Similarly, the classic NetFlow also provides a fixed attribute set. Flexible-NetFlow[19] provides a custom data model where a user could choose from a list of statistics for collection. It gives the flexibility of relevant stats collection as per the requirement, also being efficient to the bandwidth utilization in a scalable and constrained infrastructure. Current protocols such as NETCONF[255] and its RESTful extension RESTCONF[256] provide model-driven programmability using the YANG[257] data modelling language. Although YANG is a widely accepted and used modelling language, certain caveats might be critical for YANG for establishing telemetry in a multi-vendor scalable environment. First, the YANG uses an XML[258] based model descriptor, which does not support native data structures like Dictionary and List. Therefore, additional translation is needed, which adds further complexity to the process. Secondly, the data models are vendor-specific. However, there are some open-source options (e.g., OpenConfig[20]). Still, they don't contribute much to the collection flexibility as a developer would require writing a YANG model from scratch to achieve so. *ShellMon* uses a JSON-based data descriptor that is far more human-readable than XML; the user does not have to follow the strict YANG formatting while composing

[19] Flexible NetFlow is a Cisco proprietary extension of classic NetFlow, that supports template-based telemetry. A template defines the attribute and their corresponding statistics that the NetFlow collector accumulates. For more details visit https://bit.ly/3D6FzJ3

[20] For more details on the HTML based models visit https://www.openconfig.net/docs/

the model. As most programming languages natively understand JSON documents, no additional translation is needed.

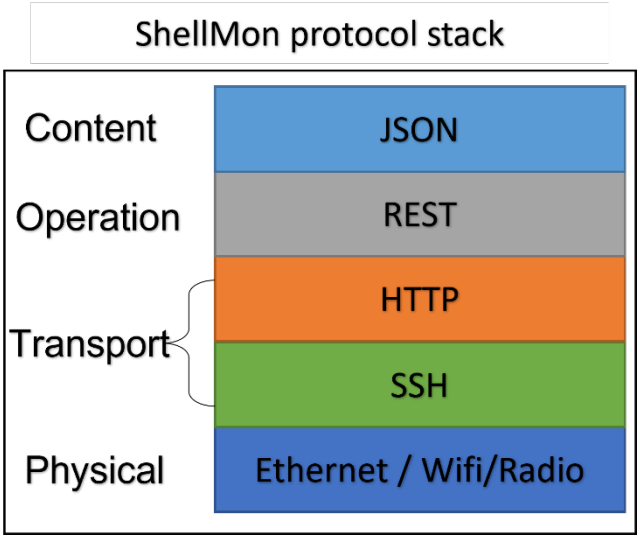

Figure 43 ShellMon protocol stack and NETCONF compliance

2. **ALL-RESTFul transport:** The transport mechanism of a telemetry API is crucial, as, in scalable infrastructure, it could affect the synchronization speed. Standard APIs such as SNMP and NetFlow use raw sockets for transporting the data over a negotiated port number. It could be a single point of failure if an intermediate firewall blocks that port. Eventually, every time a dynamic port is negotiated between the agent and the Collector, all the intermediate firewall needs to be provisioned by a new rule, dropping the former. Therefore, RESTCONF provides the transport using HTTP-based RESTFul API, which every firewall allows by default. The HTTP body encapsulates the RPC operations, which are exposed as API endpoints. RESTCONF is available only in high-end network devices. Thus, it doesn't cover the whole spectrum, and lower-end devices need to rely on the traditional protocols with their caveats above. *ShellMon* provides a platform-agnostic All-REST(Figure 43) architecture that relies on device-independent libraries such as *Psutil* and *Napalm* that use device-specific drivers underneath. *ShellMon*

leverages multi-threading for accelerated in-device metric collection and abstracts the complete process by RESTFul API endpoints.

3. **Dual-Mode collection:** All existing telemetry protocols such as SNMP, NetFlow, NETCONF, and RESTCONF are pull-based. In a Pull model, a collector maintains a list of agent endpoints (e.g., IP address, Socket ID, or URL) and connects to a Database. The Collector polls each agent periodically, which triggers the agent to invoke the collection method. The agent replies with its locally collected metrics. *ShellMon* can operate in a Pull-based Model to comply with any RESTful collector using the standard method. Additionally, it supports a Push-based message queuing technique where the agent initiates the telemetry to an intermediate message broker using AMQP[21]. The message broker decouples the Collector and agent bond and introduces autonomy in scale and replication without informing each other for enhancing high availability, security, and load-balancing.

4.3.2. Features

1. **No-SQL Realtime Database**

ShellMon uses a NO-SQL [259] Database to store telemetry information at the collector side. As mentioned above, *ShellMon* uses RESTFul communication with JSON serialization format. The collected data is stored in a hierarchical format, e.g., Figure 44 depicts an example of the node utilization data model. An RDBMS model to keep a similar structure would require complex table design, primary and foreign key mapping, normalization, and functional dependencies. A NO-SQL database solves the issue by storing the JSON data and allows JSON-based queries. That said, migration to a big-

[21] For more information on AMQP, visit https://bit.ly/3ofd7yn

data analytics engine is seamless as most options such as Hadoop. Additionally, it supports NO-SQL databases with map-reduce for scalability.

2. **Secure data acquisition**

End-to-end encryption is crucial to provide confidentiality to the data during communication in transit. This eradicates eavesdropping from untrusted entities as they cannot decrypt the encrypted cypher. OpenSSL would be leveraged to create Secure Sockets Layer (SSL) certificates that would be used to encrypt communications during transit. OpenSSL is a robust and well-known tool for Transport Layer Security (TLS) and SSL protocols. It is open-source and free for commercial and non-commercial purposes. A private Certificate Authority (CA) is created using OpenSSL to generate a root certificate and private key. After that, the root certificate is added to all participating devices, and then all certificates that are created and signed will be inherently trusted by those devices.

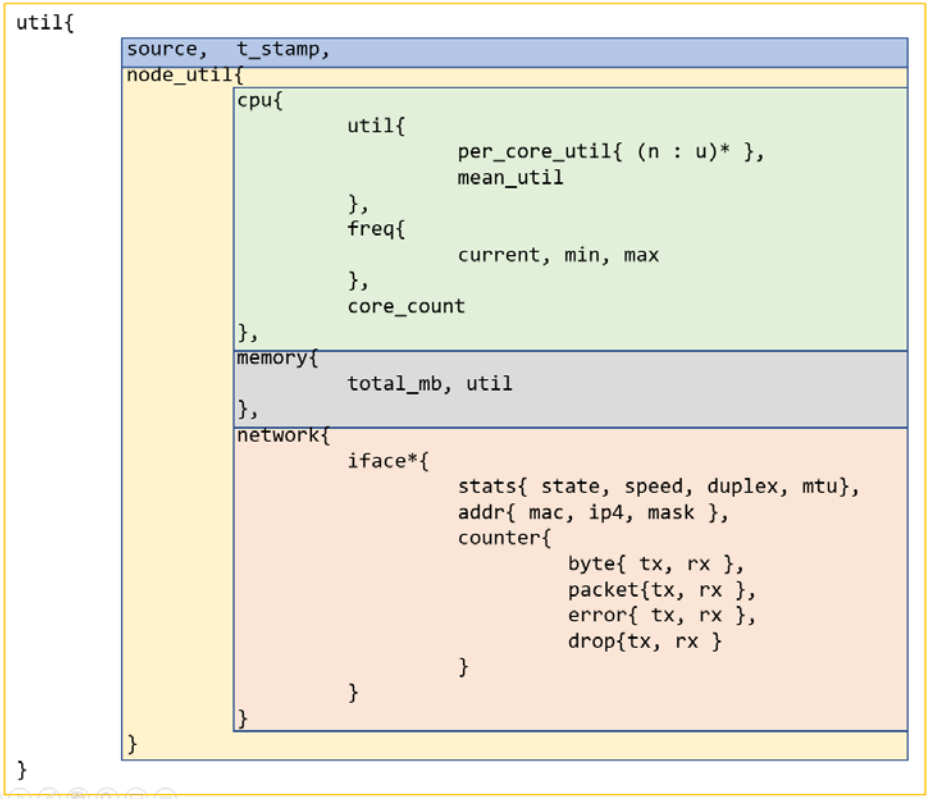

Figure 44 *ShellMon* data model for node utilization telemetry

3. **Asynchronous communication with AIOHTTP**

In a pull-based collection mode, the *ShellMon* Collector periodically polls the agents for their local metric. There are two bottlenecks in the mechanism, sequential polling of the agent is not scalable; therefore, asynchronous communication is used where each request-response takes place in an independent thread. However, HTTP, the carrier protocol for RESTful APIs, is not asynchronous by design; therefore, performance does not increase as if it were a full asynchronous communication. *ShellMon* uses a full-asynchronous RESTful transport by altering the HTTP to Asynchronous I/O HTTP (AIOHTTP [22]) based transport. Additionally, the database update is also multi-threaded; each session concurrently inserts the telemetry data to the No-SQL database acquiring a lock to eliminate any inconsistency. The Analyzer program reads from the Database parallelly and estimates statistics using moving average with a given window size to eliminate any spikes.

4. **Load Balancing and Fail-over**

Pull-Based Model: For a pull-based model, failover and load balancing require explicit implementation. *ShellMon* provides high modularity through its full-REST communication architecture. A single secure asynchronous RESTful fabric connects the clients, servers, and the Database. The host file lists the agents to monitor; therefore, load balancing can be achieved by segregating the agent list into multiple host files.

To eliminate the single point of failure from the Collector's perspective, it is vital to introduce multiple collectors. They will form a collector target group that is responsible for collecting data from the client nodes. Therefore, traffic would be

[22] For more information on AIOHTTP visit https://bit.ly/3bVgm8C

automatically distributed across the collector target group. This is to ensure traffic optimization so that no collector is overwhelmed. A control agent would then supervise the target group responsible for managing collectors and spinning up new collectors in case of a failover. This is achieved as each collector exposes a health check endpoint that the control agent uses to monitor them. If a collector fails three health point checks in a row, the control agent marks it as unhealthy. It then sends a notification to the admin. Additionally; it distributes the load of the unhealthy Collector to other healthy collectors.

4.1. **Push-Based Model:** A decoupled architecture has been used to ensure a smooth transition between a fell over and reinstating the server. A decoupled architecture guarantees that each component of the service can perform its tasks independently. Therefore, each piece remains wholly autonomous and unaware of the other. This allows for the functionalities running in each element to be self-contained. This is achieved by using a messaging queue. In this scenario, the collectors and the client nodes do not communicate directly with each other. However, they communicate using a messaging queue.

An Advanced Message Queuing Protocol (AMQP) [23] broker establishes communication between the collectors and the client nodes. AMQP is a lightweight messaging protocol that is used to transport messages between devices. It uses a publish-poll communication means and is ideal for connecting remote devices with a small code footprint and minimal network bandwidth. This is accomplished by using topics. Therefore, clients subscribed to a topic will receive messages published on that

[23] AMQP official documentation https://www.amqp.org/

topic. It uses a routing mechanism to deliver messages to queues based on the message routing key. Additionally, it supports authentication and encryption using TLS to ensure a well secure communication medium.

AMQP supports clustering natively; each AMQP broker synchronizes their queue with others. In a failover scenario, the brokers maintain seamless replication of queue contents. Therefore, the poll-based model provides load balancing and failover natively.

Chapter Summary

This chapter discusses the architecture and methodology to integrate a system-level simulator with a software-defined networking infrastructure using a relational database as middleware. Further, it gives a comparative analysis of using GNS3 and Mininet Wi-Fi-based DP. The sequence model shows the data and control flow among several percolating entities. The data model of the middleware describes the information structuring, and a set of algorithms to fetch and replicate will extend this architecture by appending an analytics plane on top of it. Data analytics empowered by deep learning algorithms will learn the run-time behaviour of the network and help to improve network automation, self-organization, and state prediction ability. The resultant architecture facilitates the end-to-end performance evaluation of 5G and beyond 5G use cases such as 5G-V2X where latency is a critical performance metric.

The *ShellMon* intelligent telemetry API architecture gives an insight description to facilitate data accumulation in a hybrid SDN architecture. The robust mechanism and the modular organization provide seamless extensibility and adaptability for future upgrades.

In summary, the chapter explains the compliance with the Self-Configuration property of the overall architecture for 5G and beyond Self-Organized networks.

Chapter 5: Self-Healing

Small cells are deployed on cellular networks' edges to provide low latency response to the nearby connected devices. In the case of robust traffic, small cells can offload their burden to the surrounding small cells, and this process is known as cell breathing. The other competing vendors mostly overlap the coverage area of one cellular operator, and there must be some reliable incentivizing mechanism to encourage resource sharing among the small cells of multiple operators. Uneven distribution of computation or communication load on the cell controllers results in a disparity in energy consumption. Service migration is a process that unloads an overloaded controller by transferring its services to an under-loaded one. A reactive migration always ensures optimal load balancing by their convergence and also meeting the scalability requirements.

Blockchain has already proven its relevance in maintaining trust and reliability among multiple independently operating entities. However, to the best of our knowledge, no one has proposed the blockchain for achieving reliable cell breathing among the small cells of multiple operators. This chapter discusses Cellchain, a blockchain network for achieving reliable cell breathing among the multi-operator small cells. Further, the chapter presents a proof-of-concept implementation over a virtualized SDN testbed that incentivizes reliable VNF sharing using containerized services deployed over the Raspberry Pi-based small cells. Experimental results show the efficiency of Cellchain for reliable cell breathing and the realization of tactile internet by confining the response time to 1ms based on the reliable infrastructure sharing between multiple operators.

5.1. Introduction

Cloud computing advocates [260] the availability of unlimited resources, and this illusion of infinite resources has attracted a considerable number of users in the last decade. This results in the tremendous increase of burden on the server infrastructure, and intercloud caters as a solution for tackling the burst traffic with limited cloud resources. Intercloud refers to the cooperative sharing of cloud infrastructure between different cloud service providers. Another solution to the increasing cloud burden is MEC), which refers to shifting computing from the cloud to the network edges.

MEC reduces the burden from cloud infrastructure and increases the Quality of Experience (QoE) by reducing the latency. However, as compared to the limited resources of the cloud, edge resources already exist in scarcity. Hence, the intercloud-like cooperation among multiple MEC providers makes more sense, but it is more challenging to achieve it in MEC. The collaboration in MEC cannot be dedicated to the central authority, like the intercloud, as it increases the latency and reduces the QoE.

The absence of central coordinating authority results in the trust deficit among the cooperating entities of MEC. Blockchain has established trust among the independently operating entities by shifting dependency from the primary entity to the independently operating entities. Hence, Blockchain is a well-proven solution for establishing trust among the intercloud-like cooperating entities in MEC, and this chapter presents the Blockchain-based reliable cooperation among multiple independently working MEC operators.

This chapter focuses on small cell-based MEC deployments. As evident from their self-descriptive name, small cells have limited capacity and resources and thus use the concept of cell breathing to offload their burden to the neighbouring cells. Small cells are crucial for the next-generation networks as they help achieve the constraint of 1ms latency in future networks.

However, instead of deploying the small cells in greater density, it is better to facilitate intercloud cooperation among multiple small cell operators.

With the emergence of the Tactile Internet, multi-operator cell breathing is no longer optional. Tactile internet supports heavy computational applications like virtual and augmented reality. This heavy computation in Tactile internet requires multi-operator coordination at the level of the small cell. This paper presents a Blockchain-based reliable, rewarding system for accomplishing the multi-operating cell breathing where each operator runs a blockchain miner (covered in subsection), which controls the data added to the blockchain. Each operator gets the time as a reward when one of its small cells shares the computation of another small cell of other operators, which in turn spend its already earned time reward.

Miners of multiple operators monitor the time-based rewards through a distributed consensus algorithm, and NFVs (Network Function Virtualization) host the miners on the small cell of an operator. Each small cell invokes a resource-aware self-migration algorithm (Algorithm 3) which switches the miner between different small cells of an operator to avoid the overburdening of small cells due to the execution of the mining process by the blockchain miners.

This chapter introduces Cell-Chain with a three-fold contribution, listed below.

1. The first contribution is a self-resource-aware migrating algorithm for seamless shifting of an NFV based blockchain miner among multiple small cells of an operator. It enables the integration of blockchain at the levels of the small cell without deploying the dedicated machines for executing the mining process of blockchain.
2. The second contribution is the implementation of CellChain as a consortium-based blockchain to achieve intercloud-like reliable resource sharing among the small cells at the edge of the network. The consortium-based blockchain ensures the restricted access of the CellChain among the pre-selected operators.

3. The third contribution is to implement the time-based rewarding system, monitored by the distributed consensus among the miners of consortium members of CellChain. Raspberry Pi with a wireless antenna has been used to present a small cell and has deployed a testbed of multiple small cells of various operators.

Experimental results of this testbed depict the effectiveness of CellChain in reducing the burden of small cells through the multi-operator cells breathing.

5.2. Cell Breathing: An enabler for Tactile Internet

This section describes the small calls, cell breathing, and tactile internet, along with the importance of cell breathing for Tactile Internet [261].

5.2.1. Evolution of Internet-based communication

Initially, the internet originated as a network of wire-connected systems for file sharing. With time, it has gone through the following evolutionary phases for supporting the human-to-machine interaction in tactile internet (Table 18).

Evolution	Description
Fixed Internet	The first generation of the internet and based on the land-line telephonic infrastructure.
Mobile Internet (H2H)	Currently deployed and focuses on human-to-human (H2H) communication by providing audio, video, and data exchange services.
Things Internet (M2M)	It focuses on machine-to-machine (M2M) communication and is currently being practised at a limited scale compared to H2H communication and aims for full coverage after the deployment of 5G.
Tactile Internet (H2M)	It focuses on real-time human-to-machine (H2M) interaction with up to 1ms latency. It will also enable humans to control the next generation of robotic systems remotely. It is currently an active topic of research and is yet to get deployed in production.

Table 18 Evolution of the Internet

5.2.2. Classification of Small Cells

Small cells are radio access nodes of low-power and have a coverage range of a few meters up to a mile in radius. Small cells are opposite to Macrocells which have a range of up to 20 miles. Small cells can be either of the following three sub-categories based on scope and capabilities (Table 19)

Small Cell type	Description
Femtocells	They are usually installed by the user and are limited in capacity and coverage. They can be deployed for a building and can support very few simultaneous connections.
Picocells	They are also known as indoor Metrocells and support up to a hundred users with a range of around 250 yards.
Microcells	They are also known as Metrocells or outdoor Metrocells and are closer to the Picocells' capabilities, but they may have a coverage range up to a mile. Microcells can also use power control to limit the coverage range.

Table 19 Classification of Small Cells with their brief descriptions

5.2.3. Cell Breathing and its Importance for Tactile Internet

Macrocells are well equipped with enough resources to support the considerable traffic. In contrast, Small Cells are limited in capacity and resources; therefore, in case of burst traffic, small cells offload their burden by sharing it with Macrocells, and this concept is known as cell breathing. However, cell breathing is not limited to Macrocells, and small cells can also offload burden with each other to accomplish cell breathing. As tactile internet requires more computation for mapping the H2M interaction, cell breathing is essential for supporting its high computational needs.

Usually, the neighbouring cells of a small cell belong to other operators, and intercloud-like multi-operator cell breathing can help the operators offer the services with relatively fewer small cells. The collaboration at the intercloud level is reliable as the central authority in the cloud accomplishes it, but similar collaboration at the Small Cell level is not reliable due to the

absence of a dedicated central authority. The CellChain model addresses the problem above by offering distributed trust in the small cells of multiple operators.

5.3. CellChain: An enabler for Reliable Multi-Operator Cell-Breathing

This section elaborates on the role of blockchain in achieving reliable, rewarding system-based cell breathing among the small cells of multi-operators.

5.3.1. Blockchain for Reliable Multi-Operator Cell Breathing

We are using the consortium blockchain, which is one of the blockchain network types [262], to allow the cells that belong to the pre-defined operators as the consortium members. Every participating operator gets a miner, which is responsible for adding new blocks (data) to the blockchain and thus controls the growth of the Blockchain [263].

5.3.2. Blockchain for Reliable Rewarding System

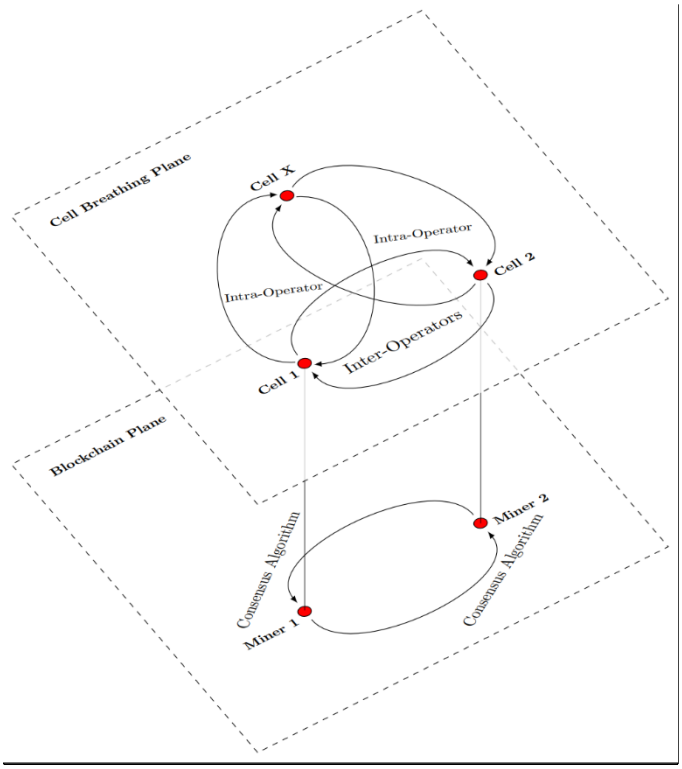

Figure 45 Reliable Rewarding System for Inter-operators Cell Breathing

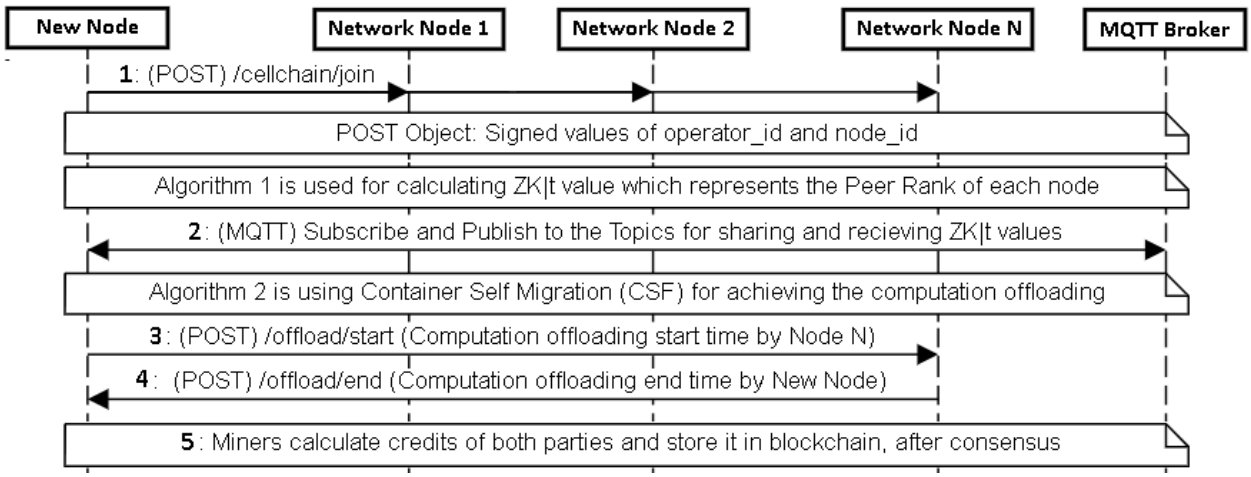

Figure 46 Execution flow of CellChain

Figure 45 shows two planes of Blockchain and the cell breathing plane. Both of these planes exist parallel to each other. Both $Cell_1$ and $Cell_2$ in the Cell Breathing plane belong to different operators; therefore, when cell breathing occurs between them, its details will be reliably stored in the blockchain after executing consensus algorithms. However, we assume that any $Cell_X$ if shares computation with another cell must be under the same operator. Hence, the blockchain-based rewarding system will activate when cell breathing occurs between the same operator's small cells. Otherwise, the total time of cell breathing becomes the reward to the operator whose cell is performing the computation on behalf of the other cell. This time is earned as a reward for the future to offload the computation of its own.

5.3.3. Execution flow of Blockchain-based CellChain

Figure 46 shows the execution flow of the cell chain. The following steps explain the execution details of the CellChain:

1. A new node that is associated with a member operator is sending a request to join the cell chain. The operator responds to it but also broadcasts its information to the other operators.
2. The new node subscribes to the MQTT broker to share its peer rank and know the peer rank of other small cells. It uses Algorithm 3 for calculating its Rank.
3. Algorithm 4 invokes the container migration process and observes the cell breathing. This step also broadcasts the starting time of the cell breathing to store it in the blockchain.

4. Broadcast the end-time of the Cell Breathing process, which the Blockchain stores.
5. A consensus algorithm results in shifting the credits between the participants of the Cell Breathing process.

5.4. Design and Implementation of CellChain

The experimental setup uses Docker containers for hosting the two types of VNF. The first type is related to the miner, and instead of confining it to a dedicated machine, containerized services can migrate to the Small Cells of the same operator. The other VNF is for computation sharing, and they stay in both small cells of the same operator and the operator.

5.4.1. System Architecture

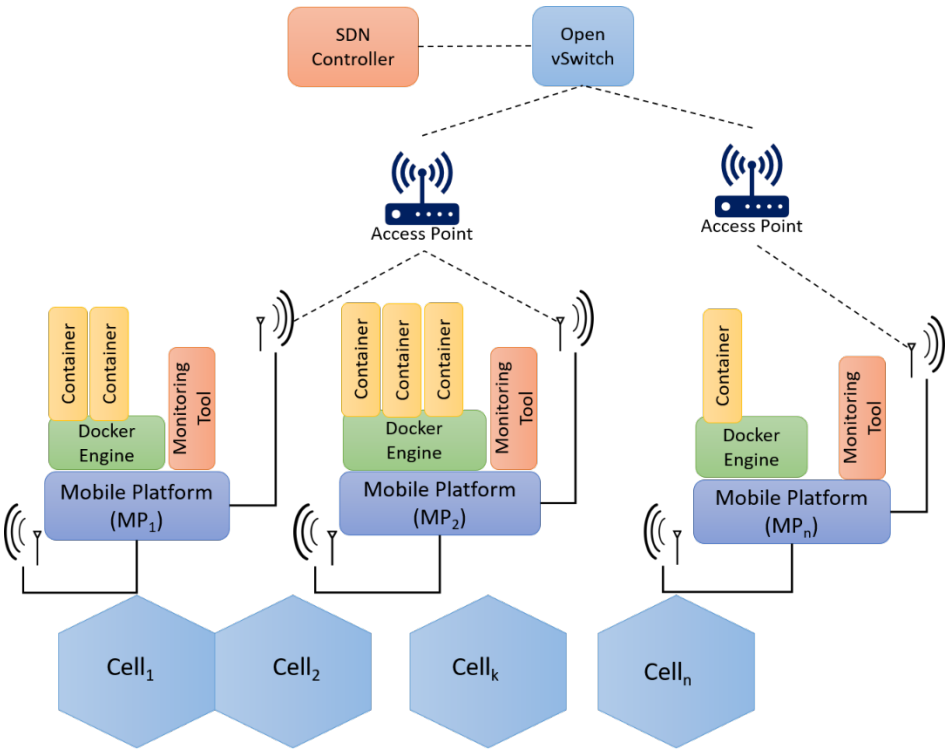

Figure *47* System Architecture of CellChain Wi-Fi

Figure 47 depicts the schematic diagram of the system. A Mobile Platform (MP) is a node that hosts several VNFs as docker containers. An MP equips at least two antennas; one connects the

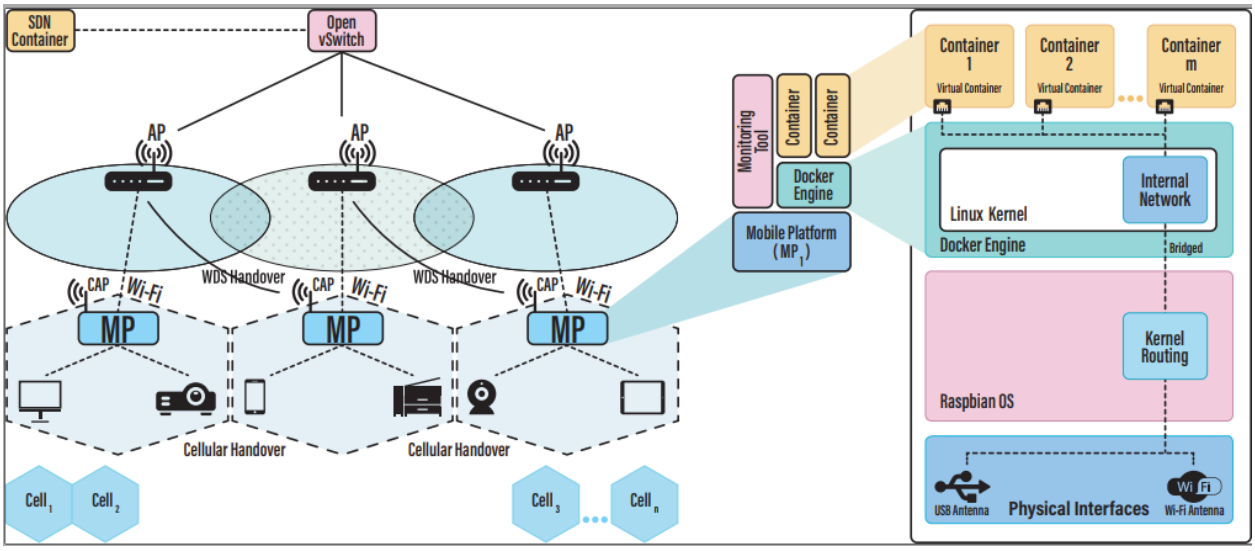

Figure 48 Physical implementation of SmallCell using Raspberry pi and Docker

end devices (e.g., Fronthaul or access), and the other connects among MPs (e.g., Backhaul or Core). An SDN controller governs the inter-MP communication using SDN switches.

5.4.2. Implementation of the CellChain RAN

Figure 48 shows the physical implementation of the Radio Area Network (RAN). APs are stationary whereas Mobile Platforms (MPs) such as an AP sitting in a vehicle may move. Each MP has at least two Wi-Fi antennas. The Backhauling connection with the AP provides connectivity to the IoT devices as a Co-Access Pont (CAP). MP works as a gateway for the associated IoT devices and hosts the broker application different IoT protocols use. It also runs a DHCP & DNS server to isolate the IoT logical address space from the underlying network. Each CAP support handover through Wireless Distribution System (WDS). We assume all the MPs share the same SSID and IP address space to accommodate all connected IoT under the same subnet. The APs are the Internet Gateways for the network. To avoid interference, CAPs run at 2.4GHz and APs at GHz frequency[24]. Communication between APs and MPs mimics

[24] The 2.4GHz band has less FSPL and higher coverage than but less throughput compared to the 5GHz band. In case of multiple fronthaul antennas, they must be on non-overlapping channels.

the Backhaul, and MPs and IoT end devices are fronthaul communication concerning the actual telecom establishment.

5.4.3. Monitoring & Containerization with Docker

The experiment uses Raspberry Pi as physical MPs and APs are Open V-Switches (OVS), as depicted in Figure 48. Each runs two main applications, A Python-based custom monitoring tool called *ShellMon* and Docker engine. In this architecture, the MP host containerized services using Docker. Containers help keep the application code and its runtime together, save space as no kernel installation is needed, and thus result in faster migration. *ShellMon* periodically collects the resource utilization information such as CPU, memory, storage, bandwidth, remaining battery, etc., and computes a cumulative utilization metric called Z-value. A container is selected if the Z-value exceeds a certain user-specific threshold, and the migration process gets initiated to a suitable target. The OVS nodes connect to an SDN controller such as OpenDaylight over an out-of-band OpenFlow channel

5.4.4. Configuration for Internal Communication

Raspberry Pi runs Raspbian Operating systems. Each raspberry Pi board embeds a physical Wi-Fi antenna used for connecting APs and external USB Wi-Fi antennas to connect the end devices. Figure 48 depicts how the container networking between the Raspbian kernel and the Docker engine. Raspbian is a Linux-based distribution that natively supports kernel routing. Docker engine running on top of the kernel supports both Bridge and Network Address Translation (NAT) mode. The Default network configuration of Docker is NAT; however, the experiment uses a bridge connection so that the containers share the same IP address space as that of the physical network. It also preserves the container IP address, even if it migrates to a different engine. DHCP and DNS services may run as a container or directly on the kernel. Whenever an end device wants to connect, the subjected MPs respond to the DHCP Discover

message and provide IP addresses from the DHCP pool to isolate the end-device address space from the container address space.

5.4.5. The Cell Rank Algorithm

The Cell Rank Algorithm (Algorithm 8) is one of the two principal algorithms of *ShellMon*. It runs as a subroutine of the other one. Cell Rank performs the following tasks:

1. It periodically gathers local resource information and computes cumulative utilization metrics (Z_{value}).
2. It Broadcast local Z_{values} among the other MPs.
3. It Receives Z_{values} from remote MPs.

The following stages describe the working mechanism of the Cell Rank algorithm.

1. **Initialization:** The algorithm starts with an empty dictionary structure (Z_{dict}) keyed at the IP address of MPs in the network and valued with their respective Z_{values}.
2. **Update Items:** When a new MP gets added, it broadcasts a subscribe (SUB) message and its IP address in the network. After receiving a SUB, all other MPs add the subscribed IP to their respective Z_{dict} valued with 0. Likewise, while leaving the Network, it broadcasts an unsubscribe (UNSUB) message, which other MPs remove the corresponding entries. If no updates appear for a certain amount of time (user-specific), the algorithm treats the silent MP as dead and eventually removes the corresponding entry.

Parameter	Usage	Measurement
U_c	CPU Utilization in percentage	$[0,1] \in \mathbb{R}^+$
N_c	Number of CPU cores	$\mathbb{N}$
N_M	Memory volume	*GB*
U_M	Memory Utilization in percentage	$[0,1] \in \mathbb{R}^+$
N_s	Free storage space	*GB*
U_{BW}	Bandwidth utilization (Load on the interface)	$[0,1] \in \mathbb{R}^+$
U'_{BAT}	Remaining battery (set to 1 if line-powered)	$[0,1] \in \mathbb{R}^+$

Table 20 Cell Rank parameters

3. **Collect Resource Information:** The Cell Rank algorithm periodically accumulates the following parameters from the devices to decide a migration trigger (Table 20).
4. **Normalization**: At any given instance t, the cumulative utilization $Z_{E|T}$ is the normalized Z-value of an edge node E (Eq. 29).

$$Z_{E|t} = Normalize(U_c, U_M, U_{BW}, U'_{BAT}, N_c, N_M, N_S) \quad \text{Eq. 29}$$
$$= \alpha_1\left(\frac{N_M}{U_C}\right) + \alpha_2\left(\frac{N_M}{U_M}\right) + \alpha_3(1 - N_s) + \alpha_4\left(\frac{1}{U_{BW}}\right) + \alpha_5(1 - U'_{BAT})$$
$$\forall \alpha_i \in [0,1]\,, \sum_i \alpha_i = 1$$

   The normalization uses the CPU and memory utilization per core and volume, respectively. The inverse of the storage and battery utilization as less space and battery charge increases the utilization cost. A_I are weighing coefficients by default set as $\frac{1}{5} = 0.2$ to give equal priority to all components. The priority values are subject to user choice for influencing one component than the other, given they summed up to 1.
5. **Publish and Receive:** All MPs periodically publish $Z_{E|t}$. If any communication exception happens, the MPs keep retrying until a timeout event occurs. The period sleeps for a time-interval $T_{Interval}$ and collects $Z_{K|t}$ from other K MPs except E to prevent redundancy and broadcast storm. Z_{Dict} holds the $Z_{K|t}$ entries and gives constant time lookup.

5.4.6. The Cell Breathing Algorithm

This algorithm is responsible for initiating the container migration (Algorithm 8). It takes three inputs, $Z_{K|t}$ (temporal utilization of MPs from Cell Rank algorithm), Z_{Cut} (the utilization threshold after the migration is triggered), Tout (A sliding time window for calculating the utilization's moving average). Let it be running from MP_k; the following are the steps of the algorithm. This section explains the Algorithm for Cell Breathing.

1. **Spin Lock:** The algorithm waits until the local utilization $Z_{E|t}$ exceeds the threshold Z_{cut}. In such a case, it checks if the mean utilization for the time window T_{out} has crossed the threshold value. This prevents any accidental trigger due to a spike in utilization.
2. **Find Migrant Container and Target**: If the mean utilization surpluses the threshold value, the algorithm selects a target MP from the local dictionary Z_{dict} with the minimum utilization. Based on the round-trip time (RTT), a decision variable defines the candidate boundary to comply with the Low-Cost Low-Delay (LCDC) constraint, which prevents any infeasible migration decision to a node which most favourable by its utilization perspective; however, it is too far that the migration process might take longer. The next task is to find the Migrant container. Among the running container from the source MP_E, the algorithm finds a container C_q such that it is consuming maximum resources. The idea is to select a Migrant container that will relax the overall load the most.
3. **Prepare Migration**: The algorithm first saves the running state of the Migrant container (Pre-Copy) by committing the running container into a compressed image file (*Cq.img*). However, the container keeps running. The pre-copy option helps preserve the state of a service when migration is anticipated. Additionally, the unique hash that

is calculated for the Blockchain (discussed in a later section) remains consistent if the service is a snapshot using the pre-copy method.

4. **Migration**: Assuming the MPs use Message Queueing Telemetry Transport protocol (MQTT) to transfer the *Cq.img* file from MP_E(source) to MP_T (target). The migration process initiates a stream transfer; alternatively, via Secure File Transfer Protocol (SFTP).
5. **Restart Container**: If the transfer completes successfully, the target MP first decompresses the image file, then loads it into its local image registry and signals the source MP about the job completion. The source MP then stops the Migrant container locally and acknowledges the target MP, which runs the container at the remote site. The handshake mechanism reduces the downtime of the service significantly.

Algorithm 7 **Cell Breathing**

Input: $Z_{K|t}$, Z_{Cut} , T_{out}

Output: Boolean

Data Structure: Ordered List

Steps:

```
Do
    While (Z_K|t < Z_cut) do
        Z_tot = 0                                   // initializing local sum
        For t in [0 : T_out] do
            Z_tot = Z_tot + Z_E|t                   // cumulative utilization
        [end loop]
        If (Z_tot / T_out) ≥ Z_cut then             // Mean utilization exceeds the threshold
            [LABEL-1 ]
            P = min_i Z_i  ∀i ∈ {MP_k} , i ≠ E       // find Target
            Q = max_j C_j ∀j ∈ {C_j} on MP_E         // find Migrant
            Commit C_Q
            C_Q.img = Compress(C_Q)
            STATUS = send( C_Q.img, MP_P )          // send C_Q.img to MP_P
            If STATUS == OK then                    // successful migration
                Remote login to MP_P
                Stop C_Q at MP_E
                Run C_Q at MP_P
                Exit remote session
                FLAG = True
                FAIL = 0                            // reset failure counter
            Else
                FAIL = FAIL + 1                     // increase failure count
                If ( FAIL > MAX_FAILS ) then
                    Break                           // exit if max retry attempt exceeds
                [end if]
                Goto LABEL-1                        // unsuccessful unsuccessful, retry
            [end if]
            FLAG = False
        [end if]
    [end loop]
While (True)
```

Algorithm 8 **Cell Rank Algorithm**

Input : $T_{Interval}$, $Addr_{Broker}$

Result : Raises Exception is Broker or Network malfunction is detected

Data Structure: Dictionary $Z_{dict} = \{k : Z_{K|t}\}$, Message Queue

Steps:

Do

If (EVENT == "Attachment Req") **then**

K = Req.ID *// fetch node ID from request message*

$Z_{dict}[K] = 0$ *// initialize node K's util as 0*

Else If (EVENT == "Detachment Req") **then**

K = Req.ID

Z_{dict}.drop(K) *// remove the entry of K from Z_{dict}*

[end if]

//Collect instantaneous system information of node MP_E at time t

U_c = SYS.CPU_UTIL

N_c = SYS.CPU_CORE_COUNT

U_M = SYS.MEM_UTIL

N_M = SYS.MEM_VOL

N_S = SYS.STO_AVAIL

U_{BW} = SYS.NET_LOAD

U'_{BAT} = SYS.BAT_AVAIL

$Z_{E|t} = Normalize(U_c, U_M, U_{BW}, U'_{BAT}, N_c, N_M, N_S)$ *// Eq. 28*

Publish $(payload = Z_{E|t}, topic = E)$ *// publish the utilization*

If not (Boker_Exception || Communication_Exception) **then**

Collect $(Z_{K|t})\ \forall k \in MP, K \neq E$

For K in Z_{dict} **do**

$Z_{dict}[K] = Z_{K|t}$ *// Update the dictionary*

[end loop]

Sleep($T_{interval}$) *// Sleep*

Else:

Raise Exception

[end-if]

While (True)

5.4.7. Time Series Prediction of Z_value

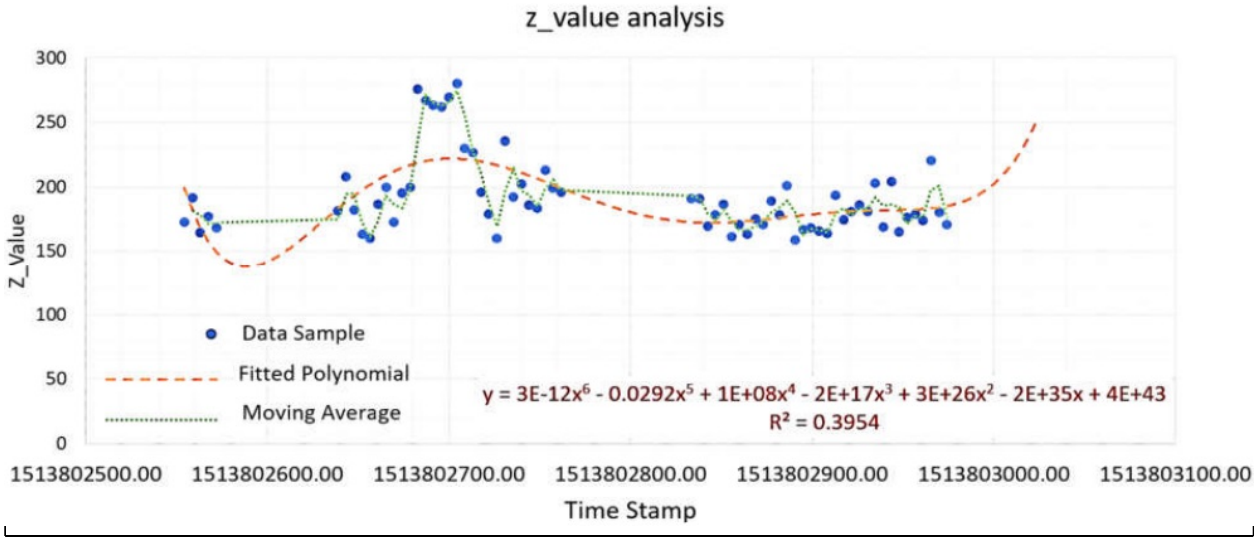

Figure 49 Forecasting of Utilization by extrapolating fitted polynomial

Calculating the mean utilization during the Spin Lock phase of the Cell Breathing algorithm (Step 1) involves a prediction component. Values from time-window T_{out} fits into a polynomial fit algorithm. The fitted polynomial is then extrapolated up to 3rd order. The obtained value is returned. This may predict a value for a growing $Z_{k|t}$; it also ignores the outlying spike for fluctuating graphs. (Figure 49)

5.5. Implementation of the Migration Framework

Figure 50 depicts the deployment diagram of the container migration framework. The nodes represent a physical or virtual network function (e.g., Raspberry Pi, Routers, switches, etc.) connected over a shared network. Every node hosts an operating system that runs a Docker engine (for VNF placement) alongside *ShellMon* (for resource monitoring) and Migrator Agent that listens to the migrator-orchestrator for triggering migration. Relocatable VNFs are run as

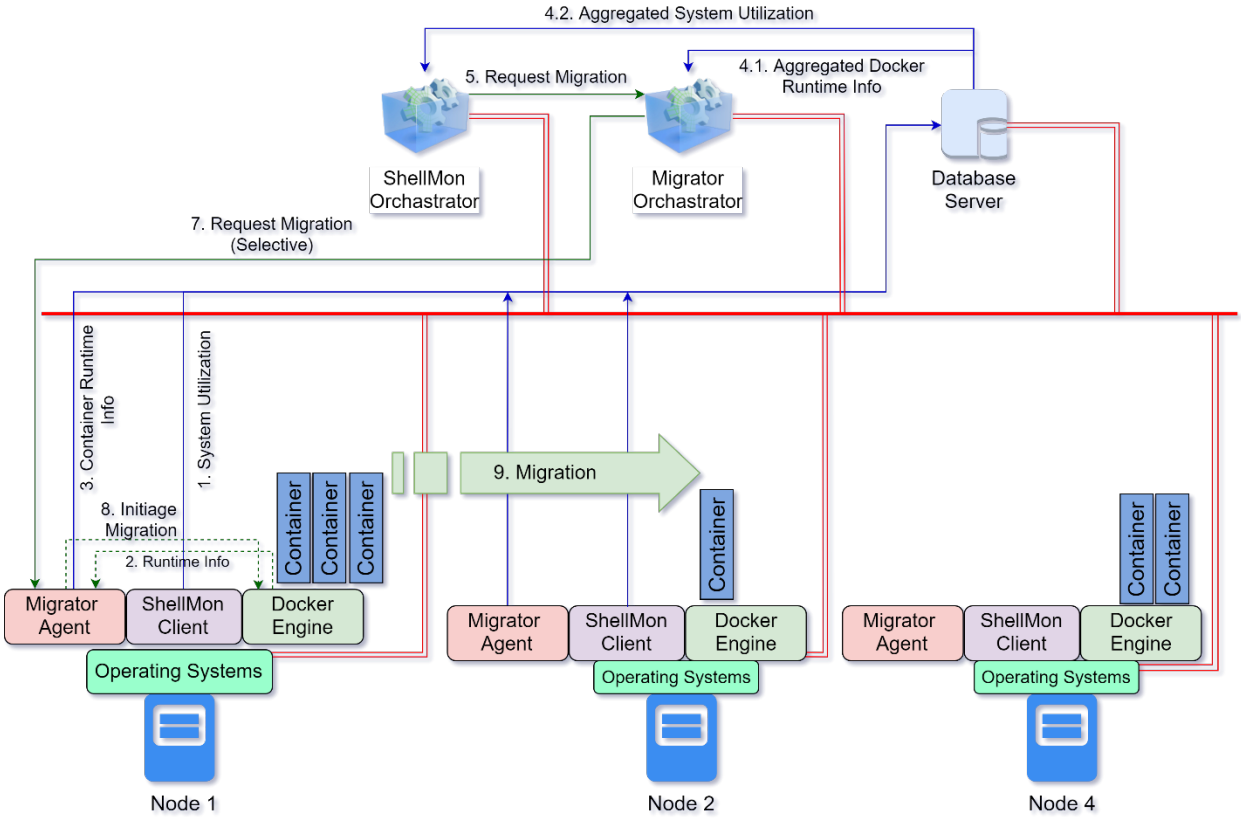

Figure 50 Communication model for VNF Migration

VNFs on top of the container engine. Modes of communication between migrator-orchestrator and agents are of three types discussed later in this section.

5.5.1. Migration Process

This section describes the experimental setup for the migration orchestrator (Figure 50)

1. The end devices such as IoT or UE send service requests such as Discover messages, Data etc., to a gateway device hosted by the MP. In the experiment, we used Raspberry Pi 3Bs as the MPs
2. Gateway redirects the request to a Docker container running the appropriate VNF to service the request.
3. The VNF returns the outcome to the gateway, which it eventually replies to the end device

4. Resource utilization of the local device periodically relays to the *ShellMon* program that runs the Cell Breathing and Cell Rank algorithm to monitor and analyze the utilization patterns.
5. *ShellMon* receives utilization data of remote MPs and broadcasts its local data.
6. If Migration is needed, *ShellMon* calls for it, and the Cell Breathing algorithm initiates the procedure
7. Migration takes place between a pair of MPs
8. The immigrant container gets detached from its source and attached to the destination MP and restores its state.
9. The end device re-establishes the communication with the containerized VNF using new the MP

5.5.2. Communication Modes

Seamless communication between the migrator-agent and the orchestrator is critical for the process. We propose three modes of communication.

1. **Peer Mode**: In peer mode (Figure 51), the agent communicates with the orchestrator using a secure TCP socket. Agents encapsulate periodic updates and send them to the orchestrator. Similarly, the migration request is sent by the orchestrator to the agents.

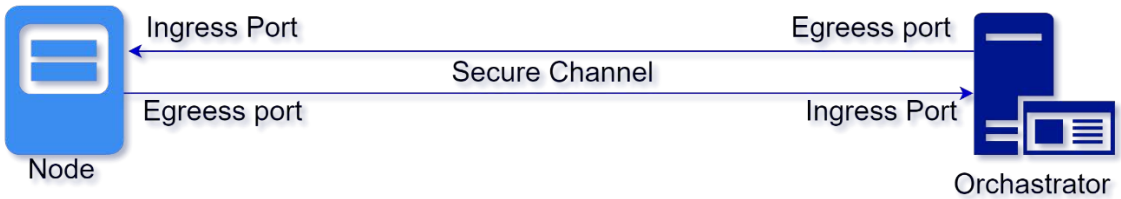

Figure *51* Communication in Peer mode with Sockets

2. **Database Mode:** In database mode (Figure 52), agents don't communicate with the orchestrator directly; instead, they periodically update the information to a logically centralized database server, from which the orchestrator fetches. In our experiments, we used both SQL (using MySQL) and NO-SQL(using Hadoop-HBase) databases.

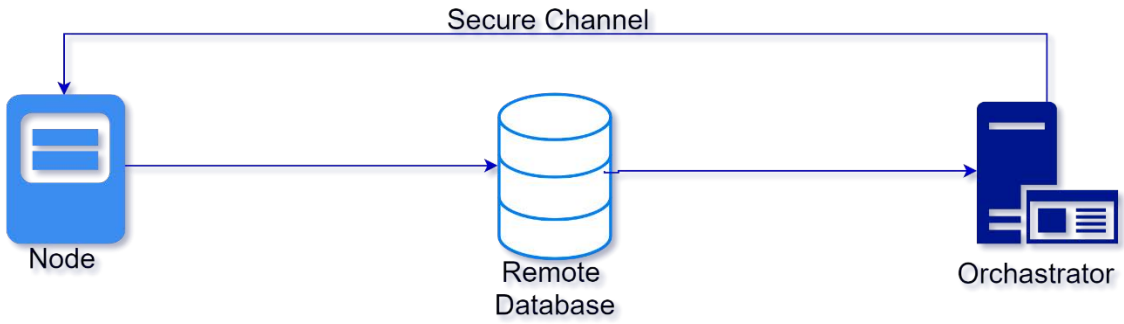

Figure *52* Communication in Database mode using the centralized database server

Broker Mode: The Broker mode (Figure 53) uses an intermediate message broker, through which agents communicate with the orchestrator. In our experiment, MQTT is the telemetry protocol. Agents publish monitoring information using the broker, from which the orchestrator subscribes. The migration request, specifying the source and the target nodes, is Published by the orchestrator and received by all nodes, but only the subjected nodes engage themselves in the migration process.

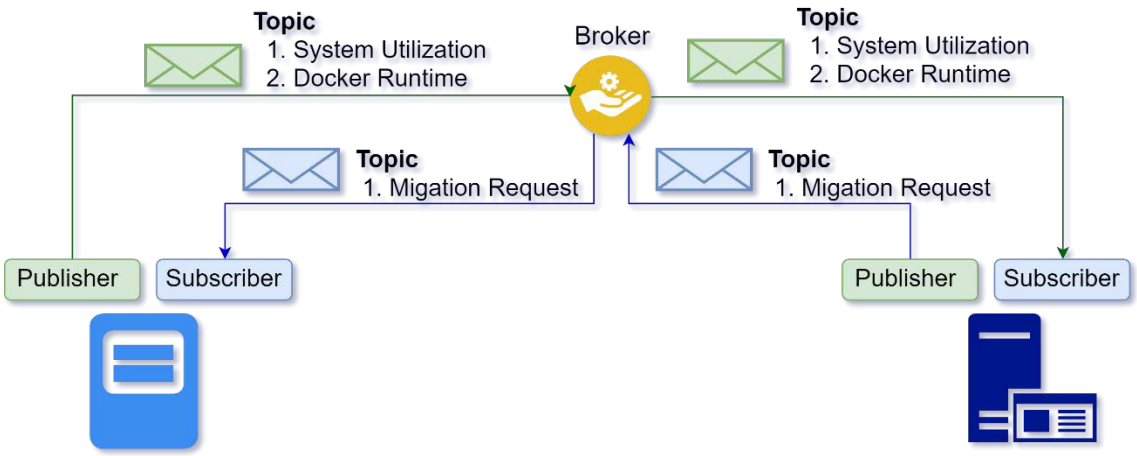

Figure *53* Communication over broker mode using MQTT

5.5.3. Blockchain Integration

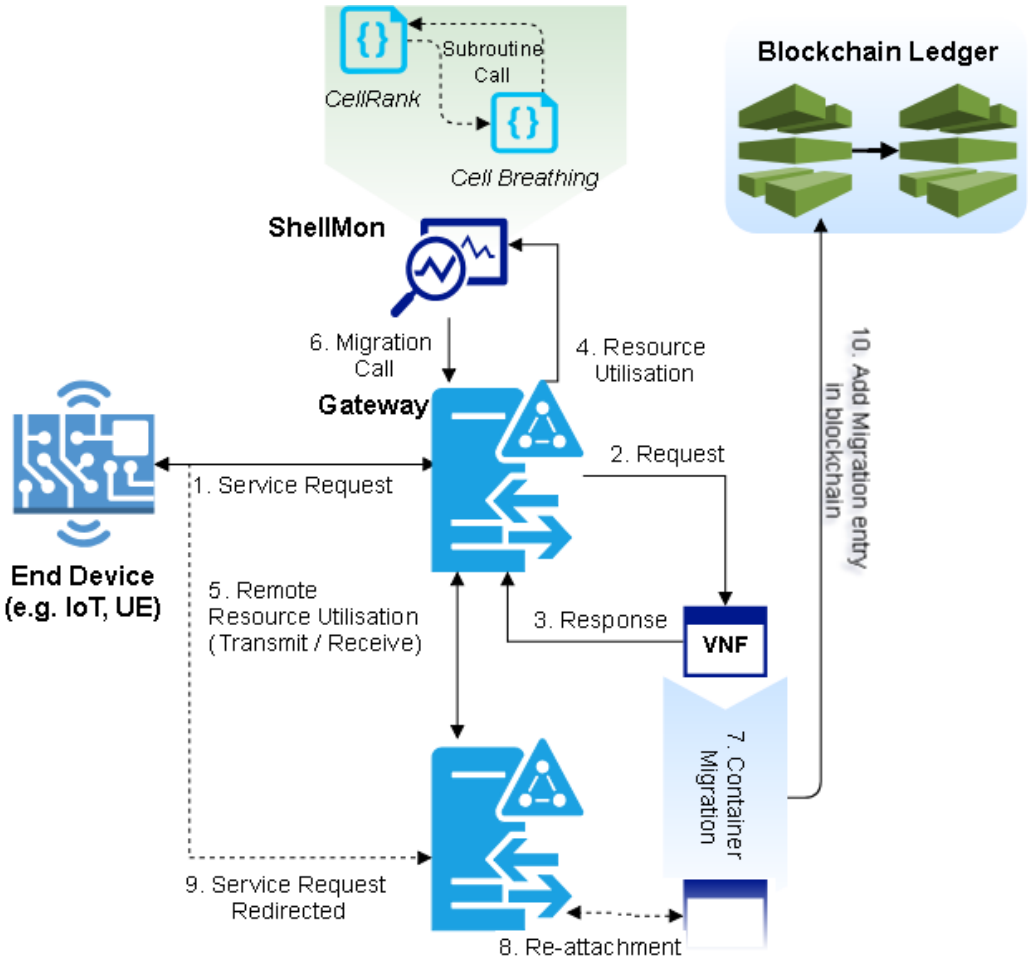

Figure 54 Integrating VNF migration across Small Cells

VNF originated from one node in this architecture and keeps migrating across the network based on the instantaneous load on the nodes and QoS. The migration of any VNF throughout its lifetime is critical to be traceable, as it is vulnerable to any rouge VNF injected into the system and tracked the mobility. The proposed architecture uses Blockchain integration to serve this purpose. Every migration is treated as a transaction and is recorded into a Blockchain. Since Blockchain is immutable, every VNF can be tracked down to its origin. Figure 54 depicts the blockchain integration.

5.6. Experimental results

We have used two different types of VNFs. One VNF is for the miner, which can only be shared within the Small-Cells of the same operator, and the other VNF is for the computation and can

be shared within the Small Cells of the same vendor or across different vendors. To automatically shift the VNFs between different small cells, we have implemented Algorithms 1 and 2. We have also performed experiments to find the accuracy of implemented algorithms, and this section covers experimental setup and experimental results.

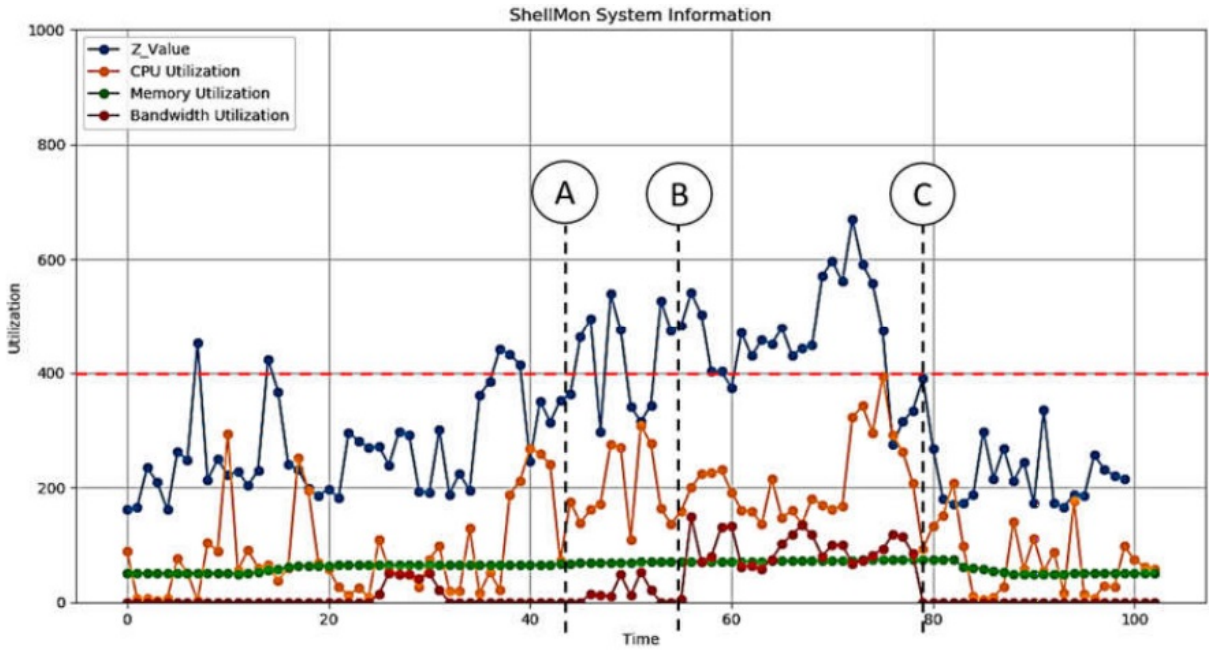

Figure *56* Self-triggered Migration of VNF

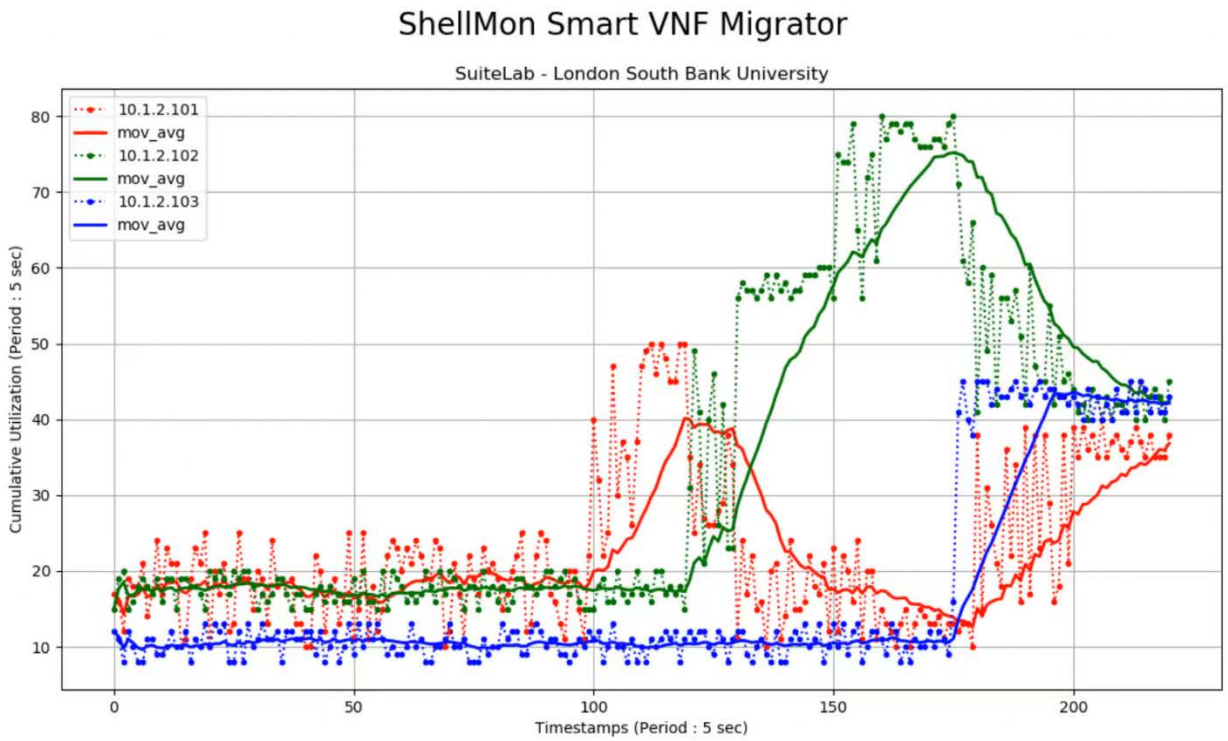

Figure 55 Convergence of node utilization

Figure 56 shows the experimental data acquired from an arbitrary MP demonstrating self-migration. In this graph, X-axis denotes the time window of 100 seconds, and Y-axis denotes utilization in the $[0,1000]$ range. The red dotted line passing through the 400-utilization mark denotes the threshold value (Z_{cut}) at 40%. The remaining colour-coded curves are labelled in the legend. The Following describes the migration. In the interval $[0, A]$, there is some instance when Z_{value} curve crosses the threshold mark. But as the majority stays under the limit, the Cell Breathing algorithm triggers the migration. In the interval $[A, B]$, some CPU and network activity takes the utilization mostly above the threshold value. Hence the Migration is triggered. In instance B, a sudden activity in-network utilization is registered due to the container image transfer, which continues until time C when the transfer commences. The victim container is also stopped; thus, the CPU utilization comes down.

Figure 49 shows the time series prediction of Z values. The X-axis denotes the relative times-tamps (in seconds), and the Y-axis is the Z-Values encountered at the corresponding time with dots representing the samples. The time series prediction fits a polynomial over the data samples and forecasts the utilization by extrapolating; hence the prediction is made. Since the curve fit follows the moving average of the data samples, it doesn't consider the outlying spikes. Figure 55 shows the convergence property of the migration algorithm which includes three nodes during the experiment. Moving average-based prediction smooths the fluctuations and removes any spikes. At first, the Red node is stressed; when the stress level exceeds the average load of the network, tasks from the Red node are offloaded to the Green node. The Green-node is then stressed further, which results in tasks being offloaded to both the Blue and Red-node, and convergence is met.

Chapter Summary

Cell breathing allows small cells to offload their burden to the nearby small cells for continuing the seamless services. Cell breathing among neighbouring small cells of multiple vendors enables the service providers to cover more areas with comparatively fewer resources. However, this multi-vendor collaboration can only be realized after the trust is established between cooperating operators.

This Chapter provides a trust management solution at the small cell level, without involving a central entity. This is achieved through a consortium blockchain where each vendor is hosting a miner at its co-located small cells. Instead of using the dedicated nodes for running the miners, the *Cell Rank* and *Cell Breathing* algorithm collaboratively transfer containerized services between different small cells of a vendor. The resultant architecture can automatically offload the computationally intensive jobs between small cells of different vendors, but miners are only transferred to the small cells of the same vendor.

In summary, the chapter explains the role of service migration as a tool to comply with the Self-Healing technique in 5G and beyond. Additionally, the self-migration architecture ensures security by the Blockchain integration to prevent any MITM and DDoS attacks.

Chapter 6: Conclusion & Future Directions

6.1.Conclusion

This thesis investigates the design and development of a cognitive routing framework compliant with the Self-Organization characteristics of the Knowledge Defined Networks (SO-KDN) targeting the requirements of the fifth generation and beyond communications systems. The contributions to knowledge address all three layers of SON: self-Healing, self-configuration, and self-optimization. Chapter 1 introduces the key concepts relevant to the discussions conducted in the following chapters and outlines the research contributions: SD-WAN, SDN integration, SDN-Routing, and Cognitive routing. Chapter 2 investigates the state-of-the-art of elementary topics which have motivated and helped to reason the research framework. These are QoS aware routing, Hybrid SDN architectures, ML's application to routing optimization, Self-Healing technologies, and 5G SON.

The Self-Optimization problem is covered in Chapter 3, which introduces three novel algorithms (*STEN*, *MRoute*, and *MRRF*) that work in conjunction to accomplish routing optimization in a KDN. Firstly, Stochastic Temporal Edge Normalization (*STEN*) devises a method that fuses node and link costs. This process results in an isomorphic transformation to the underlying network topology, which helps the routing algorithms to avoid nodes that are computationally overloaded while computing routes. Secondly, the Multi-Route (*MRoute*) algorithm takes a hybrid routing approach, where it proactively enumerates all possible paths between all pairs of nodes in the network topology and stores them in the form of a state-machines, which in turn get distributed in case of a larger multi-controller deployment. The runtime of *MRoute* uses a reactive approach to rank the discovered routes using reliability metrics. The use of data structures with a constant-time lookup capacity and proactive topology

enumeration results in a constant-time routing convergence. MRoute leverages the *STEN* while computing the inter-node costs. Thirdly, Most Reliable Route First (*MRRF*) analyses the change in costs and measures the reliability of the links using deep-learning models. As a result, the optimal routes are calculated based on the anticipated reliability rather than the costs themselves, which prevents frequent route flapping due to cost fluctuation and returns the most stable routes that eventually contribute to the 5G paradigm of Ultra-Reliable Low-Latency Communication (URLLC).

Chapter 4 focuses on the Self-Configuration feature, where the controller injects the routes computed using the trio *STEN, MRoute,* and *MRRF* into the underlying Data Plane. The Chapter introduces three more algorithms, *TopoSense*, *TopoRoute*, and *TopoBuild*, to accomplish the config automation. *TopoBuild* replicates the data plane topology to the controller, *TopoSense* observes any changes and *TopoRoute* uses these trio to compute routes on demand. The chapter puts a use-case architecture called *SDN-SIM* that demonstrates the work of self-configuration through cross-platform integration of SDN with a system-level network simulator. Additionally, it presents a platform-agnostic monitoring and configuration API called *ShellMon*.

Chapter 5 covers the final topic of contribution, which is compliance with Self-Healing. Currently, most of the existing works in the literature related to Self-Healing focus primarily on cellular health optimization. For this reason, we propose an alternative approach through service migration, especially after observing the tremendous application of cloud-native and micro-service technologies in 5G Softwarization. In service migration strategy, an orchestrator makes live services (mostly VNFs) migrate across a network to provide high availability through fault tolerance and load-balancing. Chapter 5 proposes an elastic Service-Oriented Architecture (eSOA) for Multi-access Edge Computing (MEC) called *Cell Breathing*. It orchestrates autonomy in migrating services among several MEC nodes based on a victim

selection algorithm called *Cell Breathing* and a target selection algorithm called *Cell Rank.* The architecture demonstrates convergence in load-balancing across the MEC deployment with three communication modes. Additionally, the chapter also presents a prototype Blockchain integration that poses prevention potential attacks such as MITM and DDoS.

6.2. Future Directions

The goal of the research has been the design and development of a 5G-SON compliant Routing framework that leverages machine learning models to compute routes using reliability as a metric. In the process of exploring several existing approaches we have come across the KDN model and the rest of the discussion of the thesis revolves around it.

Recent developments in the communication systems introduce several novel concepts such as Federated Learning, Consensus-based authentication, Trusted Execution Environment (TEE), Open RAN (ORAN) and Segment Routing. We see the potential of the existing research to enhance the modelling using the above technologies. Therefore, we aim for the following enhancement as a future extension to the existing work.

1. **TEE based secure MEC architecture:** A secure MEC architecture that uses *MRoute*, where the keys are stored on-device leveraging the TEE technology. In this case, the key attributes such as the secret keys for encryption, the salt values for the hash and the credentials will be stored within a Trusted Platform Module (TPM). Therefore, data intercepted outside the intended devices would be impractical for any malicious exploitation.

2. **DRL and ORAN compliance:** OpenRAN [264] consortium has proposed the ORAN architecture that aims for a vendor-agnostic intelligent RAN. Deploying *MRoute* over the Ran Intelligent Controller (RIC) could not only leverage Routing-as-a-Service but also the

native features of Next-Generation SDN (ng-SDN) such as transport independence, architecture independence and Zero-Touch Provisioning (ZTP).

3. **DLT and Segment Routing:** The Segment Routing (SR) technology has gained significant propulsion after SD-WAN has become the mainstream architecture for large scale enterprises and Datacenters. The trend shows a prompt possibility of SR replacing MPLS based ISP services. However, there has been no evidence found at the time of writing this thesis, where SR uses cognitive routing for KDN or DLT based protection against MITM and DDoS. On the contrary, research shows that MITM and DDoS are the two major types of attacks, that target SDN infrastructures. This gap has resulted in a contemporary idea called “Security by Design”, where the architecture of the infrastructure provides preventive measures against the said attack vectors. There is a prompt potential in the *Cell Breathing* architecture to mature its capability by introducing config automation for Smart-Contracts. Thus, we aim to extend our work on Self-Healing technologies by securing them using a configurable smart contract and introducing an intelligent Segment-Routing framework for transport.

6.3. Concluding remarks

Research is a process in continuity and is subject to evolution. This thesis documents the investigation of achieving self-organization in a Knowledge-Defined network subject to the light of cognitive routing. The development, design and evaluation involved in this investigation leverage the state of the art of the respective concepts present in the literature. I as the author of this thesis would like to conclude this document with a deep appreciation towards the reader for investing his/her time in it. Nevertheless, I shall keep pursuing my research covering the future directions mentioned above and keep contributing the knowledge.

Bibliography

[1] S. Ghosh, T. Dagiuklas, and M. Iqbal, “Energy-Aware IP Routing Over SDN,” in *2018 IEEE Global Communications Conference (GLOBECOM)*, Dec. 2018, pp. 1–7, doi: 10.1109/GLOCOM.2018.8647764.

[2] E. E. UGWUANYI, S. GHOSH, M. IQBAL, and T. DAGIUKLAS, “Reliable Resource Provisioning Using Bankers’ Deadlock Avoidance Algorithm in MEC for Industrial IoT,” *Ceskoslov. Gynekol.*, vol. 21, no. 6, pp. 422–423, 1956, doi: 10.1109/ACCESS.2018.2857726.

[3] E. E. Ugwuanyi, S. Ghosh, M. Iqbal, T. Dagiuklas, S. Mumtaz, and A. Al-Dulaimi, “Co-Operative and Hybrid Replacement Caching for Multi-Access Mobile Edge Computing,” in *2019 European Conference on Networks and Communications (EuCNC)*, Jun. 2019, pp. 394–399, doi: 10.1109/EuCNC.2019.8801991.

[4] S. Ghosh, E. Ugwuanyi, A. Dagiuklas, and M. Iqbal, “BlueArch--An implementation of 5G Testbed,” *J. Commun.*, 2019.

[5] S. Ghosh *et al.*, “SDN-Sim: Integrating a System-Level Simulator with a Software Defined Network,” *IEEE Commun. Stand. Mag.*, vol. 4, no. 1, pp. 18–25, 2020, doi: 10.1109/MCOMSTD.001.1900035.

[6] S. Ghosh, M. Iqbal, and T. Dagiuklas, “A centralized hybrid routing model for multicontroller SD-WANs,” *Trans. Emerg. Telecommun. Technol.*, vol. 32, no. 6, p. e4252, 2021.

[7] S. Ghosh, B. El Boudani, A. Dagiuklas, and M. Iqbal, “A Self-Organised Knowledge Defined Networks Architecture for Reliable Routing,” in *4th International Conference on Information Science and Systems ICISS 2021*, 2021.

[8] M. A. Khan *et al.*, “Robust, Resilient and Reliable Architecture for V2X Communications,” *IEEE Trans. Intell. Transp. Syst.*, 2021.

[9] R. Mijumbi, J. Serrat, J. Gorricho, S. Latre, M. Charalambides, and D. Lopez, “Management and orchestration challenges in network functions virtualization,” *IEEE Commun. Mag.*, vol. 54, no. 1, pp. 98–105, Jan. 2016, doi: 10.1109/MCOM.2016.7378433.

[10] Li Ling, Ma Xiaozhen, and Huang Yulan, “CDN cloud: A novel scheme for combining CDN and cloud computing,” in *Proceedings of 2013 2nd International Conference on Measurement, Information and Control*, Aug. 2013, vol. 01, pp. 687–690, doi: 10.1109/MIC.2013.6758055.

[11] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, “On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration,” *IEEE Commun. Surv. Tutorials*, vol. 19, no. 3, pp. 1657–1681, 2017, doi: 10.1109/COMST.2017.2705720.

[12] B. Carbunar, M. Pearce, V. Vasudevan, and M. Needham, “Predictive Caching for Video on Demand CDNs,” in *2011 IEEE Global Telecommunications Conference - GLOBECOM 2011*, Dec. 2011, pp. 1–5, doi: 10.1109/GLOCOM.2011.6133574.

[13] A. S. Thyagaturu, A. Mercian, M. P. McGarry, M. Reisslein, and W. Kellerer, “Software Defined Optical Networks (SDONs): A Comprehensive Survey,” *IEEE Commun. Surv. Tutorials*, vol. 18, no. 4, pp. 2738–2786, 2016, doi: 10.1109/COMST.2016.2586999.

[14] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, “Network Function Virtualization: State-of-the-Art and Research Challenges,” *IEEE Commun. Surv. Tutorials*, vol. 18, no. 1, pp. 236–262, 2016, doi: 10.1109/COMST.2015.2477041.

[15] I. Sarrigiannis, K. Ramantas, E. Kartsakli, P. Mekikis, A. Antonopoulos, and C. Verikoukis, “Online VNF Lifecycle Management in a MEC-enabled 5G IoT Architecture,” *IEEE Internet Things J.*, p. 1, 2019, doi: 10.1109/JIOT.2019.2944695.

[16] “OpenFlow-enabled SDN and Network Functions Virtualization.” Feb. 2014.

[17] B. Butler, “Business Value of Cisco SD-WAN Solutions : Studying the Results of Deployed Organizations,” pp. 1–13, 2019.

[18] T. Fojta, “VMware vCloud ® Architecture Toolkit™ for Service Providers Architecting a VMware vCloud Director ® Solution for VMware Cloud Providers™,” no. January, 2018, Accessed: Jun. 10, 2020. [Online]. Available: www.vmware.com.

[19] *Cisco SD-WAN Getting Started Guide*. Cisco Systems, 2019.

[20] Citrix Product Documentation, *Citrix SD-WAN 10.2*. .

[21] I. Cisco Systems, “Unicast Overlay Routing Overview.”

[22] Citrix Product Documentation, “Adaptive transport,” 2017, Accessed: Jun. 10, 2020. [Online]. Available: https://docs.citrix.com/en-us/citrix-virtual-apps-desktops/technical-overview/hdx/adaptive-transport.html.

[23] J. Moy, “OSPF Version 2,” no. 2328. RFC Editor, Apr. 1998, doi: 10.17487/RFC2328.

[24] D. Savage, J. Ng, S. Moore, D. Slice, P. Paluch, and R. White, “Cisco’s Enhanced Interior Gateway Routing Protocol (EIGRP),” no. 7868. RFC Editor, May 2016, doi: 10.17487/RFC7868.

[25] A. Jain, Sadagopan N S, S. K. Lohani, and M. Vutukuru, “A comparison of SDN and

NFV for re-designing the LTE Packet Core," in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov. 2016, pp. 74–80, doi: 10.1109/NFV-SDN.2016.7919479.

[26] R. Nikbazm, M. Dashtbani, and M. Ahmadi, "Enabling SDN on a special deployment of OpenStack," in *2015 5th International Conference on Computer and Knowledge Engineering (ICCKE)*, Oct. 2015, pp. 337–342, doi: 10.1109/ICCKE.2015.7365852.

[27] H. Zhang and J. Yan, "Performance of SDN Routing in Comparison with Legacy Routing Protocols," in *2015 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, 2015, pp. 491–494, doi: 10.1109/CyberC.2015.30.

[28] T. Zhang, A. Bianco, and P. Giaccone, "The role of inter-controller traffic in SDN controllers placement," in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov. 2016, pp. 87–92, doi: 10.1109/NFV-SDN.2016.7919481.

[29] L. Zhao *et al.*, "Vehicular Communications: Standardization and Open Issues," *IEEE Commun. Stand. Mag.*, vol. 2, no. 4, pp. 74–80, Dec. 2018, doi: 10.1109/MCOMSTD.2018.1800027.

[30] S. A. Busari, M. A. Khan, K. M. S. Huq, S. Mumtaz, and J. Rodriguez, "Millimetre-wave massive MIMO for cellular vehicle-to-infrastructure communication," *IET Intell. Transp. Syst.*, vol. 13, no. 6, pp. 983–990, Jun. 2019, doi: 10.1049/iet-its.2018.5492.

[31] S. A. Busari *et al.*, "Generalized Hybrid Beamforming for Vehicular Connectivity Using THz Massive MIMO," *IEEE Trans. Veh. Technol.*, vol. 68, no. 9, pp. 8372–8383, 2019, doi: 10.1109/TVT.2019.2921563.

[32] M. K. Müller *et al.*, "Flexible multi-node simulation of cellular mobile communications: the Vienna 5G System Level Simulator," *EURASIP J. Wirel. Commun. Netw.*, vol. 2018, no. 1, p. 227, Sep. 2018.

[33] P. Alvarez, C. Galiotto, J. van de Belt, D. Finn, H. Ahmadi, and L. DaSilva, "Simulating dense small cell networks," in *2016 IEEE Wireless Communications and Networking Conference*, Apr. 2016, pp. 1–6.

[34] D. Kreutz, F. M. V Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[35] E. Cuervo *et al.*, "Maui: making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, 2010, pp. 49–62.

[36] R. Alexander *et al.*, “RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks,” no. 6550. RFC Editor, Mar. 2012, doi: 10.17487/RFC6550.

[37] K. Kowalik, B. Keegan, and M. Davis, “Rare-resource aware routing for mesh,” in *2007 IEEE International Conference on Communications*, 2007, pp. 4931–4936.

[38] V. C. Gungor, C. Sastry, Z. Song, and R. Integlia, “Resource-aware and link quality based routing metric for wireless sensor and actor networks,” in *2007 IEEE International Conference on Communications*, 2007, pp. 3364–3369.

[39] E. Haleplidis, K. Pentikousis, S. Denazis, J. H. Salim, D. Meyer, and O. Koufopavlou, “Software-Defined Networking (SDN): Layers and Architecture Terminology,” no. 7426. RFC Editor, Jan. 2015, doi: 10.17487/RFC7426.

[40] W.-L. Jin, “A link queue model of network traffic flow,” *arXiv Prepr. arXiv1209.2361*, 2012.

[41] M. Fidler and A. Rizk, “A guide to the stochastic network calculus,” *IEEE Commun. Surv. \& Tutorials*, vol. 17, no. 1, pp. 92–105, 2014.

[42] K. Nisar *et al.*, “A survey on the architecture, application, and security of software defined networking: Challenges and open issues,” *Internet of Things*, vol. 12, p. 100289, 2020, doi: https://doi.org/10.1016/j.iot.2020.100289.

[43] 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 Communications Surveys and Tutorials*. 2017, doi: 10.1109/COMST.2017.2727878.

[44] D. D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski, “A Knowledge Plane for the Internet,” in *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2003, pp. 3–10, doi: 10.1145/863955.863957.

[45] 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 Sens. J.*, vol. 16, no. 17, pp. 6782–6792, 2016.

[46] O. G. Aliu, A. Imran, M. A. Imran, and B. Evans, “A survey of self organisation in future cellular networks,” *IEEE Communications Surveys and Tutorials*, vol. 15, no. 1. pp. 336–361, 2013, doi: 10.1109/SURV.2012.021312.00116.

[47] M. A. M. Albreem, “5G wireless communication systems: Vision and challenges,” in *2015 International Conference on Computer, Communications, and Control Technology (I4CT)*, 2015, pp. 493–497.

[48] J. G. Andrews *et al.*, “What will 5G be?,” *IEEE J. Sel. areas Commun.*, vol. 32, no. 6, pp. 1065–1082, 2014.

[49] P. Wainio and K. Seppänen, “Self-optimizing last-mile backhaul network for 5G small cells,” in *2016 IEEE International Conference on Communications Workshops (ICC)*, 2016, pp. 232–239.

[50] J. Xie *et al.*, “A Survey of Machine Learning Techniques Applied to Software Defined Networking (SDN): Research Issues and Challenges,” *IEEE Commun. Surv. Tutorials*, vol. 21, no. 1, pp. 393–430, 2019.

[51] W. T. Zaumen and J. J. Garcia-Luna-Aceves, “Loop-free multipath routing using generalized diffusing computations,” in *Proceedings. IEEE INFOCOM ’98, the Conference on Computer Communications. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Gateway to the 21st Century (Cat. No.98*, 1998, vol. 3, pp. 1408–1417 vol.3.

[52] E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numer. Math.*, vol. 1, no. 1, pp. 269–271, Dec. 1959, doi: 10.1007/BF01386390.

[53] L. Davoli, L. Veltri, P. L. Ventre, G. Siracusano, and S. Salsano, “Traffic Engineering with Segment Routing: SDN-Based Architectural Design and Open Source Implementation,” in *2015 Fourth European Workshop on Software Defined Networks*, 2015, pp. 111–112, doi: 10.1109/EWSDN.2015.73.

[54] I. Bueno, J. I. Aznar, E. Escalona, J. Ferrer, and J. A. Garcia-Espin, “An OpenNaaS Based SDN Framework for Dynamic QoS Control,” in *2013 IEEE SDN for Future Networks and Services (SDN4FNS)*, 2013, pp. 1–7, doi: 10.1109/SDN4FNS.2013.6702533.

[55] C. E. Rothenberg, M. R. Nascimento, M. R. Salvador, C. N. A. Corrêa, S. de Lucena, and R. Raszuk, “Revisiting Routing Control Platforms with the Eyes and Muscles of Software-Defined Networking,” in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, 2012, pp. 13–18, doi: 10.1145/2342441.2342445.

[56] J. W. Guck, M. Reisslein, and W. Kellerer, “Function Split Between Delay-Constrained Routing and Resource Allocation for Centrally Managed QoS in Industrial Networks,” *IEEE Trans. Ind. Informatics*, vol. 12, no. 6, pp. 2050–2061, 2016, doi: 10.1109/TII.2016.2592481.

[57] J. W. Guck, A. Van Bemten, M. Reisslein, and W. Kellerer, “Unicast QoS Routing Algorithms for SDN: A Comprehensive Survey and Performance Evaluation,” *IEEE Commun. Surv. Tutorials*, vol. 20, no. 1, pp. 388–418, 2018, doi: 10.1109/COMST.2017.2749760.

[58] P. E. Hart, N. J. Nilsson, and B. Raphael, “A Formal Basis for the Heuristic Determination of Minimum Cost Paths,” *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, 1968, doi: 10.1109/TSSC.1968.300136.

[59] E. I. Chong, “On finding single source single destination k shortest paths,” in *Proc. International Conference on Computing and Information, July 1995*, 1995, pp. 40–47.

[60] G. Liu and K. G. Ramakrishnan, “A* Prune: an algorithm for finding K shortest paths subject to multiple constraints,” in *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No. 01CH37213)*, 2001, vol. 2, pp. 743–749.

[61] W. C. Lee, M. G. Hluchyi, and P. A. Humblet, “Routing subject to quality of service constraints in integrated communication networks,” *IEEE Netw.*, vol. 9, no. 4, pp. 46–55, 1995.

[62] R. BELLMAN, “ON A ROUTING PROBLEM,” *Q. Appl. Math.*, vol. 16, no. 1, pp. 87–90, 1958, [Online]. Available: http://www.jstor.org/stable/43634538.

[63] J. Y. Yen, “Finding the K Shortest Loopless Paths in a Network,” *Manage. Sci.*, vol. 17, no. 11, pp. 712–716, 1971, [Online]. Available: http://www.jstor.org/stable/2629312.

[64] R. Widyono and others, *The design and evaluation of routing algorithms for real-time channels*. Citeseer, 1994.

[65] Z. Jia and P. Varaiya, “Heuristic methods for delay-constrained leastcost routing problem using k-shortest-path algorithms,” in *Proc. IEEE INFOCOM*, 2001, pp. 1–9.

[66] D. Blokh and G. Gutin, “An approximate algorithm for combinatorial optimization problems with two parameters,” *Australas. J. Comb.*, vol. 14, pp. 157–164, 1996.

[67] A. Juttner, B. Szviatovski, I. Mécs, and Z. Rajkó, “Lagrange relaxation based method for the QoS routing problem,” in *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No. 01CH37213)*, 2001, vol. 2, pp. 859–868.

[68] G. Y. Handler and I. Zang, “A dual algorithm for the constrained shortest path problem,” *Networks*, vol. 10, no. 4, pp. 293–309, 1980, doi: https://doi.org/10.1002/net.3230100403.

[69] L. Santos, J. Coutinho-Rodrigues, and J. R. Current, “An improved solution algorithm for the constrained shortest path problem,” *Transp. Res. Part B Methodol.*, vol. 41, no. 7, pp. 756–771, 2007.

[70] G. Feng, C. Douligeris, K. Makki, and N. Pissinou, “Performance evaluation of delay-constrained least-cost QoS routing algorithms based on linear and nonlinear lagrange relaxation,” in *2002 IEEE International Conference on Communications. Conference Proceedings. ICC 2002 (Cat. No. 02CH37333)*, 2002, vol. 4, pp. 2273–2278.

[71] L. Guo and I. Matta, “Search space reduction in QoS routing,” *Comput. Networks*, vol. 41, no. 1, pp. 73–88, 2003.

[72] T. Korkmaz and M. Krunz, “Multi-constrained optimal path selection,” in *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No. 01CH37213)*, 2001, vol. 2, pp. 834–843.

[73] G. Feng, K. Makki, N. Pissinou, and C. Douligeris, “Heuristic and exact algorithms for QoS routing with multiple constraints,” *IEICE Trans. Commun.*, vol. 85, no. 12, pp. 2838–2850, 2002.

[74] D. S. Reeves and H. F. Salama, “A distributed algorithm for delay-constrained unicast routing,” *IEEE/ACM Trans. Netw.*, vol. 8, no. 2, pp. 239–250, 2000.

[75] H. F. Salama, D. S. Reeves, and Y. Viniotis, “A distributed algorithm for delay-constrained unicast routing,” in *Proceedings of INFOCOM’97*, 1997, vol. 1, pp. 84–91.

[76] Q. Sun and H. Langendörfer, “A new distributed routing algorithm for supporting delay-sensitive applications,” *Comput. Commun.*, vol. 21, no. 6, pp. 572–578, 1998.

[77] K. Ishida, K. Amano, and N. Kannari, “A delay-constrained least-cost path routing protocol and the synthesis method,” in *Proceedings Fifth International Conference on Real-Time Computing Systems and Applications (Cat. No. 98EX236)*, 1998, pp. 58–65.

[78] R. Sriram, G. Manimaran, and C. S. R. Murthy, “Preferred link based delay-constrained least-cost routing in wide area networks,” *Comput. Commun.*, vol. 21, no. 18, pp. 1655–1669, 1998.

[79] W. Liu, W. Lou, and Y. Fang, “An efficient quality of service routing algorithm for delay-sensitive applications,” *Comput. Networks*, vol. 47, no. 1, pp. 87–104, 2005.

[80] J. W. Guck, A. Van Bemten, M. Reisslein, and W. Kellerer, “Unicast QoS Routing Algorithms for SDN: A Comprehensive Survey and Performance Evaluation,” *IEEE Commun. Surv. Tutorials*, vol. 20, no. 1, pp. 388–415, 2018.

[81] R. Amin, M. Reisslein, and N. Shah, “Hybrid SDN Networks: A Survey of Existing Approaches,” *IEEE Commun. Surv. Tutorials*, vol. 20, no. 4, pp. 3259–3306, 2018, doi: 10.1109/COMST.2018.2837161.

[82] R. Amin, M. Reisslein, and N. Shah, “Hybrid SDN networks: A survey of existing approaches,” *IEEE Communications Surveys and Tutorials*, vol. 20, no. 4. pp. 3259–3306, 2018, doi: 10.1109/COMST.2018.2837161.

[83] M. Canini, A. Feldmann, D. Levin, F. Schaffert, and S. Schmid, “Software-defined networks: Incremental deployment with panopticon,” *Computer (Long. Beach. Calif).*, vol. 47, no. 11, pp. 56–60, 2014.

[84] R. Cardona, “Introduction to Spine-and-Leaf Topologies,” in *The Fast-Track Guide to VXLAN BGP EVPN Fabrics*, Springer, 2021, pp. 1–26.

[85] K. Poularakis, G. Iosifidis, G. Smaragdakis, and L. Tassiulas, “One step at a time: Optimizing SDN upgrades in ISP networks,” in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, 2017, pp. 1–9.

[86] H. Xu, J. Fan, J. Wu, C. Qiao, and L. Huang, “Joint deployment and routing in hybrid SDNs,” in *2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*, 2017, pp. 1–10.

[87] M. Caria and A. Jukan, “The perfect match: Optical bypass and SDN partitioning,” in *2015 IEEE 16th International Conference on High Performance Switching and Routing (HPSR)*, 2015, pp. 1–6.

[88] J. Nunez-Martinez, J. Baranda, and J. Mangues-Bafalluy, “A service-based model for the hybrid software defined wireless mesh backhaul of small cells,” in *2015 11th International Conference on Network and Service Management (CNSM)*, 2015, pp. 390–393.

[89] L. He, X. Zhang, Z. Cheng, and Y. Jiang, “Design and implementation of SDN/IP hybrid space information network prototype,” in *2016 IEEE/CIC International Conference on Communications in China (ICCC Workshops)*, 2016, pp. 1–6.

[90] D. J. Casey and B. E. Mullins, “SDN shim: Controlling legacy devices,” in *2015 IEEE 40th Conference on Local Computer Networks (LCN)*, 2015, pp. 169–172.

[91] T. Das, M. Caria, A. Jukan, and M. Hoffmann, “Insights on SDN migration trajectory,” in *2015 IEEE International Conference on Communications (ICC)*, 2015, pp. 5348–5353.

[92] T. Lukovszki, M. Rost, and S. Schmid, “It’s a match! near-optimal and incremental middlebox deployment,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 46, no. 1, pp. 30–36, 2016.

[93] M. Caria, T. Das, A. Jukan, and M. Hoffmann, “Divide and conquer: Partitioning OSPF networks with SDN,” in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2015, pp. 467–474.

[94] M. Caria, A. Jukan, and M. Hoffmann, “SDN partitioning: A centralized control plane for distributed routing protocols,” *IEEE Trans. Netw. Serv. Manag.*, vol. 13, no. 3, pp. 381–393, 2016.

[95] D. K. Hong, Y. Ma, S. Banerjee, and Z. M. Mao, “Incremental deployment of SDN in hybrid enterprise and ISP networks,” in *Proceedings of the Symposium on SDN Research*, 2016, pp. 1–7.

[96] H. Xu, X.-Y. Li, L. Huang, H. Deng, H. Huang, and H. Wang, “Incremental deployment and throughput maximization routing for a hybrid SDN,” *IEEE/ACM Trans. Netw.*, vol.

25, no. 3, pp. 1861–1875, 2017.

[97] H. Lu, N. Arora, H. Zhang, C. Lumezanu, J. Rhee, and G. Jiang, “Hybnet: Network manager for a hybrid network infrastructure,” in *Proceedings of the Industrial Track of the 13th ACM/IFIP/USENIX International Middleware Conference*, 2013, pp. 1–6.

[98] V. D. Chemalamarri, P. Nanda, and K. F. Navarro, “SYMPHONY-A controller architecture for hybrid software defined networks,” in *2015 Fourth European Workshop on Software Defined Networks*, 2015, pp. 55–60.

[99] M. Markovitch and S. Schmid, “SHEAR: A highly available and flexible network architecture marrying distributed and logically centralized control planes,” in *2015 IEEE 23rd International Conference on Network Protocols (ICNP)*, 2015, pp. 78–89.

[100] J. Stringer *et al.*, “Cardigan: SDN distributed routing fabric going live at an Internet exchange,” in *2014 IEEE Symposium on Computers and Communications (ISCC)*, 2014, pp. 1–7.

[101] R. Hand and E. Keller, “Closedflow: Openflow-like control over proprietary devices,” in *Proceedings of the third workshop on Hot topics in software defined networking*, 2014, pp. 7–12.

[102] T. Nelson, A. D. Ferguson, D. Yu, R. Fonseca, and S. Krishnamurthi, “Exodus: Toward automatic migration of enterprise network configurations to SDNs,” in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, 2015, pp. 1–7.

[103] F. Farias, J. Salvatti, P. Victor, and A. Abelem, “Integrating legacy forwarding environment to OpenFlow/SDN control plane,” in *2013 15th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 2013, pp. 1–3.

[104] M. A. S. Santos, B. T. De Oliveira, C. B. Margi, B. A. A. Nunes, T. Turletti, and K. Obraczka, “Software-defined networking based capacity sharing in hybrid networks,” in *2013 21st IEEE international conference on network protocols (ICNP)*, 2013, pp. 1–6.

[105] T. Choi, S. Kang, S. Yoon, S. Yang, S. Song, and H. Park, “SuVMF: Software-defined unified virtual monitoring function for SDN-based large-scale networks,” in *Proceedings of The Ninth International Conference on Future Internet Technologies*, 2014, pp. 1–6.

[106] A. N. Katov, A. Mihovska, and N. R. Prasad, “Hybrid SDN architecture for resource consolidation in MPLS networks,” in *2015 Wireless Telecommunications Symposium (WTS)*, 2015, pp. 1–8.

[107] C. Sieber *et al.*, “Network configuration with quality of service abstractions for SDN

and legacy networks," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2015, pp. 1135–1136.

[108] C. Sieber, A. Blenk, A. Basta, D. Hock, and W. Kellerer, "Towards a programmable management plane for SDN and legacy networks," in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, 2016, pp. 319–327.

[109] R. Katiyar, P. Pawar, A. Gupta, and K. Kataoka, "Auto-configuration of sdn switches in sdn/non-sdn hybrid network," in *Proceedings of the Asian Internet Engineering Conference*, 2015, pp. 48–53.

[110] A. Martinez, M. Yannuzzi, J. E. L. de Vergara, R. Serral-Gracià, and W. Ram\'\irez, "An ontology-based information extraction system for bridging the configuration gap in hybrid sdn environments," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2015, pp. 441–449.

[111] A. Mishra, D. Bansod, and K. Haribabu, "A Framework for OpenFlow-like Policy-based Routing in Hybrid Software Defined Networks.," in *INC*, 2016, pp. 97–102.

[112] W. Fan, X. Wang, and Y. Wu, "Incremental graph pattern matching," *ACM Trans. Database Syst.*, vol. 38, no. 3, pp. 1–47, 2013.

[113] R. Amin, N. Shah, B. Shah, and O. Alfandi, "Auto-configuration of ACL policy in case of topology change in hybrid SDN," *IEEE Access*, vol. 4, pp. 9437–9450, 2016.

[114] C. Sieber, R. Durner, and W. Kellerer, "How fast can you reconfigure your partially deployed SDN network?," in *2017 IFIP Networking Conference (IFIP Networking) and Workshops*, 2017, pp. 1–9.

[115] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, and Y. Liu, "A Survey of Machine Learning Techniques Applied to Software Defined Networking (SDN): Research Issues and Challenges," *IEEE Communications Surveys and Tutorials*, 2018.

[116] F. Tang, B. Mao, Y. Kawamoto, and N. Kato, "Survey on Machine Learning for Intelligent End-to-End Communication Toward 6G: From Network Access, Routing to Traffic Control and Streaming Adaption," *IEEE Commun. Surv. Tutorials*, vol. 23, no. 3, pp. 1578–1598, 2021, doi: 10.1109/COMST.2021.3073009.

[117] R. Amin, E. Rojas, A. Aqdus, S. Ramzan, D. Casillas-Perez, and J. M. Arco, "A Survey on Machine Learning Techniques for Routing Optimization in SDN," *IEEE Access*, vol. 9, pp. 104582–104611, 2021, doi: 10.1109/ACCESS.2021.3099092.

[118] M. Lin and Y. Zhao, "Artificial intelligence-empowered resource management for future wireless communications: A survey," *China Commun.*, vol. 17, no. 3, pp. 58–77, 2020, doi: 10.23919/JCC.2020.03.006.

[119] F. O. Olowononi, D. B. Rawat, and C. Liu, "Resilient Machine Learning for Networked

Cyber Physical Systems: A Survey for Machine Learning Security to Securing Machine Learning for CPS,” *IEEE Commun. Surv. Tutorials*, vol. 23, no. 1, pp. 524–552, 2021, doi: 10.1109/COMST.2020.3036778.

[120] L. Yanjun, L. Xiaobo, and Y. Osamu, “Traffic engineering framework with machine learning based meta-layer in software-defined networks,” in *2014 4th IEEE International Conference on Network Infrastructure and Digital Content*, 2014, pp. 121–125.

[121] A. Azzouni, R. Boutaba, and G. Pujolle, “NeuRoute: Predictive dynamic routing for software-defined networks,” in *2017 13th International Conference on Network and Service Management, CNSM 2017*, 2018, vol. 2018-Janua, pp. 1–6, doi: 10.23919/CNSM.2017.8256059.

[122] S. Sendra, A. Rego, J. Lloret, J. M. Jimenez, and O. Romero, “Including artificial intelligence in a routing protocol using Software Defined Networks,” in *2017 IEEE International Conference on Communications Workshops, ICC Workshops 2017*, 2017, pp. 670–674, doi: 10.1109/ICCW.2017.7962735.

[123] F. Francois and E. Gelenbe, “Optimizing Secure SDN-Enabled Inter-Data Centre Overlay Networks through Cognitive Routing,” in *2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2016, pp. 283–288.

[124] F. Francois and E. Gelenbe, “Towards a cognitive routing engine for software defined networks,” in *2016 IEEE International Conference on Communications (ICC)*, 2016, pp. 1–6.

[125] 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 *2016 IEEE International Conference on Services Computing (SCC)*, 2016, pp. 25–33.

[126] G. Stampa, M. Arias, D. Sanchez-Charles, V. Muntes-Mulero, and A. Cabellos, “A Deep-Reinforcement Learning Approach for Software-Defined Networking Routing Optimization.” 2017.

[127] Á. López-Raventós, F. Wilhelmi, S. Barrachina-Muñoz, and B. Bellalta, “Machine learning and software defined networks for high-density wlans,” *arXiv preprint arXiv:1804.05534*, vol. 16. Apr, 2018.

[128] R. Alvizu, S. Troia, G. Maier, and A. Pattavina, “Matheuristic with machine-learning-based prediction for software-defined mobile metro-core networks,” *J. Opt. Commun. Netw.*, vol. 9, no. 9, pp. D19--D30, 2017.

[129] C. Chen-Xiao and X. Ya-Bin, “Research on load balance method in SDN,” *Int. J. Grid Distrib. Comput.*, vol. 9, no. 1, pp. 25–36, 2016.

[130] A. Azzouni and G. Pujolle, “NeuTM: A neural network-based framework for traffic matrix prediction in SDN,” in *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*, 2018, pp. 1–5.

[131] J. Buvat and S. Basu, “Quest for margins: Operational cost strategies for mobile operators in europe,” *Capgemini*, vol. 42, 2009.

[132] P. Donegan, “Mobile Network Outages \& Service Degradations: A Heavy Reading Survey Analysis,” *Firmenschrift. Heavy Read.*, 2013.

[133] L. Jorguseski, A. Pais, F. Gunnarsson, A. Centonza, and C. Willcock, “Self-organizing networks in 3GPP: standardization and future trends,” *IEEE Commun. Mag.*, vol. 52, no. 12, pp. 28–34, 2014.

[134] Z. Zhang, K. Long, and J. Wang, “Self-organization paradigms and optimization approaches for cognitive radio technologies: a survey,” *IEEE Wirel. Commun.*, vol. 20, no. 2, pp. 36–42, 2013.

[135] R. Barco, P. Lazaro, and P. Munoz, “A unified framework for self-healing in wireless networks,” *IEEE Commun. Mag.*, vol. 50, no. 12, pp. 134–142, 2012.

[136] B. Tian, S. Han, S. Parvin, J. Hu, and S. Das, “Self-Healing Key Distribution Schemes for Wireless Networks: A Survey,” *Comput. J.*, vol. 54, no. 4, pp. 549–569, 2011, doi: 10.1093/comjnl/bxr022.

[137] S.-I. Yang, D. M. Frangopol, and L. C. Neves, “Service life prediction of structural systems using lifetime functions with emphasis on bridges,” *Reliab. Eng. \& Syst. Saf.*, vol. 86, no. 1, pp. 39–51, 2004.

[138] A. Imran, A. Zoha, and A. Abu-Dayya, “Challenges in 5G: how to empower SON with big data for enabling 5G,” *IEEE Netw.*, vol. 28, no. 6, pp. 27–33, 2014.

[139] M. M. S. Marwangi *et al.*, “Challenges and practical implementation of self-organizing networks in LTE/LTE-Advanced systems,” in *ICIMU 2011: Proceedings of the 5th international Conference on Information Technology \& Multimedia*, 2011, pp. 1–5.

[140] P. Stuckmann *et al.*, “The EUREKA Gandalf project: monitoring and self-tuning techniques for heterogeneous radio access networks,” in *2005 IEEE 61st Vehicular Technology Conference*, 2005, vol. 4, pp. 2570–2574.

[141] L. C. Schmelz *et al.*, “Self-organisation in wireless networks use cases and their interrelation,” in *Wireless World Res. Forum Meeting*, 2009, vol. 22, pp. 1–5.

[142] www.qson.org, “Quality of Service Aware Energy Efficient Self Organizing Future Cellular Networks.”

[143] R. Litjens *et al.*, “Self-management for unified heterogeneous radio access networks,”

in *2013 IEEE 77th Vehicular Technology Conference (VTC Spring)*, 2013, pp. 1–5.

[144] K. Ha *et al.*, “Adaptive VM handoff across cloudlets,” *Tech. Rep. C.*, 2015.

[145] A. Machen, S. Wang, K. K. Leung, B. J. Ko, and T. Salonidis, “Migrating running applications across mobile edge clouds: poster,” in *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*, 2016, pp. 435–436.

[146] K. Ha, P. Pillai, W. Richter, Y. Abe, and M. Satyanarayanan, “Just-in-time provisioning for cyber foraging,” in *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, 2013, pp. 153–166.

[147] S. Wang, J. Xu, N. Zhang, and Y. Liu, “A Survey on Service Migration in Mobile Edge Computing,” *IEEE Access*, vol. 6, pp. 23511–23528, 2018, doi: 10.1109/ACCESS.2018.2828102.

[148] T. Taleb, A. Ksentini, and P. A. Frangoudis, “Follow-me cloud: When cloud services follow mobile users,” *IEEE Trans. Cloud Comput.*, vol. 7, no. 2, pp. 369–382, 2016.

[149] T. Taleb and A. Ksentini, “An analytical model for follow me cloud,” in *2013 IEEE Global Communications Conference (GLOBECOM)*, 2013, pp. 1291–1296.

[150] T. Taleb and A. Ksentini, “Follow me cloud: interworking federated clouds and distributed mobile networks,” *IEEE Netw.*, vol. 27, no. 5, pp. 12–19, 2013.

[151] A. Ksentini, T. Taleb, and M. Chen, “A Markov decision process-based service migration procedure for follow me cloud,” in *2014 IEEE International Conference on Communications (ICC)*, 2014, pp. 1350–1354.

[152] S. Wang, R. Urgaonkar, T. He, M. Zafer, K. Chan, and K. K. Leung, “Mobility-induced service migration in mobile micro-clouds,” in *2014 IEEE military communications conference*, 2014, pp. 835–840.

[153] K.-H. Chiang and N. Shenoy, “A 2-D random-walk mobility model for location-management studies in wireless networks,” *IEEE Trans. Veh. Technol.*, vol. 53, no. 2, pp. 413–424, 2004.

[154] S. Wang, R. Urgaonkar, T. He, K. Chan, M. Zafer, and K. K. Leung, “Dynamic service placement for mobile micro-clouds with predicted future costs,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 4, pp. 1002–1016, 2016.

[155] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, “Dynamic service migration in mobile edge-clouds,” in *2015 IFIP Networking Conference (IFIP Networking)*, 2015, pp. 1–9.

[156] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, “Network slicing in 5G: Survey and challenges,” *IEEE Commun. Mag.*, vol. 55, no. 5, pp. 94–100, 2017.

[157] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, “Mobile edge computing: A survey,” *IEEE Internet Things J.*, vol. 5, no. 1, pp. 450–465, 2018.

[158] M. B. Yassein, S. Aljawarneh, M. Al-Rousan, W. Mardini, and W. Al-Rashdan, “Combined software-defined network (SDN) and Internet of Things (IoT),” in *Electrical and Computing Technologies and Applications (ICECTA), 2017 International Conference on*, 2017, pp. 1–6.

[159] Y. C. Tay, K. Gaurav, and P. Karkun, “A Performance Comparison of Containers and Virtual Machines in Workload Migration Context,” in *Distributed Computing Systems Workshops (ICDCSW), 2017 IEEE 37th International Conference on*, 2017, pp. 61–66.

[160] Y. Qiu, C.-H. Lung, S. Ajila, and P. Srivastava, “LXC container migration in cloudlets under multipath TCP,” in *Computer Software and Applications Conference (COMPSAC), 2017 IEEE 41st Annual*, 2017, vol. 2, pp. 31–36.

[161] S. Nadgowda, S. Suneja, N. Bila, and C. Isci, “Voyager: complete container state migration,” in *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*, 2017, pp. 2137–2142.

[162] C. Dupont, R. Giaffreda, and L. Capra, “Edge computing in IoT context: horizontal and vertical Linux container migration,” *Glob. Internet Things Summit (GIoTS). IEEE*, 2017.

[163] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008.

[164] P. Tasatanattakool and C. Techapanupreeda, “Blockchain: Challenges and applications,” in *Information Networking (ICOIN), 2018 International Conference on*, 2018, pp. 473–475.

[165] H. Halpin and M. Piekarska, “Introduction to Security and Privacy on the Blockchain,” in *Security and Privacy Workshops (EuroS\&PW), 2017 IEEE European Symposium on*, 2017, pp. 1–3.

[166] P.-H. Kuo, A. Mourad, and J. Ahn, “Potential Applicability of Distributed Ledger to Wireless Networking Technologies,” *IEEE Wirel. Commun.*, vol. 25, no. 4, pp. 4–6, 2018.

[167] J. Backman, S. Yrjölä, K. Valtanen, and O. Mämmelä, “Blockchain network slice broker in 5G: Slice leasing in factory of the future use case,” in *Internet of Things Business Models, Users, and Networks, 2017*, 2017, pp. 1–8.

[168] E. Di Pascale, J. McMenamy, I. Macaluso, and L. Doyle, “Smart Contract SLAs for Dense Small-Cell-as-a-Service,” *arXiv Prepr. arXiv1703.04502*, 2017.

[169] E. J. Scheid and B. Stiller, “Automatic SLA Compensation based on Smart Contracts,” 2018.

[170] R. V. Rosa and C. E. Rothenberg, “Blockchain-based Decentralized Applications for

Multiple Administrative Domain Networking," *IEEE Com. Mag*, 2018.

[171] M. Matinmikko-Blue, S. Yrjoelae, and M. Latva-aho, "Micro Operators for Ultra-Dense Network Deployment with Network Slicing and Spectrum Micro Licensing," in *2018 IEEE 87th Vehicular Technology Conference (VTC Spring)*, 2018, pp. 1–6.

[172] M. Liu, F. R. Yu, Y. Teng, V. C. M. Leung, and M. Song, "Joint computation offloading and content caching for wireless blockchain networks," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2018, pp. 517–522.

[173] M. Giordani, M. Polese, M. Mezzavilla, S. Rangan, and M. Zorzi, "Toward 6G Networks: Use Cases and Technologies," *IEEE Commun. Mag.*, vol. 58, no. 3, pp. 55–61, 2020.

[174] G. S. Malkin, "RIP Version 2," no. 2453. RFC Editor, Nov. 1998, doi: 10.17487/RFC2453.

[175] M. Al-Fares, A. Loukissas, and A. Vahdat, "A Scalable, Commodity Data Center Network Architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, Aug. 2008, doi: 10.1145/1402946.1402967.

[176] "Archives - Open Networking Foundation." https://www.opennetworking.org/ng-sdn/ (accessed Aug. 31, 2020).

[177] "Design Zone for Campus - Cisco SD-Access Solution Design Guide (CVD) - Cisco." https://www.cisco.com/c/en/us/td/docs/solutions/CVD/Campus/cisco-sda-design-guide.html (accessed Aug. 31, 2020).

[178] VMware, "VMware vSphere Documentation," *vmware.com*, 2020. https://docs.vmware.com/en/VMware-vSphere/index.html (accessed Aug. 31, 2020).

[179] 5GPPP, "View on 5G Architecture, Version 3.0, February 2020," 2020. doi: 10.5281/zenodo.3265031.

[180] "Overview — Open Network Install Environment documentation." https://opencomputeproject.github.io/onie/overview/index.html#onie-overview (accessed Aug. 31, 2020).

[181] "ONLP APIs for Applications | OpenNetworkLinux." http://opencomputeproject.github.io/OpenNetworkLinux/onlp/applications/ (accessed Aug. 31, 2020).

[182] "GitHub - opencomputeproject/SAI: Switch Abstraction Interface." https://github.com/opencomputeproject/SAI (accessed Aug. 31, 2020).

[183] "SONiC." https://azure.github.io/SONiC/ (accessed Aug. 31, 2020).

[184] Z. N. Abdullah, I. Ahmad, and I. Hussain, “Segment Routing in Software Defined Networks: A Survey,” *IEEE Communications Surveys and Tutorials*, 2018.

[185] A. Viswanathan, E. C. Rosen, and R. Callon, “Multiprotocol Label Switching Architecture,” no. 3031. RFC Editor, Jan. 2001, doi: 10.17487/RFC3031.

[186] B. Thomas, L. Andersson, and I. Minei, “LDP Specification,” no. 5036. RFC Editor, Oct. 2007, doi: 10.17487/RFC5036.

[187] D. O. Awduche, L. Berger, D.-H. Gan, T. Li, D. V. Srinivasan, and G. Swallow, “RSVP-TE: Extensions to RSVP for LSP Tunnels,” no. 3209. RFC Editor, Dec. 2001, doi: 10.17487/RFC3209.

[188] P. V. Klaine, M. A. Imran, O. Onireti, and R. D. Souza, *A Survey of Machine Learning Techniques Applied to Self-Organizing Cellular Networks*, vol. 19, no. 4. 2017, pp. 2392–2431.

[189] ORAN, *No Title*. 2020.

[190] ONF, *No Title*. .

[191] J. Strassner, M. O’Foghlu, W. Donnelly, and N. Agoulmine, “Beyond the Knowledge Plane: An Inference Plane to Support the Next Generation Internet,” in *2007 First International Global Information Infrastructure Symposium*, 2007, pp. 112–119.

[192] 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. Tutorials*, vol. 19, no. 4, pp. 2432–2455, Oct. 2017, doi: 10.1109/COMST.2017.2707140.

[193] 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. Tutorials*, vol. 21, no. 4, pp. 3039–3071, 2019.

[194] 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.

[195] A. Azzouni, R. Boutaba, and G. Pujolle, “NeuRoute: Predictive dynamic routing for software-defined networks,” in *2017 13th International Conference on Network and Service Management (CNSM)*, 2017, pp. 1–6.

[196] S. Sendra, A. Rego, J. Lloret, J. M. Jimenez, and O. Romero, “Including artificial intelligence in a routing protocol using Software Defined Networks,” in *2017 IEEE International Conference on Communications Workshops (ICC Workshops)*, 2017, pp. 670–674.

[197] A. Gosavi, “Queuing Formulas.” Department of Engineering Management and Systems

Engineering.

[198] Z. Yang, Y. Cui, B. Li, Y. Liu, and Y. Xu, “Software-Defined Wide Area Network (SD-WAN): Architecture, Advances and Opportunities,” in *2019 28th International Conference on Computer Communication and Networks (ICCCN)*, 2019, pp. 1–9, doi: 10.1109/ICCCN.2019.8847124.

[199] R. Moskowitz, D. Karrenberg, Y. Rekhter, E. Lear, and G. J. de Groot, “Address Allocation for Private Internets,” no. 1918. RFC Editor, Feb. 1996, doi: 10.17487/RFC1918.

[200] W. F. Sharpe, “The Sharpe Ratio,” *J. Portf. Manag.*, vol. 21, no. 1, pp. 49–58, 1994, doi: 10.3905/jpm.1994.409501.

[201] T. Bodnar and T. Zabolotskyy, “How risky is the optimal portfolio which maximizes the Sharpe ratio?,” *AStA Adv. Stat. Anal.*, vol. 101, no. 1, pp. 1–28, 2017.

[202] J. S. Turner and D. E. Taylor, “Diversifying the internet,” in *GLOBECOM’05. IEEE Global Telecommunications Conference, 2005.*, 2005, vol. 2, pp. 6--pp.

[203] A. T. Campbell, I. Katzela, K. Miki, and J. Vicente, “Open signaling for ATM, internet and mobile networks (OPENSIG’98),” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 29, no. 1, pp. 97–108, 1999.

[204] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden, “A survey of active network research,” *IEEE Commun. Mag.*, vol. 35, no. 1, pp. 80–86, 1997.

[205] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, “Ethane: Taking control of the enterprise,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 1–12, 2007.

[206] N. McKeown *et al.*, “OpenFlow: enabling innovation in campus networks,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[207] N. Kim and J. Kim, “Building netopen networking services over openflow-based programmable networks,” in *The International Conference on Information Networking 2011 (ICOIN2011)*, 2011, pp. 525–529.

[208] A. I. Frunz\ua, C. I. R\^\incu, and A. Jâtaru, “Remote Network Monitoring Using SDN Based Solutions,” in *2018 International Conference on Communications (COMM)*, 2018, pp. 301–304.

[209] H. Song, “Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane,” in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, 2013, pp. 127–132.

[210] J. M. Halpern *et al.*, “Forwarding and Control Element Separation (ForCES) Protocol Specification,” no. 5810. RFC Editor, Mar. 2010, doi: 10.17487/RFC5810.

[211] M. Jarschel, T. Zinner, T. Hoßfeld, P. Tran-Gia, and W. Kellerer, “Interfaces, attributes, and use cases: A compass for SDN,” *IEEE Commun. Mag.*, vol. 52, no. 6, pp. 210–217, 2014.

[212] A. Lara, A. Kolasani, and B. Ramamurthy, “Network innovation using openflow: A survey,” *IEEE Commun. Surv. \& tutorials*, vol. 16, no. 1, pp. 493–512, 2013.

[213] A. Autenrieth, J.-P. Elbers, P. Kaczmarek, and P. Kostecki, “Cloud orchestration with SDN/OpenFlow in carrier transport networks,” in *2013 15th International Conference on Transparent Optical Networks (ICTON)*, 2013, pp. 1–4.

[214] “(No Title).” https://sdwan-docs.cisco.com/@api/deki/pages/11355/pdf/Unicast%2BOverlay%2BRouting%2BOverview.pdf?stylesheet=default (accessed Jun. 10, 2020).

[215] P. H. Isolani, J. A. Wickboldt, C. B. Both, J. Rochol, and L. Z. Granville, “Interactive monitoring, visualization, and configuration of OpenFlow-based SDN,” in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2015, pp. 207–215.

[216] N. Foster *et al.*, “Frenetic: A network programming language,” *ACM Sigplan Not.*, vol. 46, no. 9, pp. 279–291, 2011.

[217] M. Reitblatt, M. Canini, A. Guha, and N. Foster, “Fattire: Declarative fault tolerance for software-defined networks,” in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, 2013, pp. 109–114.

[218] N. Gude *et al.*, “NOX: towards an operating system for networks,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, 2008.

[219]H. Nilsson, A. Courtney, and J. Peterson, “Functional reactive programming, continued,” in *Proceedings of the 2002 ACM SIGPLAN workshop on Haskell*, 2002, pp. 51–64.

[220] C. Elliott and P. Hudak, “Functional reactive animation,” in *Proceedings of the second ACM SIGPLAN international conference on Functional programming*, 1997, pp. 263–273.

[221] D. M. Jones, “Forms of language specification examples from commonly used computer languages,” 2008.

[222] M. K. Shin, K. H. Nam, M. Kang, and others, “Formal specification framework for software-defined networks (SDN),” *IETF Draft.*, 2013.

[223] A. Guha, M. Reitblatt, and N. Foster, “Formal foundations for software defined

networks,” *Open Net Summit*, 2013.

[224] M. Satpathy, R. Harrison, C. Snook, and M. Butler, “A comparative study of formal and informal specifications through an industrial case study,” 2001.

[225] M. Fowler, *Domain-specific languages*. Pearson Education, 2010.

[226] M. Strembeck and U. Zdun, “An approach for the systematic development of domain-specific languages,” *Softw. Pract. Exp.*, vol. 39, no. 15, pp. 1253–1292, 2009.

[227] A. Voellmy, H. Kim, and N. Feamster, “Procera: a language for high-level reactive network control,” in *Proceedings of the first workshop on Hot topics in software defined networks*, 2012, pp. 43–48.

[228] T. Stahl, M. Völter, and K. Czarnecki, *Model-driven software development: technology, engineering, management*. John Wiley \& Sons, Inc., 2006.

[229] A. Barišić, V. Amaral, M. Goulao, and B. Barroca, “Quality in use of domain-specific languages: a case study,” in *Proceedings of the 3rd ACM SIGPLAN Workshop on Evaluation and Usability of Programming Languages and Tools*, 2011, pp. 65–72.

[230] A. Van Deursen, P. Klint, and J. Visser, “Domain-specific languages: An annotated bibliography,” *ACM Sigplan Not.*, vol. 35, no. 6, pp. 26–36, 2000.

[231] D. S. Frankel, “Model driven architecture: applying MDA to enterprise computing. 2003,” *Google Sch. Google Sch. Digit. Libr. Digit. Libr.*

[232] F. A. Lopes, M. Santos, R. Fidalgo, and S. Fernandes, “A Software Engineering Perspective on SDN Programmability,” *IEEE Commun. Surv. Tutorials*, vol. 18, no. 2, pp. 1255–1272, 2016, doi: 10.1109/COMST.2015.2501026.

[233] T. L. Hinrichs, N. S. Gude, M. Casado, J. C. Mitchell, and S. Shenker, “Practical declarative network management,” in *Proceedings of the 1st ACM workshop on Research on enterprise networking*, 2009, pp. 1–10.

[234] A. Voellmy, A. Agarwal, and P. Hudak, “Nettle: Functional reactive programming for openflow networks,” 2010.

[235] N. P. Katta, J. Rexford, and D. Walker, “Logic programming for software-defined networks,” in *Workshop on Cross-Model Design and Validation (XLDI)*, 2012, vol. 412, p. 332.

[236] C. Monsanto, N. Foster, R. Harrison, and D. Walker, “A compiler and run-time system for network programming languages,” *Acm sigplan Not.*, vol. 47, no. 1, pp. 217–230, 2012.

[237] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker, “Composing software

defined networks," in *10th $\{$USENIX$\}$ Symposium on Networked Systems Design and Implementation ($\{$NSDI$\}$ 13)*, 2013, pp. 1–13.

[238] T. Koponen *et al.*, "Network virtualization in multi-tenant datacenters," in *11th $\{$USENIX$\}$ Symposium on Networked Systems Design and Implementation ($\{$NSDI$\}$ 14)*, 2014, pp. 203–216.

[239] T. Nelson, A. D. Ferguson, M. J. G. Scheer, and S. Krishnamurthi, "Tierless programming and reasoning for software-defined networks," in *11th $\{$USENIX$\}$ Symposium on Networked Systems Design and Implementation ($\{$NSDI$\}$ 14)*, 2014, pp. 519–531.

[240] R. Soulé *et al.*, "Merlin: A language for provisioning network resources," in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, 2014, pp. 213–226.

[241] H. Kim, J. Reich, A. Gupta, M. Shahbaz, N. Feamster, and R. Clark, "Kinetic: Verifiable dynamic network control," in *12th $\{$USENIX$\}$ Symposium on Networked Systems Design and Implementation ($\{$NSDI$\}$ 15)*, 2015, pp. 59–72.

[242] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, 2014.

[243] B. O'Connor *et al.*, "Using P4 on fixed-pipeline and programmable Stratum switches," in *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, 2019, pp. 1–2.

[244] M. Rupp, S. Schwarz, and M. Taranetz, *The Vienna LTE-Advanced Simulators: Up and Downlink, Link and System Level Simulation*, 1st ed. Springer Singapore, 2016.

[245] S. A. Busari, K. M. S. Huq, S. Mumtaz, and J. Rodriguez, "Impact of 3D Channel Modeling for Ultra-High Speed Beyond-5G Networks," in *2018 IEEE Globecom Workshops (GC Wkshps)*, Dec. 2018, pp. 1–6.

[246] M. Xiao *et al.*, "Millimeter Wave Communications for Future Mobile Networks," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 9, pp. 1909–1935, 2017, doi: 10.1109/JSAC.2017.2719924.

[247] M. Mezzavilla *et al.*, "End-to-End Simulation of 5G mmWave Networks," *IEEE Commun. Surv. Tutorials*, vol. 20, no. 3, pp. 2237–2263, 2018.

[248] L. Zhang, J. Liu, M. Xiao, G. Wu, Y. Liang, and S. Li, "Performance Analysis and Optimization in Downlink NOMA Systems With Cooperative Full-Duplex Relaying," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 10, pp. 2398–2412, Oct. 2017, doi: 10.1109/JSAC.2017.2724678.

[249] G. Yang, M. Xiao, M. Alam, and Y. Huang, "Low-Latency Heterogeneous Networks

with Millimeter-Wave Communications,” *IEEE Commun. Mag.*, vol. 56, no. 6, pp. 124–129, Jun. 2018, doi: 10.1109/MCOM.2018.1700874.

[250] 5GPPP, “View on 5G Architecture (Version 2 . 0),” no. July, 2017.

[251] Y. Wang, J. Xu, and L. Jiang, “Challenges of System-Level Simulations and Performance Evaluation for 5G Wireless Networks,” *IEEE Access*, vol. 2, pp. 1553–1561, 2014.

[252] M. Fedor, M. L. Schoffstall, J. R. Davin, and D. J. D. Case, “Simple Network Management Protocol (SNMP),” no. 1157. RFC Editor, May 1990, doi: 10.17487/RFC1157.

[253] B. Claise, “Cisco Systems NetFlow Services Export Version 9,” no. 3954. RFC Editor, Oct. 2004, doi: 10.17487/RFC3954.

[254] “RPC: Remote Procedure Call Protocol specification: Version 2,” no. 1057. RFC Editor, Jun. 1988, doi: 10.17487/RFC1057.

[255] R. Enns, M. Björklund, A. Bierman, and J. Schönwälder, “Network Configuration Protocol (NETCONF),” no. 6241. RFC Editor, Jun. 2011, doi: 10.17487/RFC6241.

[256] M. Bjorklund, “RESTCONF Protocol,” 2017, Accessed: Oct. 04, 2021. [Online]. Available: http://www.rfc-editor.org/info/rfc8040.

[257] M. Bjorklund, “Internet Engineering Task Force (IETF) The YANG 1.1 Data Modeling Language,” 2016, Accessed: Oct. 04, 2021. [Online]. Available: http://trustee.ietf.org/license-info.

[258] D. M. T. Rose, S. Hollenbeck, and L. M. Masinter, “Guidelines for the Use of Extensible Markup Language (XML) within IETF Protocols,” no. 3470. RFC Editor, Jan. 2003, doi: 10.17487/RFC3470.

[259] “The MongoDB 5.0 Manual — MongoDB Manual.” https://docs.mongodb.com/manual/ (accessed Oct. 04, 2021).

[260] V. D. Philip, Y. Gourhant, and D. Zeghlache, “Preliminary analysis of 4G-lte mobile network sharing for improving resiliency and operator differentiation,” in *E-Technologies and networks for Development*, Springer, 2011, pp. 73–93.

[261] N. Promwongsa *et al.*, “A Comprehensive Survey of the Tactile Internet: State-of-the-Art and Research Directions,” *IEEE Commun. Surv. Tutorials*, vol. 23, no. 1, pp. 472–523, 2021, doi: 10.1109/COMST.2020.3025995.

[262] R. Neisse, G. Steri, and I. Nai-Fovino, “A blockchain-based approach for data accountability and provenance tracking,” in *Proceedings of the 12th International Conference on Availability, Reliability and Security*, 2017, p. 14.

[263] M. Pilkington, “11 Blockchain technology: principles and applications,” *Res. Handb. Digit. Transform.*, p. 225, 2016.

[264] L. Gavrilovska, V. Rakovic, and D. Denkovski, “From Cloud RAN to Open RAN.,” *Wirel. Pers. Commun.*, vol. 113, no. 3, pp. 1523–1539, 2020.