Intelligent Routing as a Service (iRaaS)

A Flexible Routing Framework for Knowledge-Defined Networks

Author A *, Author B *, Author C*, and Author D *

* Company, Country

Emails: {Author A, Author B, Author C, Author D}@companyName.com

***Abstract*—The scope of the Sixth-Generation Self-Organized Networks (6G-SON) advances its predecessor's capability towards agility, flexibility, and adaptability. On-demand overlay networking technologies have shown a prominent maturity while coping with the rising complexity and scale of enterprise, service provider, and data centre networks. In the recent past, the Software-Defined Networking paradigm has offered Model Driven Programmability resulting in minimizing the network management complexity through automation and orchestration. However, leveraging Machine Learning-driven network optimization, a.k.a. Knowledge-Defined Networking (KDN), has still been a domain of interest for the Network Softwarization research community. In this article, we propose Intelligent Routing as a Service (iRaaS) architecture as an application layer cognitive routing framework for KDNs. iRaaS offers routing logic customization (i.e., customizing metric function, path-discovery algorithm, etc.) and provides an option to include heuristic parameters from trained models as a part of the metric calculation. iRaaS sits on the application plane above the knowledge plane in a KDN stack, thus providing platform- and vendor-agnostic coupling with existing network infrastructures. This article covers the scope of iRaaS by using reliability as a heuristic for standard path-discovery algorithms e.g., Shortest Path First (SPF) and Diffusion Update algorithm (DUAL) along with the architectural specification. We validate our approach through a Proof-of-Concept deployment.**

***Index Terms*—SDN, KDN, Routing-as-a-Service, Network Programmability & Automation, Routing API.**

I. Introduction

As computer networks become increasingly complex, manual network optimizations become less feasible. As a result, many organizations have turned to network automation, which offers improved efficiency and reduced human error. Network automation is used to configure, provision, manage, and test devices and systems, improving effectiveness and redundancy, and meeting compliance standards. With the active rollout of the 5G ecosystem, Softwarization, Virtualization, Cloudification (RFC 7868), and Interoperability have been the prominent adaptations by the stakeholders. The 5G Infrastructure Public Private Partnership (5G PPP) architectural working group identifies two new stakeholders in the domain, namely, Virtualization Infrastructure Service Provider (VISP) and Data-Centre Service Provider (DCSP). Additionally, Cloud-native architecture has become a de-facto standard for contemporary Network Functions Virtualization (NFV) deployment. It provides agile, lightweight, and manageable solutions along with a seamless blend of DevOps principles and practices. This results in consumer enterprises opting for features like Zero Touch Provisioning (ZTP), which offloads a significant amount of network administration burden to the service provider end, where the networks are centrally managed and orchestrated with predefined policies. To accommodate this transition, centralized optimization algorithms (Routing, Quality-of-Service (QoS), etc.), infrastructure automation tools (Ansible, Puppet, Chef, etc.), high-availability protocols (first-hop redundancy protocol (FHRP), Stateful Switchover (SSO), etc.), and ML-based network-state prediction models (e.g., Time-Series analysis, Traffic-classification, route- prediction, etc.) play a key role.

The developments as mentioned above clearly demand the underlying telecommunication infrastructure to be extremely flexible and self-aware. Although proposed in IMT-2020, the Ultra-Reliable and Low Latency Communications (URLLC) verticals are yet to be achieved completely. Due to spectrum limitations, the data plane latency cannot be suppressed any further. Therefore, research has turned to optimising the control plane using Routing Optimization, which aims to reduce latency due to the optimal path-finding process. Nevertheless, RO enablers such as rapid converging routing protocols, optimal route-redistribution, route reliability estimation, and Link-State prediction bring forth scalability and flexibility challenges when considering deploying multi-vendor, multi-protocol, elastic, and dynamic networks. By design, classical routing protocols (e.g. OSPF, EIGRP, ISIS) do not support optimization by centralized computing models, and there exists no routing protocol that natively supports a Software Define Network (SDN) and/or Knowledge Defined Networking (KDN) model.

Further challenges are presented when considering using SDN in dynamic environments such as Mobile Ad hoc Networks (MANETs) to support mission-critical use cases, e.g. deployment of networks in disaster zones. In such environments, reliable networks are crucial, while high mobility and link dynamics have a major impact on network performance and convergence.

In this paper, we address the aforementioned reliability, scalability and flexibility challenges of such cases of Self-Organized Networks (SON). We present an intent-based, data plane-agnostic and intelligent Routing-as-a-Service platform

Identify applicable funding agency here. If none, delete this.

that:

- Allows the deployment of customized routing logic and enables the life-cycle management and use of ML models for optimizing network reliability.
- Provides a centralized service-based architecture overseeing the entire network, diminishing the time-consuming control plane packet exchange, hence reducing control plane-induced latency.
- Captures the declarative requirements of the platform user and the end-to-end network topology abstracted by the underlying SDN/KDN and legacy controller-less infrastructure.
- Uses robust telemetry to capture the state of the end-to-end network.
- Adopts state-of-the-art standards and open-source solutions, validating our solution's sustainability and extensibility.

The remainder of this paper is the following, section II gives a background of the context, section III describes the high-level architecture of iRaaS, section IV provides details of the iRaaS system design with sequence diagram and a bespoke telemetry architecture named ShellMon, section V validates the iRaaS architecture with a proof of concept testbed setup related results, Finally, Section VI concludes this article with a summary and future scope aimed for this work.

II. Background

In MANETs, routing protocols can be classified into three basic categories: proactive, reactive, and hybrid routing protocols. These routing protocols update the routing table information periodically or in response to changes in the network topology. MANET routing suffers from scalability, bandwidth constraints, availability, and security. Traditional Internet Protocol (IP) routing does not consider radio link characteristics. Cisco has introduced the concept of Radio Aware Routing (RAR) to optimize IP routing over diverse radio networks to give users real-time access to critical information while on the move [1]. The latest RAR protocol, Dynamic Link Exchange Protocol (DLEP), has been standardized in IETF (RFC-8175). Cisco has worked towards routing optimization in MANET [2]. OSPFv3 improves routing efficiency and reduces overhead traffic in MANET environments so that network clusters can scale to support more users.

In the same manner, Enhanced Interior Gateway Routing Protocol (EIGRP), formerly known as a Cisco-proprietary, has introduced a new routing protocol that is characterized as a distance-vector routing protocol using the Diffusing Update Algorithm (DUAL) [3]. EIGRP has been designed for operation in large networks, supports classless routing, allows to exchange of network information with a variable network mask, supports the usage of authentication for message transfer, and very fast network convergence.

Nevertheless, the significant increase in the size and complexity of computer networks results in slower convergence times and decreased network performance for traditional hardware-based routing. One of the main reasons for that is that these routing protocols are inherently distributed in nature; thus, they rely on the underlying communication systems to exchange information. The communication overhead to complete the distributed algorithms of such routing protocols creates a bottleneck.

To address these problems, the Software-Defined Networking (SDN) paradigm decouples the control plane (CP) and moves it to a logically centralized location, keeping the Data Plane (DP) distributed [4]. In SDN architecture, network devices only forward traffic, whereas all the control functionalities execute centrally at the CP. The CP sees the underlying network from a bird's-eye-view similar to the Link State Routing (LSR) model but not replicating them to individual routers, which diminishes the communication bottleneck.

Routing algorithms typically operate in two main phases: building a topology and calculating the shortest paths. They rely on neighbouring routers to gather network reachability information, either through distance vector or Link State methods. When the network topology changes, these protocols pause to recalculate routes, a process called Convergence, which delays packet forwarding. The convergence time increases with the network size. In contrast, SDN simplifies this by allowing a centralized controller to overview the entire network, reducing both communication and computational complexity. This setup accelerates routing decisions by minimizing control packet exchanges and performing calculations at the controller.

So far, various network models and protocols have been developed including Next Generation SDN (NG-SDN) from Open Networking Foundation (ONF), Cisco- Viptella SDWAN, SD-Access, VMWare NSX, 5G-PPP software networks, Disaggregated platforms like ONIE, ONL, SAI, SONiC form Open Compute Project (OCP). 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 and QoS Routing. The former leverages Multi-Protocol Label Switching (MPLS)-based communication at the underlay and replaces Label Discovery Protocol (LDP) and Resource Reservation Protocols (RSVP) in CP. Quality of Service (QoS) Routing steers traffic to an optimal path preserving various communication constraints. Furthermore, as shown in [5] several SDN routing solutions integrate ML-based optimisations.

From the perspective of Internet routing, benefits of Routing-as-a-Service have been presented in [6] explaining how it can resolve the conflict of path selection for satisfying the QoS requirements of end-to-end network users while allowing Autonomous System administrators (e.g. ISPs) to control traffic flows over their infrastructure fully.

Incorporating SDN, a RaaS platform in [7] allows the platform user to select routing algorithms used as network functions to compose a customized routing service. Nevertheless, this solution relies on existing routing protocols.

A Software-defined Wide Area Network (SD-WAN) platform is presented in [8] calculating optimal paths using CPLEX over Open Network Operating System (ONOS)-controlled routers with possible extension to ML-based optimal path calculation. Nevertheless, it does not provide a way for customizing the metric function used for calculating the optimal path and does not offer a telemetry architecture that any potential ML models will require.

Finally, application layer routing is particularly beneficial for heterogeneous SDNs, where the network infrastructure is diverse and programmatically controlled. In such environments, application layer routing leverages the centralized intelligence of SDN controllers to make dynamic, application-aware decisions. This approach enables the network to adapt in real-time to changing application demands and network conditions, optimizing for factors like bandwidth, latency, and security requirements specific to each application. For heterogeneous SDNs, which may span across different *domains* and incorporate a variety of physical and virtual network functions, application layer routing ensures that traffic is efficiently and intelligently routed, taking advantage of the programmable nature of SDNs to enhance performance, scalability, and resilience. By aligning network behaviour with application requirements, it facilitates a more responsive and optimized network environment tailor-made for the diverse needs of modern digital applications.

III. High-level System Architecture

Based on the analysis in the previous section, we envision a system that extends the standard SDN implementation and provides routing in an "as-a-service" fashion. This implies that the routing service will be provided as an application remotely accessible by end-users and administrators and deployed in a softwarized fashion.

Starting by the definition of the *domain*, we imply a system which is a "component subsystem" of a wider system where the internal design and/or operations of the system are not fully exposed outside the domain. This could, for example, be an administrative domain (e.g., an Internet Service Provider (ISP)). We later envision that there will be a management plane responsible for the intelligent intent-based routing across the different *domains*. This architecture can be seen in Figure 1. This figure also introduced our key functional blocks that provided the envisioned capability described in this paper. As per our envisaged functionality, we propose to implement a centralized application-layer routing model that accompanies and extends traditional SDN and non-SDN controllers, enhancing their performance, scalability and resilience of the system.

Our envisaged iRaaS application sits within a Cross-Domain Network Manager (CDNM) responsible for orchestrating the different services deployed there. The Cross-Domain Route Management (CDRM) takes the routing decisions and communicates them downstream to the administrative domain controlled by an Intra-Domain Network Manager (IDNM). The domain comprises a hybrid SDN network spanning across multiple controllers (i.e., SDN and Non-SDN). We refer to the combined control plane of SDN and Non-SDN controllers within an administrative domain as Shim-Layer, as it abstracts the platform specificity of the underlying data plane from the planes above. We break down our application layer into two different entities, i.e., the iRaaS Client and the iRaaS Server, to enhance load balancing and enable higher scalability. Moreover, a telemetry application is envisioned that collects real-time KPIs from the different domains and sends them to the RaaS server to enable intelligent decisions. The following sections describe in detail the aforementioned system entities.

A. iRaaS Client

iRaaS client receives Route Intent from external admins through an API proxy, which enables an administrator (human or program) to interact through a single point of contact. In addition to selecting common routing attributes, e.g., routing protocol and its associated parameters, path manipulation logic, etc. iRaaS offers an admin to customize routing logic. A custom routing logic can be known as the Shortest Path Algorithm (e.g., SPF, DBF, DUAL, etc.) or a bespoke one encoded in compliance with the iRaaS Server API. Additionally, it offers the ability to customize the cost function (e.g., an admin might use SPF as routing logic for implementing LSR in a hierarchical topology with EIGRP-like composite metric). This level of flexibility of iRaaS contributes to its novelty and uniqueness. The iRaaS client is also responsible for building an aggregated graph of the underlying topologies of both SDN and non-SDN available controllers. The iRaaS Client sends the Route Intent and aggregated graph to the iRaaS Server.

The iRaaS architecture supports hybrid SDN at the access plane. The admin informs the Client about the respective controllers' management interface address while requesting through the API Proxy. The client establishes management access with all controllers at the access network leveraging the CaaS service and respective drivers (SDN/Non-SDN) and fetches the controller-wise downstream topologies through standard management protocol, e.g., NETCONF [9], RESTCONF [10], etc.). Finally, the northbound interfaces between the iRaaS Client and the Shim layer may use an exterior gateway transport for connectivity.

B. iRaaS Server

The server receives the Route Intents and graphs from the client through a standardised interface. The server integrates the graphs received and the cost of the link costs calculated by the cost function specified and (near) real-time telemetry of the metric KPIs. The iRaaS Server calculates the optimal path(s) as per the routing logic consulting with an MLOps pipeline and responds back to the iRaaS Client, which then configures the access network devices through the downstream controllers. The MLOps pipeline can be a standard ML pipeline where various ML models can be tested, validated, and trained on

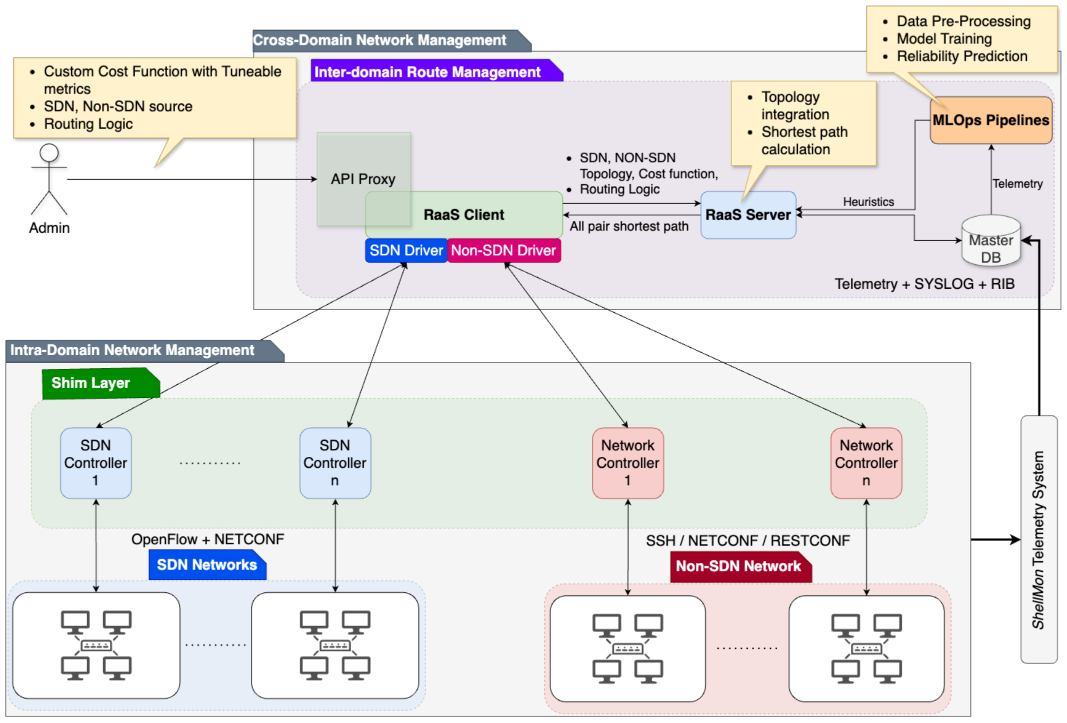


Fig. 1. The proposed iRaaS System Architecture. The system consists of multiple domains and a cross-domain management plane that is responsible for the orchestration and routing across each domain.

existing historical telemetry data and, when deployed in the system, can provide intelligent decisions and optimal routing paths. An example of such a module using telemetry data to predict a Sharpe Ratio-based path-reliability by performing a time-series analysis of metrics data over a rolling time window can be found in [11].

C. *Telemetry Framework*

The telemetry framework resides in conjunction with the administrative domains and the IDNM and provides a platform and vendor-agnostic multi-modal implementation that collects data in a standardized fashion. The telemetry framework exposes a number of RESTful interfaces and publish-subscribe messaging buses responsible for collecting telemetry data and storing them in the database available in the CDRM. In the following section, we go into more detail on the implementation of the telemetry framework describing the different communication modes supported.

IV. System Design and Provided Functionality

Following the above high-level architecture, in this section, we describe in more detail how iRaaS is provided by our system and describe all function blocks that build up our platform. As briefly discussed earlier, our proposition supports both traditional SDN implementations and monolithic implementations that communicate without available agents. Our system, as described, is intended to provide a platform-agnostic, flexible, and programmable path-calculation mechanism that, operating at the application layer, enables the scalability and future-proofing of modern routing solutions as well as the integration with future network architectures such as SONs in the Sixth-Generation (6G) networks.

A. *Routing-as-a-Service*

RaaS is a data plane-agnostic principle of flexible and programmable path calculation mechanism served at the application layer [6], [7]. In our system, we define the Routing Logic as a pair of metric functions f_{metric} and shortest-path algorithm f_{sp}. Where, $f_{metric}(w_i a_i | i \in [1, n], i \in \mathbb{N})$ is an arbitrary multivariate scalar function with a finite dependent variable set $A = \{a_i\}$, called attributes, weighed by corresponding weighing factor $w_i \in W$. f_{sp} takes a weighted simple graph (i.e., free of self-loops and parallel-edges) $G(V, E)$, where V and E are the vertices (nodes) and edge sets of G respectively, with a pair of nodes $v_s, v_d \in V$ and returns optimal path(s) $P_{s,d} \in 2^E$ such that path cost is minimal. Therefore, considering an Intent-Based Networking (IBN) paradigm, a RaaS application must accept the routing logic $(f_{metric}(A, W), f_{sp}(G(V, E)))$ as an *intent* though an open-API, and returns the optimal path(s) $\{P_{s,d}\}$ as the response.

Figure 2 depicts the sequence diagram of a RaaS application operating in Client-Server mode on a Hybrid-SDN topology.

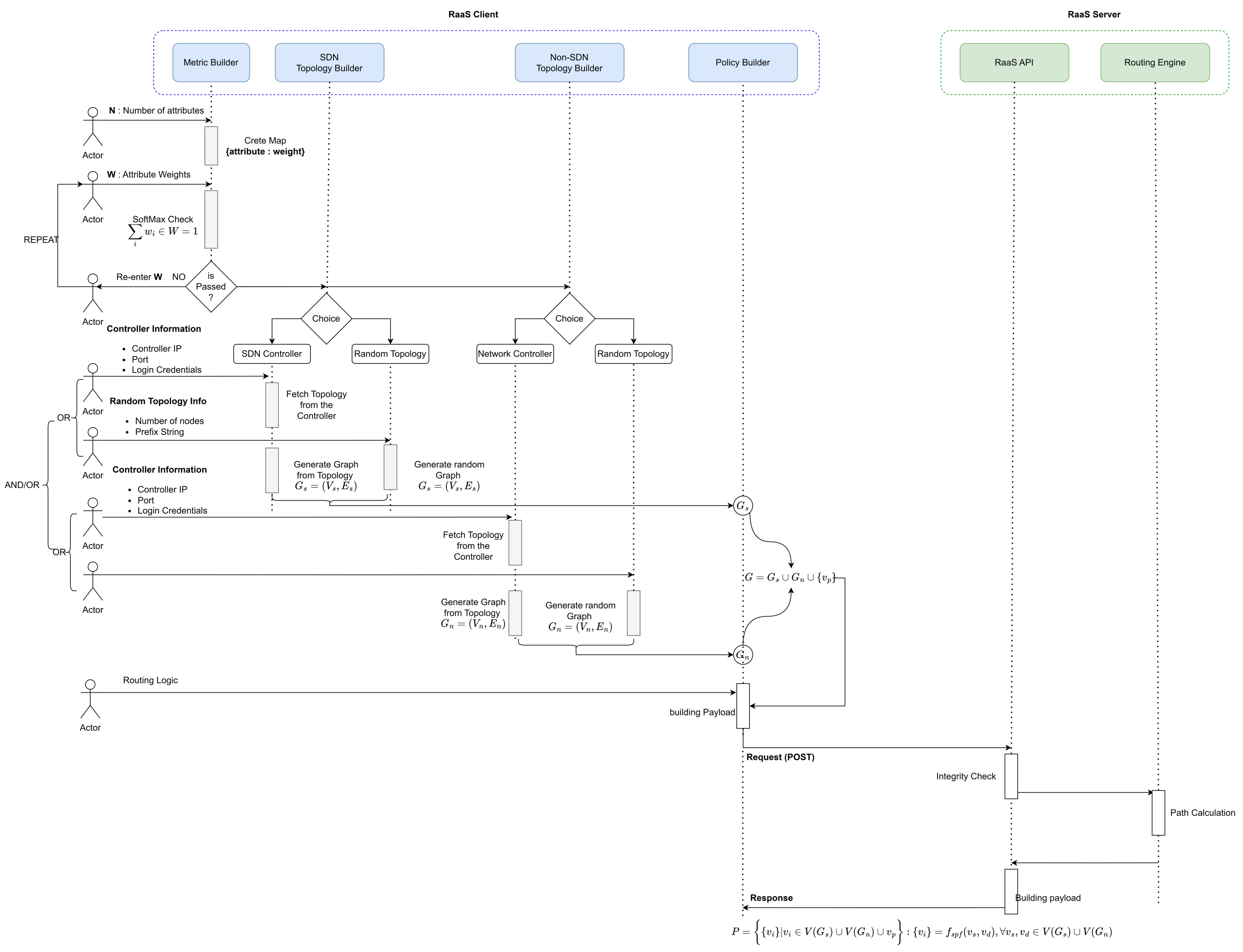


Fig. 2. Sequence diagram of RaaS Client-Server model for a Hybrid-SDN topology

As described in Section III, splitting the application into client and server side results in load balancing and higher scalability. Where the client-side operations are I/O-intensive due to intent-API exposure and network controllers interfacing, the server-side is more compute-intensive as it runs the graph algorithms. The operation pipeline comprises the following phases:

1) *Metric Building*: In this phase, the attribute set A such that $|A| = N$ is provided with their corresponding weights W along with a metric function f_{metric}. The weighing factor $w_i \in W$ of an attribute $a_i \in A$ signifies the priority of the attribute in the f_{metric} definition. therefore, the sum of all w_i must be 1.
2) *Topology Building*: In this phase, the iRaaS client interfaces with the network controllers to fetch their underlying topology. A network controller abstracts the platform-level information of the network and returns a graph representation of the topology. In a hybrid-SDN scenario, the RaaS client interfaces with each instance of the SDN and Non-SDN controllers to fetch their underplaying topologies into a set of graphs $\mathbb{G} = \{G_j(V_j, E_j)\}$. In such case, we propose a normalisation method of transforming $\mathbb{G}$ into a fused-graph $G(V, E)$ by adding a pseudo-node v_p which is fully adjacent with a designated node $v_{dn} \in V_j$ from each graph G_j. An ideal designated node is preferably but not necessarily to be at the centre of its topology. This mimics the behaviour of a summary point in traditional routing protocols. The cost of the logical link between each v_{dn} and v_p is an explicitly defined non-zero value. This is because the routing between topologies across controllers under the same administrative domain requires the inter-controller data path to be involved, which is considered as an exterior link. v_p represents the exterior link, and the cost between v_p and v_{dn} represents the cost of accessing it.
3) *Policy Building*: The iRaaS Client in this phase prepares the routing logic by packing the normalised topology

$G(V, E)$ and the metric function and sending it to the iRaaS Server for processing.

4) *Integrity Check*: The iRaaS server checks the integrity of the routing logic received from the iRaaS client and passes it to the routing engine upon successful validation.
5) *Path Calculation*: The iRaaS servers calculates link-costs by applying the f_{metric} function on telemetry data and returns the optimal path(s) as per the shortest path algorithm.

B. Intelligent Routing Behaviours

The iRaaS Client and Server components provide the underlying framework for making intelligent routing decisions. In this section, we describe the set of behaviours that are essential for an intelligent routing model. In either case of using a heuristic algorithm or an ML model, these behaviours are critical for providing an optimal routing path across different domains. Moreover, for the graphs aggregated in our application plane, two algorithms are described that can minimise the footprint of our application and provide a more robust implementation.

1) Topology Aggregation: The aforementioned developments rely on typologies either reactively or proactively and maintain a local database for it. An iRaaS application must leverage the network controllers' north-bound interface to fetch and aggregate individual topologies into a global topology map. The aggregation process could be arbitrary, however, the graph aggregation algorithm must consider the inter-controller connectivity cost while taking the union of the candidate graphs. In the previous subsection, we took a simplistic approach by introducing a logical pivot-vertex v_p per aggregation and a designated vertex v_{dn} per candidate graph with a non-zero constant cost between them representing the inter-controller communication cost.

2) Cost Normalization: The aggregated graph generated from the Topology Aggregation phase is an in-memory data structure at the iRaaS application server. The telemetry system monitors and maintains a database of network key performance indicators (KPIs); the metric function uses a subset of selected attributes from the KPIs to weight the aggregated graph. Traditional routing protocols only consider link costs for path-finding. However, in a softwarised network infrastructure where network functions are not necessarily physical, computational costs are also significant in calculating end-to-end costs. Therefore, a composite metric that considers both link and computational cost into a normalised cost is required, resulting in a more accurate routing decision. That said, normalising computational cost with link cost requires an isomorphic transformation of the aggregated graph. Since the computation load appears as a weighted self-loop, hence, it makes the graph a regular graph rather than a simple one. A simple graph is a prerequisite for running any shortest-path algorithm on it. Authors in [12] present such a normalization technique named Stochastic Temporal Edge Normalisation (STEN).

3) Redundant Path Discovery: To ensure rapid convergence, the path-finding algorithm f_{sp} must not only discover the best path but a set of alternate or redundant paths. This may resemble the Feasible Successor approach of DUAL which allows rapid path switchover in case of a primary path failure without involving any diffusion updates in the topology. However, DUAL's loop-prevention mechanism may not find any Feasible Successor despite having available alternate paths if none of the non-successor neighbours pass the Feasibility Criteria. In such a case, EIGRP puts DUAL into an active state and initiates querying neighbours for alternate path discovery. This issue occurs because EIGRP is a distance vector routing protocol. In RaaS, the global topology-building process follows the link-state approach. Therefore with a global view of the topology, an explicit loop-prevention mechanism is unnecessary and hence, f_{sp} can safely discover and maintain alternate paths to reactively switch between them when the primary one fails.

4) Reliability-Based Metric and Reactive Route Ranking: As the KPIs fluctuate over time with the network dynamics, it may invoke f_{sp} unnecessarily many times, resulting in inefficient runtime behaviour such as route-flapping and high computational complexity. We propose two approaches to optimise the runtime footprint for RaaS applications.

The first one is Reactive Route Ranking where f_{sp} processes the aggregated graph in two phases. In Phase-1, it calculates all possible paths between all pairs of vertices. Defining a cut-off diameter and initializing the link costs with seed values speed up the process further. As the path discovery between each pair is sequentially independent, therefore they can run simultaneously. The result is a forest of Shortest Path Trees (SPT) where each instance is rooted by the destination vertex v_d with all branches representing a unique path to the source v_s as the identical leaf for all branches. In Phase-2, a Raas application may calculate the path cost of each branch for all SPTs and rank the paths of each SPT. The two-phase approach limits re-convergence to only topology change scenarios. If the topology grows then only new connections are updated in the SPTs, and if it shrinks, then vanishing links are updated with an infinite cost.

Second, we propose the use of Reliability as a metric. Reliability is statistically calculated using Sharpe Ratio [13] from a rolling window of f_{metric}. RaaS application uses the expected Reliability from a Recurrent Neural Network (RNN) to make routing decisions. Therefore, iRaaS avoids fluctuating unreliable routes and emphasizes more on reliable paths rather than the shortest, least-costly and fastest paths.

In summary, the above two optimization steps ensure minimum invocation of re-convergence with Reactive Route Ranking and prioritize reliable routes. Authors in [11] have explained the above techniques in detail.

C. Telemetry Framework and its Communication Modes

This section describes the telemetry framework architecture, which carries the monitoring data from the data to the applica-

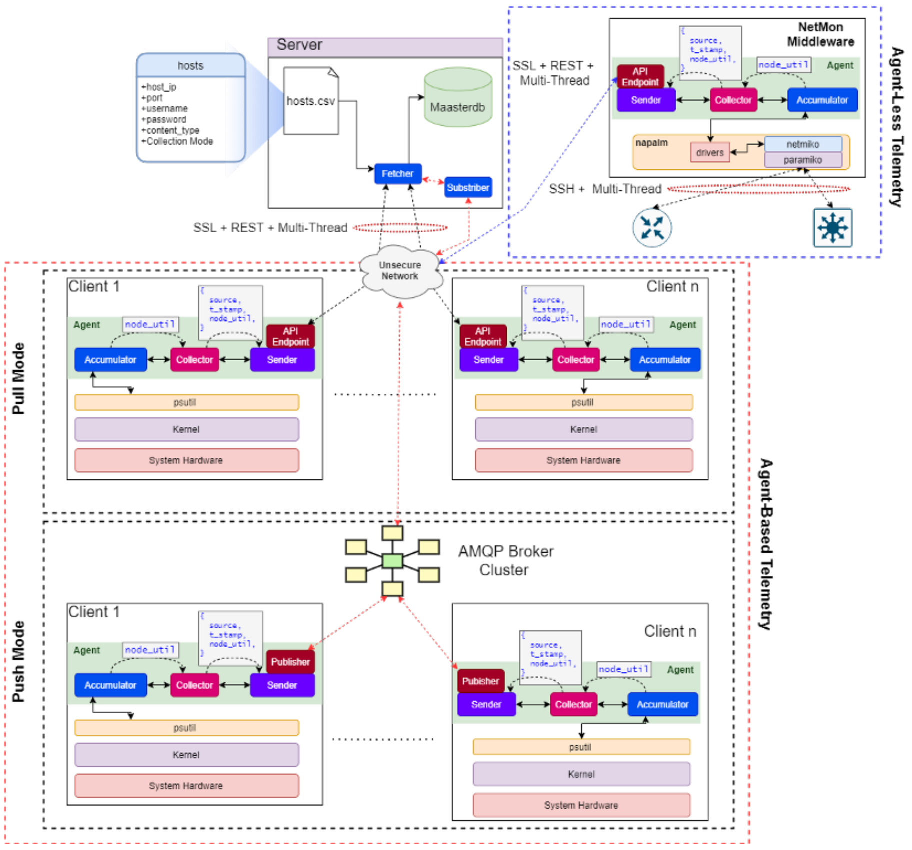


Fig. 3. System Level Diagram of the Telemetry System

TABLE I
OPERATIONAL SPECIFICATION OF *ShellMon*

Attributes	Type	Usage
Mode of Transport	Mode 1: Request-Response (RESTful)	Pull based telemetry collection
	Mode 2: Publish-Subscribe (AMQP)	Push based telemetry collection
Mode of Collection	Mode 1: Agent Based	Target device with monolithic karnel
	Mode 2: Agent Less	Target device allows agent installation
Serialisation Format	JSON	Standard data format, low parsing footprint

tion plane. The architecture of this system can be seen in Figure 3. We propose a modular, multi-modal telemetry API named *ShellMon* for the RaaS framework. Unlike vendor, platform and version-dependent telemetry protocols such as SNMP [14], NetFlow [15], *ShellMon* provides platform and vendor agnostic multi-modal telemetry with a common standard data format. Table I summarises the operational specifications of *ShellMon*.

ShellMon server maintains a *host* file containing the clients' information (e.g., hostname, port number, access credentials, the content type of the payload and connection mode). The *Fetcher* and *subscriber* modules use request-response and publish-subscribe modes, respectively. The telemetry is collected and stored in a Master database shared with the iRaaS application. The remainder of this section describes the various modes of communication that *ShellMon* offers.

1) Transport Mode 1: RESTfull Request-Response: In this mode, the collection mechanism operates in a RESTful fashion. *ShellMon* server sends poll requests in regular intervals, which triggers the clients to invoke device-level local KPI collection. Clients timestamp the KPI samples and send them back to the server. That said, this mode relies on HTTP's keep-alive mechanism to monitor the liveliness of the clients.

2) Transport Mode 2: Publish-Subscribe : This mode is suitable for large-scale client-base, where the number of ports available on the server side is constrained. The agent comprises identical modules as of the RESTful mode. However, instead of an API end-point, it publishes KPIs from a local publisher. *ShellMon* uses an Advanced Message Queuing Protocol (AMQP) [16] message broker for the transport.

3) Collection Mode 1: Agent-Less Telemetry: This mode is suitable for network devices running monolithic kernels such as Cisco IOS and Juniper JunOS, which do not allow the installation of external agents. The *NetMon* middleware communicates with the network devices through asynchronous SSH sessions to collect telemetry information using a multi-vendor SSH library called Napalm [ref-netmiko]. The Accumulator module of the NetMon agent collects monitoring data samples

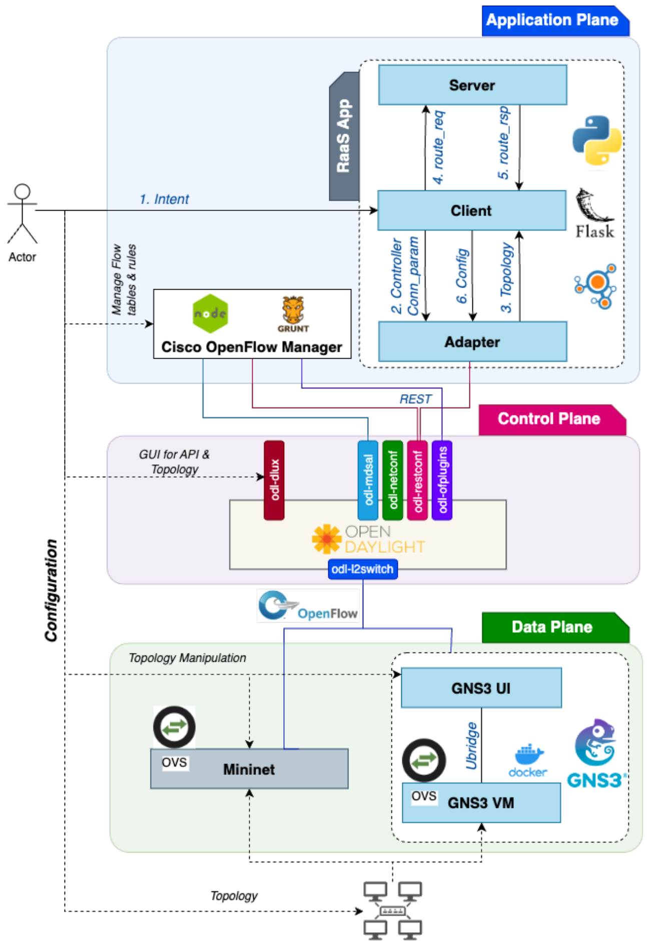


Fig. 4. iRaaS Deployment Diagram

from Napalm into a key-value store named *node_util*. The Collector module tags it with *source_id* and timestamp. Finally, the Sender module exposes an API endpoint for the monitoring data to be polled. NetMon uses a request-response mechanism for polling, i.e., a poll request from the *ShellMon* server initiates the collection cycle; therefore, the agent does not require any local queue.

4) Collection Mode 2: Agent-Based Telemetry: In this mode, an agent runs on top of the network device kernel. The Accumulator, Collector and Sender module behaves the same as the Agent-Less mode.

V. Test-bed Setup and Proof-of-Concept

Figure 4 depicts the deployment diagram of the RaaS testbed along with the technology stack used. The application, control and data planes are segregated by three Virtual Machines (VMs) for runtime isolating. The data plane comprises a Mininet [17] and a GNS3 for simulating network topologies. Mininet supports Open-V-Switches (OVS) natively, we used a containerized version of OVS to deploy in GNS3. We use OpenDaylight (ODL) at the control plane to interface with OVS and the RaaS application using OpenFlow v1.3 and RESTCONF respectively. To carry out control plane operations, we use the following Opendaylight features; *odl-l2swicth* module controls the south-bound interface using OpenFlow, *odl-restconf* controls the north-bound interface using RESTCONF, *odl-mdsal* provides Model-Driven Service Abstraction Layer to parse YANG data-models, *odl-ofplugin* provides a standard interface between the control and data plane, and *odl-dlux* provides ODL GUI.

The application plane hosts the iRaaS application and a Cisco OpenFlow Manager (OFM) VM. OFM inspects the topology from the ODL controller and provides a GUI-based flow management tool. The Client module of the iRaaS app receives intent from the administrator that includes the controller access information and routing logic. The Client interfaces with the Adapter module to fetch topology from the data plane following the methods specified in figure 2. Further, it sends a *route_request* to the Server Module which computes the optimal paths and replies with a *route_response*. In this setup, we have developed the iRaaS app using Flask micro-framework for API development and the *NetworkX* library for graph computation.

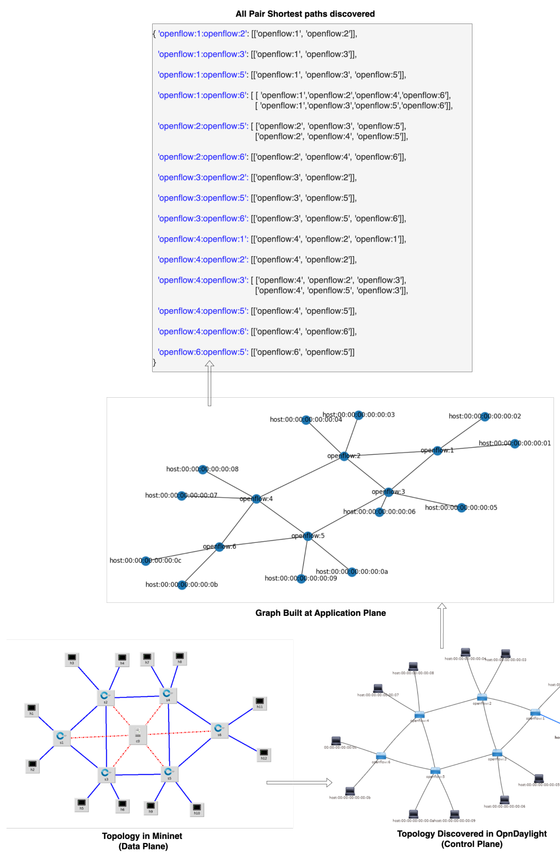
Figure 5 depicts the flow of topology processing from the Mininet data plane to the RaaS application through the Open-Daylight control plane. The shown example illustrates a partial mesh topology of six Open-V-Switch instances each connecting two hosts and communicating with a remote SDN controller over OpenFlow v1.3. iRaaS adapter fetches the topology from the OpenDaylight controller and the iRaaS client builds a graph data structure using the NetworkX library as shown in the figure. In this test, we choose all-pair Dijkstra's algorithm to find all routes between each pair of OpenFlow switches as shown at the top of the figure.

The above test results validate the proof of concept of the proposed architecture. However, the same is also capable of running customized routing algorithms by altering configuration at the iRaaS server. That said, the paper [11] shows rapid convergence in Knowledge-Defined Networks comparing the scalability against SPF and DUAL.

VI. Conclusion and Future Scope

This paper presents a system-level architecture of Intelligent Routing as a Service (iRaaS), a sequence diagram explaining the data between various iRaaS components and a robust telemetry architecture for collecting monitoring data from the underlying network infrastructure. A proof of concept setup also validates the operational capabilities of the proposed architecture with a deployment diagram detailing the assembly of various open-source components constituting the test bed used for experiments.

We aim to advance the iRaaS concept with a robust cognitive plane comprising additional machine learning-based operations algorithms such as traffic classification and state prediction.


All Pair Shortest paths discovered

```
{ 'openflow:1:openflow:2': [['openflow:1', 'openflow:2']],

  'openflow:1:openflow:3': [['openflow:1', 'openflow:3']],

  'openflow:1:openflow:5': [['openflow:1', 'openflow:3', 'openflow:5']],

  'openflow:1:openflow:6': [ [ 'openflow:1','openflow:2','openflow:4','openflow:6'],
                             [ 'openflow:1','openflow:3','openflow:5','openflow:6']],

  'openflow:2:openflow:5': [ ['openflow:2', 'openflow:3', 'openflow:5'],
                             ['openflow:2', 'openflow:4', 'openflow:5']],

  'openflow:2:openflow:6': [['openflow:2', 'openflow:4', 'openflow:6']],

  'openflow:3:openflow:2': [['openflow:3', 'openflow:2']],

  'openflow:3:openflow:5': [['openflow:3', 'openflow:5']],

  'openflow:3:openflow:6': [['openflow:3', 'openflow:5', 'openflow:6']],

  'openflow:4:openflow:1': [['openflow:4', 'openflow:2', 'openflow:1']],

  'openflow:4:openflow:2': [['openflow:4', 'openflow:2']],

  'openflow:4:openflow:3': [ ['openflow:4', 'openflow:2', 'openflow:3'],
                             ['openflow:4', 'openflow:5', 'openflow:3']],

  'openflow:4:openflow:5': [['openflow:4', 'openflow:5']],

  'openflow:4:openflow:6': [['openflow:4', 'openflow:6']],

  'openflow:6:openflow:5': [['openflow:6', 'openflow:5']]
}
```

Fig. 5. iRaaS calculates All-pair shortest path at the application plane

Acknowledgement

This work is a contribution by Project REASON, a UK Government funded project under the Future Open Networks Research Challenge (FONRC) sponsored by the Department of Science Innovation and Technology (DSIT).

References

[1] "Solutions - Radio Aware Routing: Enabling Communications on the Move White Paper — cisco.com," https://www.cisco.com/c/en/us/solutions/collateral/industries/radio-aware-routing-wp.html, [Accessed 26-02-2024].

[2] "IP Mobility: Mobile Networks Configuration Guide, Cisco IOS Release 15M&T - Cisco Mobile Networks [Cisco IOS 15.4M&T] — cisco.com," https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/mob$_{n}twks/configuration/15-mt/mob_{n}twks-15-mt-book/imo-mbl-ntwks.html, [Accessed 26-02-2024]$.

[3] D. Savage, J. Ng, S. Moore, D. Slice, P. Paluch, and R. White, "Cisco's Enhanced Interior Gateway Routing Protocol (EIGRP)," RFC 7868, May 2016. [Online]. Available: https://www.rfc-editor.org/info/rfc7868

[4] E. Haleplidis, K. Pentikousis, S. Denazis, J. H. Salim, D. Meyer, and O. Koufopavlou, "Software-Defined Networking (SDN): Layers and Architecture Terminology," RFC 7426, Jan. 2015. [Online]. Available: https://www.rfc-editor.org/info/rfc7426

[5] 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. 104 582–104 611, 2021.

[6] K. Lakshminarayanan, I. Stoica, and S. Shenker, "Routing as a service," EECS Department, University of California, Berkeley, Tech. Rep. UCB/CSD-04-1327, 2004. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2004/5916.html

[7] C. Bu, X. Wang, H. Cheng, M. Huang, and K. Li, "Routing as a service (raas): An open framework for customizing routing services," *Journal of Network and Computer Applications*, vol. 125, pp. 130–145, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804518303321

[8] O. Fares, A. Dandoush, and N. Aitsaadi, "Sdn-based platform enabling intelligent routing within transit autonomous system networks," in *2022 IEEE 19th Annual Consumer Communications Networking Conference (CCNC)*, 2022, pp. 909–912.

[9] R. Enns, M. Björklund, A. Bierman, and J. Schönwälder, "Network Configuration Protocol (NETCONF)," RFC 6241, Jun. 2011. [Online]. Available: https://www.rfc-editor.org/info/rfc6241

[10] A. Bierman, M. Björklund, K. Watsen, and R. Fernando, "RESTCONF Protocol," Internet Engineering Task Force, Internet-Draft draft-ietf-netconf-restconf-01, Jul. 2014, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-netconf-restconf/01/

[11] S. Ghosh, T. Dagiuklas, M. Iqbal, and X. Wang, "A cognitive routing framework for reliable communication in iot for industry 5.0," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 8, pp. 5446–5457, 2022.

[12] S. Ghosh, T. Dagiuklas, and M. Iqbal, "Energy-aware ip routing over sdn," in *2018 IEEE Global Communications Conference (GLOBECOM)*, 2018, pp. 1–7.

[13] W. F. Sharpe, "Mutual fund performance," *The Journal of business*, vol. 39, no. 1, pp. 119–138, 1966.

[14] J. Schönwälder, "Simple Network Management Protocol (SNMP) Context EngineID Discovery," RFC 5343, Sep. 2008. [Online]. Available: https://www.rfc-editor.org/info/rfc5343

[15] B. Claise, "Cisco Systems NetFlow Services Export Version 9," RFC 3954, Oct. 2004. [Online]. Available: https://www.rfc-editor.org/info/rfc3954

[16] S. Vinoski, "Advanced message queuing protocol," *IEEE Internet Computing*, vol. 10, no. 6, pp. 87–89, 2006.

[17] M. P. Contributors, "Mininet: An Instant Virtual Network on Your Laptop (or Other PC) - Mininet — mininet.org," https://mininet.org/, [Accessed 11-02-2024].