Received 16 May 2023, accepted 15 June 2023, date of publication 22 June 2023, date of current version 28 June 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3288813

RESEARCH ARTICLE

Building a Digital Twin Network of SDN Using Knowledge Graphs

DEEPU RAJ RAMACHANDRAN POTTI[1], TAHIR AHMED SHAIK[1], ANISH HIRWE[2], PRAVEEN TAMMANA[1], AND KOTARO KATAOKA[1]
[1]Indian Institute of Technology Hyderabad, Hyderabad 502285, India
[2]Indian Institute of Technology Palakkad, Palakkad 678557, India

Corresponding author: Deepu Raj Ramachandran Potti (cs21mtech12003@iith.ac.in)

ABSTRACT In SDN-based networks, contextual understanding of network behaviour and safe execution of operational decisions are important as part of the network management process. Digital Twin Network (DTN) is a promising concept which creates a virtual twin of a live SDN-based network to monitor the network from different angles with various granularities and enables it to verify an operational change without disturbing the live one. However, modelling the arbitrary decisions by an SDN Controller to form a DTN and managing the contextual information in a DTN are challenging tasks in various aspects. This paper proposes a data representation based DTN architecture integrating Knowledge Graph (KG) for data modelling and storage and Template as the context description approach. The combination of KG and Template can make the DTN management scalable with the flexibility of defining the contextual information with the relationships between network entities. It also enables efficient querying of the data storage and provides the reusability of functions and data storage for the DTN applications development. The proposed DTN architecture was implemented using an ONOS-based SDN Controller and Neo4j-based KG with built-in DTN applications for practical use. The PoC implementation exhibited short query response time and high query throughput in reading from and updating a KG, though the initial creation of a KG incurs a considerable delay which increases with its size.

INDEX TERMS Digital twin, digital twin networks, knowledge graphs, software defined networks (SDN), templates.

I. INTRODUCTION

Digital Twin Network (DTN) is a digital representation of a live network environment such as mobile access networks, data center networks, a campus enterprise network, etc. [1]. The primary goal of DTN is to provide a unified Digital Twin platform using context-rich and efficient data modelling standards to enable various use cases [2] such as safe validation of network configurations, user intent-based network automation and behavioural analysis [3], network optimisation [4], and data collection for analytical purposes [5] such as enhancing network efficiency (e.g., minimise or maximise link utilisation), and minimise operational costs of scaling up an existing network.

Even though the concept of a Digital Twin [6] seems to be established and understandable, its architecture has not been standardised for networking [1]. Some studies have attempted to define Digital Twins for specific networking environments such as Data Center Networks (DCN) [7], [8] and Industrial Internet of Things [9]. The existing works focus on the emulation/simulation of a live network as a tool for specific DTN-related tasks (network optimisation, topology discovery/management, etc.) [10], [11], [12]. DTN can be built on top of emulated network with Virtual Machines (VM) or containers where each VM or container represents a physical object entity. Emulating or simulating a large scale network to build a DTN can introduce high CapEx and OpEx (e.g., \$100 per hour for emulating just one data center). Therefore, data representation [3], [7], [8] has also been explored as a mode of constructing a DTN. However, these works focus on producing a virtual twin of a live network from the network's

The associate editor coordinating the review of this manuscript and approving it for publication was Tiago Cruz.

configuration. Therefore, such a virtual twin is static and does not reflect the dynamic changes that happen in the live network, such as traffic trends, topology change owing to physical layer trouble, etc.

The concept of DTN can be leveraged for a Software-Defined Network (SDN) [13], managing different physical and logical entities in a network, including switches, device configurations, access control lists, etc., that are modelled into a virtual representation along with their actual characteristics. The concept and role of DTN differ from those of SDN in several aspects. SDN is a network architecture to deploy and operate a network, which can be physical, virtual or a mix of both. In SDN, a behavioural change in the network is executed by the SDN Controller instructing the SDN switches about what to do to incoming packets. On the other hand, a DTN is a virtual twin of a live network that can be SDN-based, non-SDN based, or a hybrid of both. A DTN primarily works as the scalable and common pool of network state for an SDN-based network among various use cases. Consequently, the DTN enables the scalable platform for 1) contextually understanding the network behaviour, 2) verifying planned changes to the live network before its actual execution, and 3) enabling Intent Based Networking (IBN) [14], [15] by acting as the interface between the network administrator and the physical object. These platform objectives are difficult to scale because they depend on the combination of many SDN Controller services or independent applications those unlikely shares the network state effectively.

For example, consider the following use case of causality analysis: "Explain the reason for end-to-end layer 3 unreachability, say the failure of ping, between 2 hosts in an SDN-based network." Several applications must be developed on the SDN Controllers in this use case. Typically they include 1) the mapping between the IP address and MAC address as well as switch port and MAC address, 2) checking the physical end-to-end forwarding path between the ingress and egress switches, 3) checking the flow rule consistency among the SDN switches on the forwarding path, and 4) producing the response after summarising the results obtained by each application. These applications may independently communicate with the SDN Controller to repetitively obtain the network topology using the SDN Controller services. Accordingly, if even more high-level applications need access to the common information, such access can reduce the scalability of SDN Controller operation. By having the DTN of the SDN-based network, inefficient data retrieval against the SDN Controller can be drastically reduced, and the causality analysis can be scalable. Similarly, network change verification and IBN scale well with DTN. Also, any change in the DTN may or may not directly affect its live network, depending on the intention to use the DTN. At the same time, the behaviour of the SDN-based network, as a physical object, can immediately appear in the DTN.

There are several challenges when building a DTN [16], [17]. This paper addresses the challenges of 1) dynamically reflecting the live network to its virtual twin, 2) extracting the contextual insight from the raw data acquired from the live network, and 3) accurately modelling the arbitrary SDN behaviour in a live network into the virtual twin. Especially in an SDN-based network, the SDN Controller or a controller application software can arbitrarily and dynamically determine the network behaviour beyond standardised routing protocols, for example, per-flow forwarding path selection. In addition to knowing the services and features enabled in the SDN-based network, handling flow rule information is important to reflect its actual and detailed behaviour to the DTN.

This paper models the DTN of an SDN-based network as the representation of network state using a data structure called Knowledge Graph (KG) [18]. The physical objects, such as networking devices and their configurations, policies, events, etc., are represented as nodes in a KG, and the edges between the KG nodes represent the relationship in the KG (e.g. Host *isConnected* Switch). This kind of representation eliminates the need for designing database tables and writing long, complex SQL queries that are needed for relational databases, and helps in querying the required information from the KG easily and fast. The proposed DTN addresses the problem of the static nature of the existing solutions by focusing on the dynamic construction and updation of the digital twin by taking advantage of the real-time and global visibility of the network through OpenFlow messages.

The proposed DTN design also integrates templates [19], which are used to develop the DTN applications. A template is a user-friendly format for representing information about physical objects (e.g., switches, hosts, ports, links, etc.). This paper enabled the use of the Template, which allows the user to define the relationships between the objects and feed the corresponding data to the KG. In our research context, templates contain information about the network, which is used for developing the DTN applications. Templates can also work as the repositories of both queries to a KG and function to process their responses.

In summary, we make the following main contributions:

- Integrating KG to scale the data modelling of DTN handling the large data sets of network state,
- Integrating the concept of Template to achieve the flexibility of describing how a KG should be developed and accessed according to the requirements of a use case application,
- Producing insights about the performance of DTN when KG is used as its storage, and
- Publishing the Proof of Concept implementation of the proposed DTN architecture as open-source software [20].

The novelty of the proposed DTN resides in the integration of KG for DTN construction.

The rest of the paper is organized as follows. Section II is about the related works of DTN. Section III describes

TABLE 1. Comparison among the Existing and Proposed DTN Solutions.

Criteria	CrystalNet [10]	NetGraph [8]	Robotron [7]	MALT [3]	Proposed Solution
Supported Type of Live Network (Physical Object)	Non-SDN based Network	Non-SDN based Network	Non-SDN based Network	SDN and Non-SDN based Networks	SDN-based Network
Mode of Building DTN	Network Emulation	Data Representation	Data Representation	Data Representation	Data Representation
Data Modelling Method	Unspecified	Graph Database	Entity-Relationship Data Modelling	Entity-Relationship Data Modelling	Knowledge Graph
Target Use Cases	Broad Network Management	Data Center Network Management	Management of PoPs, Backbone, and Data Centers	Broad Network Management	Broad Network Management
Likeliness of Adding DTN Applications and Models	Unspecified	High (by modifying Digital Twin Model)	Low (Unlikely to change)	High (by versioning Profile)	High (by adding New Template)
Data Description Method	Details Not Specified	Vendor Specific and YANG	Template (Format Not Specified)	Details Not Specified	Template (Generic YAML)
Collected Data	List of Devices, Device Firmware, Device Configuration, Topology Description, Routing State	Device Configuration and Network State Information	Network Device Attributes and Topology Descriptions	Device Management Information	Topology Description, Flow Rules (OpenFlow), Link Statistics
Data Source	Network Device	Network Device	Network Device	Network Device and SDN Controller	SDN Controller
Data Acquisition Protocol / API	Unspecified	CLI, NETCONF, Telemetry/Syslog	CLI	Unspecified	REST API

the concept on which the paper is based, and Section IV reinforces the concept with the System Design. Section V provides the implementation overview of the proposed design. Section VI explains the evaluation result of the proposed design. Section VII discusses the limitations and future directions of the proposed DTN. Finally, Section VIII gives the conclusion of the proposed solution.

II. RELATED WORKS

A. DIGITAL TWIN USE CASES

The use cases of Digital Twin [2] are broad including the domains of healthcare, industry, smart cities, IoT, IoV, etc. [6], [21], [22], [23], [24], [25], [26]. Digital Twin for networking has also been explored in some existing work even though such examples are limited. Table 1 summarizes the features and characteristics of the existing solutions and the proposed one.

B. EMULATION BASED DTN

CrystalNet [10] emulates a production network to validate the network configuration and to identify software bugs, misconfiguration, and human errors. VMs and containers emulate routers and switches by running the firmware of the physical network devices and reproduce the control plane accurately. Though the VMs and containers are available from most major device vendors, it is computationally intensive and expensive to emulate a large scale network as a DTN. Moreover, vendor support for emulating proprietary devices may not necessarily be available. Given the feasibility of the emulation based approach, data representation based approach is more feasible especially to construct a DTN for a large network.

C. DATA REPRESENTATION BASED DTN

NetGraph [8] is defined for the DCN environment. The NetGraph provides certain functional blocks such as modelling and intelligent management of configuration and network state data, automatic configuration and translation of device and network models into configuration files, inventory search or retrieval across the whole network, and network validation. In NetGraph, a compatible configuration template must be provided to integrate a new device model from a different vendor into a DTN, which can constrain the adaptation process of the DTN to a new networking scenario.

Robotron [7] is a network management system for a large scale network, including Points of Presence, Backbone and Data Centers. The system translates high-level user intents into Entity-Relationship (ER) data models for low-level validation and safe deployment to a production network such as a Data Center Network. Its applicability to other types of networks and other use cases has not been studied broadly. Due to the use of ER data modelling and a relational database, the ad-hoc addition of a new database schema to a DTN is difficult, and the performance of handling time series data may be suboptimal.

MALT [3] provides the topology modelling suitable and abstracted in multiple layers for broad use cases in network management, including device profiles such as routers and line cards, WAN capacity planning, and topology design for a DCN that can be SDN or non-SDN based. MALT also uses ER data modelling, which leads to drawbacks similar to Robotron.

D. KNOWLEDGE GRAPH AS A DTN STORAGE

Various applications [27], [28], [29] employed Knowledge Graph for their storage purposes owing to its capability of retaining context. DEPO [30] used KG to model the information regarding Software-Defined Infrastructure, including network devices, services, compute nodes for hosting virtualized entities, etc., to enable the automated policy verification before its physical deployment. Our proposed solution also uses Knowledge Graph to model SDN's broad data whose contexts and their efficient processing are important to enable the basic features and use cases of DTN.

III. DTN FOR SDN

A. SDN AS A PHYSICAL OBJECT

1) PHYSICAL INFRASTRUCTURE

The proposed DTN is defined on the SDN infrastructure, which consists of an SDN Controller, REST services offered by the SDN Controller, and the data plane. The SDN involves broad properties to be considered for its data-driven representation originating from switches, hosts, links, flow rules, etc. The main component of SDN is the controller that is responsible for the management and centralized control of the networking entities.

In this paper, DTN is developed on the foundations of an SDN-based network and its various services provided by the SDN Controller for managing inventory of hosts, links, handling packet-in, querying flow rules and statistics, computing paths in the network, providing a stream of diagnostic information, etc. DTN builds the virtual twin based on the raw data extraction and collection from SDN Controllers using its different services.

2) SDN SERVICES

Fig. 1 illustrates the various SDN services [31] for modelling the data for DTN construction.

Topology Service provides topological information, including inter-switch connectivity, end devices, link status, port status, etc. Network Statistics are used to build statistical data models based on port statistics, flow table statistics, load statistics of the links, etc. Flow Rule information is very important because it reflects decisions made by the SDN Controller software. The proposed DTN architecture takes Flow Rules as input to form the virtual twin. Access to the raw data provided by these services is available between DTN Manager and SDN Controller through the REST APIs.

B. DATA DRIVEN DTN AS A VIRTUAL TWIN

1) OVERVIEW

This paper pursues a data-driven approach to construct a virtual twin of an SDN-based network. Our architecture is designed to achieve the following features:

1) The data modelling of the contextual representation among the physical objects in the SDN-based network.
2) The fine granularity and the intuitiveness of describing and extracting the contextual representation.

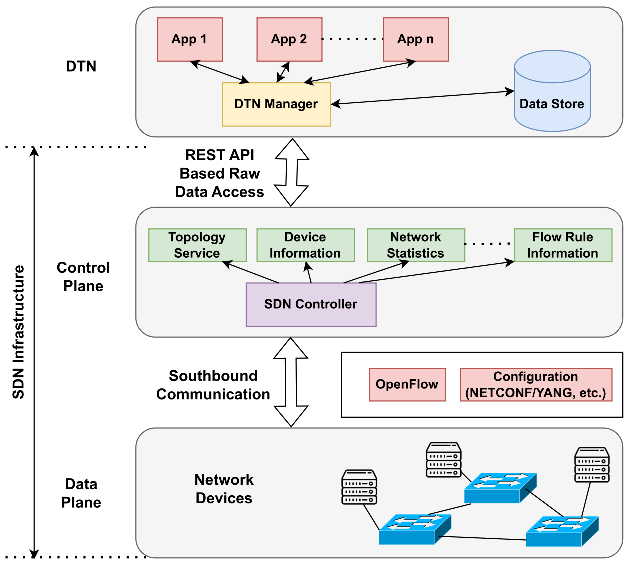


FIGURE 1. DTN consuming SDN Services.

3) The scalability and performance to tolerate the frequent update of the virtual twin on the basis of the dynamic nature of the SDN-based network in the real-world deployment.

2) EMPLOYING KNOWLEDGE GRAPH AND TEMPLATE

Knowledge Graph is the key choice for the main storage of contextual information for the virtual twin creation and maintenance in the proposed architecture. KG is useful to draw inferences about a fact or an event in the network that can be represented as numerous relationships between the nodes in the graph. Such functions enable the causality analysis of an incident, even though it could happen due to multiple factors, as well as the pre-validation of a planned reconfiguration in the network. Efficiency and response time are very important when querying a database to extract contextual information. Knowledge Graph using a cypher query is better in both aspects than a relational database using SQL that may need to join multiple queries across multiple tables to respond with the target information especially when the amount of information kept in the storage is very large.

The proposed approach extracts the curated set of contextual information out of the network state collected from an SDN Controller. A KG stores the information so that the collected data from the SDN Controller is synaptically linked and forms the virtual twin representing the original network.

The concept of Template makes the context definition and extraction affordable for the network administrator. The template is extended to mention a relationship of physical objects, and such a relationship makes the data feed to KGs intuitive for a network administrator, who is supposed to be benefited from the DTN. The following subsections provide the details of the interplay of KG, Template, SDN Controller, and the DTN Manager, which integrates these components.

3) WORKFLOW

As shown in Fig. 2, the workflow process of the DTN operation can be briefly described by the following three steps.

1) An application template is deployed to DTN Manager. The application template defines the scope of entities, mechanisms, and policy functions, whose details are explained in §IV-A1.
2) DTN Manager processes the application templates and builds the virtual twin using KG based on the data collected from the SDN Controller.
3) The DTN application executes a query to the virtual twin and returns the response as the result of taking intended action to DTN.

We look more at the internal operations and components of the DTN system that enable these workflows in §IV-B.

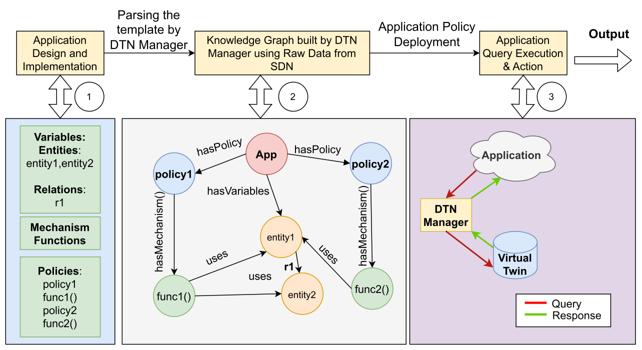


FIGURE 2. DTN Workflow.

IV. SYSTEM DESIGN

This section describes the proposed DTN architecture, the system design, and the data modelling process. The architecture of DTN is illustrated in Fig. 3.

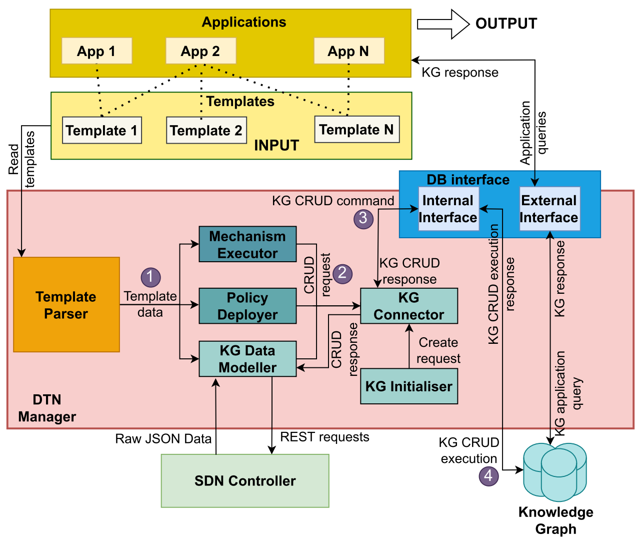


FIGURE 3. DTN System Diagram.

The DTN architecture builds a KG based on the templates defined by the network administrator. The core part of the DTN framework is the DTN Manager. The input to the DTN Manager is a template which describes the intentions of each application, and the output is the response against the application queries, which are executed to KG through the DTN Manager. Typically, a template developer is either a network administrator or a domain expert who is not a programmer. An application developer is a programmer who understands the templates defined by the network administrator or the domain expert and writes the application based on the templates.

A. DTN APPLICATIONS

1) DEFINING A TEMPLATE FOR AN APPLICATION

A template works as a human editable interface between the network administrator and the DTN manager so that the network administrator can describe how the DTN application (for e.g., end-to-end / host level reachability check in the network) can be projected to the corresponding data representation produced by the proposed DTN architecture. An Application contains an intent logic to be performed using the data retrieved from KG. An intent logic is a combination of relevant parameters for a DTN use case application and the corresponding algorithm to process them. However, such an intent logic must describe entities, properties, and relations among them to be kept in the corresponding KG. The proposed system uses the concept of a Template, which is a skeleton specification standard for describing the intention of the application behaviour and its execution process.

An intent logic is defined in the form of one or more sets of policies in the template. Except for the fundamental parameters used in the DTN Manager, which are about the network topology, most intent logic will likely be set in a template but outside the DTN Manager.

A template is described using YAML [32] format that contains three sections consisting of the variables, mechanisms, and policies. An example template for one of the built-in applications of the proposed DTN, Physical Reachability Check, is shown in Listing 1.

a: VARIABLES

The variables contain entities and relationships. **Entities** define networking objects such as switch, host, flow table, flow rule, etc. Each entity is specified along with its unique set of properties; for example, a host entity has macAddress, and a switch entity has dpid, firmware, and protocol.

A relation is defined through mapping a pair of entities and also contains certain unique properties. An important consideration is that the template should be generic to support different types of SDN Controllers, which have different syntaxes to respond. The generalization and inclusiveness of template description will allow the same template to support multiple SDN Controllers. The balance of generalization and inclusiveness is important so that the template will not become too detailed and large in order to avoid constructing unnecessarily large KG.

Listing 1. Example Template of Physical Reachability Check Application

```
variables:
    entities:
        Switch:
            - dpid
            - firmware
            - protocol
        Host:
            - macAddress

        ForwardingDevice:

        Object:

    relationships:
        - map:
            - Switch
            - Switch
            - isConnected
        - map:
            - Host
            - Switch
            - isConnected
            Properties:
                - port
        - map:
            - Switch
            - ForwardingDevice
            - isA
        - map:
            - ForwardingDevice
            - Object
            - isA

mechanisms:
    script: 'topology'

policies:
    - policy:
        name: 'CheckReachabilityInTopology'
        deploy : 'getTopoReachability'

    - policy:
        name: '
            CountSwitchHopsInPhysicalShortestPath'
        deploy : 'getHopCounts'
```

In the template example, the **relationships** define the relationship between entities, for example, *"isConnected"* between Switch entities, *"isConnected"* between Switch and Host entities, and *"isA"* between Switch and Forwarding Device, and *"isA"* between Forwarding Device and Object. The two switches and hosts form the entities for the simple topology shown in Fig. 4. The properties of the entities and relationships are also specified in the template. The switch entity has the properties dpid, firmware, and protocol, while the Host entity has the property macAddress. The relationship *"isConnected"* between Switch and Host entities has a single property: port. The links between the Switches and Hosts in Fig. 4 form the relationship. The entities form the nodes in the KG, and the relationships between the entities are represented as edges between them.

b: POLICIES

The policies define the unique or specific set of intent-based tasks that an application executes. A template can contain multiple policies. Each policy works as a function 1) to construct a Knowledge Graph, and 2) to extract the contextual information from the KG through querying it. The response to a query can be directly used as a response from the application or be parsed further to produce a more complex answer in the application. However, each of the contexts itself ("A *is* B", "C *has* D", etc.) must be defined in the template respectively. Also, each policy can have a set of subtasks when the policy can not be executed as a single task. There has to be at least one policy defined in the template for an application to query the data from the KG. A subtask can also trigger a query to a KG.

c: MECHANISMS

The mechanism defines the set of all subtasks to execute various intent logics for an application. The mechanism allows templates, policies, and KGs to be reusable for other applications and avoid developing the same task using the same data.

The example template has a certain policy *"CheckReachabilityInTopology"*, having a subtask *"getTopoReachability()"* for getting the end-end reachability from the KG based on the topological details. On the other hand, another policy, *"CountSwitchHopsInPhysicalShortestPath"* has the subtask *"getHopCounts()"* that gets the end-end hop counts for the reachable end devices. The set of subtasks *"getTopoReachability()"* and *"getHopCounts()"* implement the intent logic of the application. We discuss the implementation of the Physical Reachability Check application in the next sections.

The mapping of templates and applications can vary and be dynamic. A single application can use multiple templates for defining its use cases; in another scenario, multiple applications can also use and share a common template. The flexibility of the combination of applications, templates and KGs can be leveraged for developing more complex applications.

2) BUILT-IN APPLICATIONS

This work presents three basic built-in applications that run on the proposed DTN to show examples of end-to-end use and working of the proposed DTN architecture.

a: TOPOLOGY DUMP APPLICATION

This application gets the mapping information concerning the connected ports between a host and a switch and between switches in the network topology. The application contains two different policies. First, the *"DumpHostToSwitchMap"* policy retrieves information about the specific ports connecting hosts and switches in the network. For the topology shown in Fig. 4, this policy returns the ports to which the two Hosts are connected to each of the two Switches, which in this

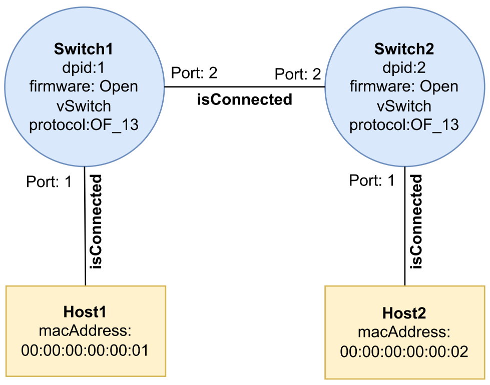


FIGURE 4. **Simple topology with two switches and hosts.**

case is port number 1. The second policy, *"DumpSwitch-ToSwitchMap"*, gets the information about the switches with the port information for the pair of connected switches in the network. For the pair Switch1 and Switch2, it returns port number 2.

b: PHYSICAL REACHABILITY CHECK APPLICATION

The Physical Reachability Check application provides the use case of identifying physical reachability from the topological aspect of the network and also retrieves the hop counts for a given pair of hosts if there exists a path between the pair of hosts. The KG built for the Topology Dump application is shared with this application. This application has two policies defined; the *"CheckReachabilityInTopology"* policy checks for the host-to-host path reachability based on the shortest path first approach for each pair of source and destination hosts in the network and returns the binary response as reachable or not reachable. This policy returns the response as reachable for both the Hosts, i.e., Host1 and Host2 for the topology in Fig. 4. The second policy, *"CountSwitchHopsIn-PhysicalShortestPath"*, gets the hop count between the hosts in the network. For the topology in Fig. 4, this policy returns the hop count as 1 between Host1 and Host2.

c: FLOW RULE REACHABILITY CHECK APPLICATION

This application uses the flow rule information defined in the switches to check for the reachability between all the pairs of hosts in the network and returns the hop count between the hosts only if they are reachable based on the flow rules. A source host is reachable to a certain destination host in terms of flow rules only if the intermediary switches have flow rules that allow the packets from the source host to flow to the destination host. The policy *"CheckReach-abilityByFlowRule"* defines the logic of a flow rule-based reachability check. If both the switches in Fig. 4 have the flow rule instruction installed to forward the packet coming from the two hosts, this policy will give a response as reachable for both hosts. The application has another policy *"CountSwitchHopsInForwardingPath"*, which retrieves the hop count between the reachable pair of hosts in the network based on the flow rules. This policy returns the hop count as 1 between Host1 and Host2 if none of the two switches has a flow rule instruction written in it to block the packets coming from the two hosts.

B. DTN MANAGER

The DTN Manager is an ensemble of several components that provides the data definition, collection, and data management functionalities. The DTN Manager takes a template and the OpenFlow related data as inputs. An essential component of the DTN Manager is the use of a Knowledge Graph as the DTN storage. A KG stores the entities, relationships, and properties defined in a template as well as OpenFlow related data collected from an SDN Controller. The network topology is immediately reflected to the Knowledge Graph as long as the DTN Manager is active, and the other OpenFlow information is collected according to a template on demand. The output of the DTN Manager is a query response from a Knowledge Graph to a query issued by the applications.

1) TEMPLATE PARSER

The input templates to the DTN Manager for building the KG are parsed by the Template Parser. Once the template is parsed into the form of a key-value store by the Template Parser, such information is passed to the KG Data Modeller, Mechanism Executor, and Policy Deployer.

2) KG INITIALISER AND KG CONNECTOR

KG Initialiser executes the sanity check of the KG of the DTN Manager, including the KG availability and the connection establishment. KG Connector executes create, read, update and delete (CRUD) operations to the KG. KG Connector handles all the queries to KG from the other modules of DTN Manager and routes back the response from the KG to the respective modules.

3) POLICY DEPLOYER AND MECHANISM EXECUTOR

The policies and mechanisms defined in the template are passed to the KG Connector by the Policy Deployer and the Mechanism Executor modules for updating the KG. Policy Deployer receives the information of a task and a subtask described as a policy in a template. Policy Deployer adds the incoming information to the knowledge graph. The Mechanism Executor maintains the mapping between the template and the policies defined. A template can be arbitrarily updated by the network administrator by adding, updating, or deleting a policy. The Mechanism Executor always keeps the list of the latest available policies for the corresponding template. The combination of Policy Deployer and Mechanism Executor helps to make the life cycle management of a policy to be flexible against on-demand changes so that a single policy can be continuously updated and used rather than create a new template for a similar application.

4) KG DATA MODELLER

The KG Data Modeller refers to the variable section in the key-value store input from the Template Parser and collects the data from the SDN Controller using various REST APIs. The template can describe any predefined parameter, including topology, links, devices, flow rules, port status, services, and other metrics representing the facts, live network configurations, and state-level information. Expecting that different types of SDN Controllers can have different syntaxes to respond to the REST query, the unification of input data to the KG Connector is also important so that multiple types of SDN Controllers can be represented in the KG using a common template.

After receiving the queried information from the SDN Controller, the KG Data Modeller creates and updates the KG to construct the DTN using input data. KG operations by KG Data Modeller continue until the KG is completely built.

C. KNOWLEDGE GRAPHS

A KG is a large network representing various entities, the relations between them, and their properties in the form of a graph data structure representing real-time semantics. The information in a KG is represented in the format such as *(entityA relationship entityB)*, where the entities form the nodes in KG and the relationship is represented by an annotated edge between the entity nodes (the term node refers to a vertex in the KG unless otherwise stated). Entities can contain multiple relationships between them represented by the multiple annotated edges in the KG. Each node and edge can also hold properties specific to the entities and the relationships.

Running the user queries/intents and getting the desired results from the raw data collected requires a data modelling standard. The DTN uses KG to represent the raw data. KG has the ability to map entities and existing relations and provide scalability for searching and context representation. Using a relational database instead of KG would have limited the capabilities of scalable search and context representation we wanted for the DTN [33], [34], [35]. Querying a relational database for various use cases using joins and nested queries would be cumbersome and time-consuming compared to querying a KG based on the relationships and properties modelled in it.

1) DATA MODELLING USING KNOWLEDGE GRAPHS

Consider a switch containing a flow table that contains certain flow rules. The switch, flow table, and flow rule represent the real-time entities that exist in an SDN network. This information can be represented in a KG as

```
Switch A <hasComponent> FlowTable \
<hasComponent> FlowRule
```

where the `<hasComponent>` defines the kind of relationship existing among them. A visualisation of such a KG is shown in the following Fig. 5.

Consider another example where there are hosts connected to switches in a network. The host and switch represent the real-time entities. This kind of information in a KG can be represented as

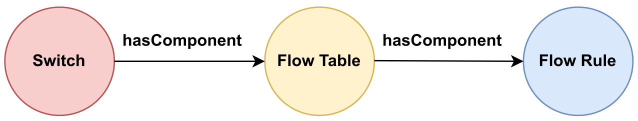


FIGURE 5. A Knowledge Graph for Switch, Flow Table and Flow Rule.

```
Host <isConnected> Switch
```

where the `<isConnected>` represents the relationship between the host and switch entities, as shown in Fig. 6.

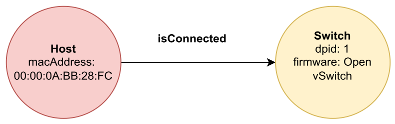


FIGURE 6. Host and Switch Knowledge Graph.

This kind of information modelling gives the KG the ability for better context representation and meaningfulness. An important consideration here is that DTN maintains the KG's current network state by constantly updating the KG after a fixed time interval. Alternatively, the DTN can also use timestamp annotations of the entities and relations in the KG models to retain the current network state.

2) QUERYING THE KG

Querying is an important process for the DTN system for its interaction with the KG and running of the deployed applications after collecting raw data and building the KG-based virtual representation of the entire data. We can classify the queries of the DTN system into two types: DTN manager queries and application or intent queries. The DTN uses the standard Neo4j [36] query mechanism of "CYPHER" [37]. The CYPHER query language defined by Neo4j provides a rich set of clauses and constructs for various querying capabilities.

a: DTN MANAGER QUERIES

These are the queries used by the DTN Manager for creating and deleting entities and relationships periodically throughout its runtime to completely build the KG for an application based on the input template. These queries are issued to the KG by the KG Connector module through the internal interface. The two crucial cypher query clauses include the "CREATE" and "DELETE" for performing the specific operations. Consider an example of creating a particular "Switch" entity with "dpid" as "1". This is represented in the cypher as

```
CREATE (n:Switch{dpid:"1"})
```

This query creates a switch entity node with the specified "dpid" property where "n" is a temporary variable to store the reference to the created node.

b: APPLICATION/INTENT QUERIES

Applications run the policy for performing the user intents based on data retrieved from the KG. An application issues a read-only cypher query to the corresponding KG through the external interface. The MATCH and RETURN clauses are essential and the most frequently used to search for the specified entity and return it through the subgraph matching process. The subgraph matching process is the problem of finding all subgraphs of the target graph that are isomorphic to the given query graph. In essence, the application queries are read-only and can not modify the KG.

Consider the following example of a query to a KG which consists of a set of entities such as hosts, which are connected to switch entities through a relationship "isConnected" as shown in Fig. 6. An example query following the syntax of the Neo4j cypher standard to fetch a particular Host with mac address "00:00:0A:BB:28:FC" connected to a Switch with dpid "1" is represented as follows.

```
MATCH                                      \
(h:Host{macAddress:"00:00:0A:BB:28:FC"})\
-[r:isConnected]->(s:Switch{dpid:"1"})   \
RETURN  h,s
```

The above query returns the following response.

h	s
{"macAddress": "00:00:0A:BB:28:FC"}	{"dpid":"1","firmware":"Open vSwitch"}

The temporary variables "h" and "s" are references to the "Host" and "Switch" nodes, respectively. Also, note that the "macAddress" and "dpid" are examples of the properties specific to the particular entities "Host" and "Switch" respectively.

Consider another example to fetch the properties of a specific switch entity.

```
MATCH(s: Switch{dpid:"1"})    \
RETURN properties(s)
```

The above query returns all the stored and specified properties in the template for the specific "Switch" entity referenced by the temporary variable "s".

properties(s)
{"dpid":"1","firmware":"Open vSwitch"}

For the example KG in Fig. 6, the query returns the properties dpid (with value 1) and firmware (with value Open vSwitch) associated with the Switch node.

The application developers specify such queries based on a template and a policy to be sent to the KG according to the application developers' intent.

V. IMPLEMENTATION

A. TOOLS AND SPECIFICATIONS

The implementation of DTN uses tools which form its basic components for its functioning as summarized in Table 2. The complete DTN system consists of approximately 2000 lines of code [20].

TABLE 2. Tools & Specifications.

Type of Software	Name of Software	Version
SDN Controller	ONOS	2.5.1
Network Emulator	Mininet	2.3.0
Knowledge Graph	Neo4j	1.4.9
Programming Language	Python	3.9.2

B. KG BUILDING PHASE

KG construction approaches: The DTN Manager builds the KG based on the entities and relationships described in the template of an application. There is a trade-off in terms of memory consumption and response time between building a dedicated KG for an individual application and building a common KG to cover multiple applications. Building an exclusive (application-specific) KG is beneficial if the KG is small enough with fewer nodes and relationships. However, each application will not function until its KG is built.

Another approach is to build a single KG that can be shared by multiple applications and works as a superset of KGs if those were built by each of the applications individually. In such a case, the cost of building multiple KGs can be reduced, especially when most parts of KGs are common among the applications. However, constructing and using a common KG may involve more memory consumption and longer response time if the KG becomes large. In this paper, the Topology Dump and Physical Reachability Check applications (§IV-A2) share a common KG and is illustrated in Fig. 7a. The Flow Rule Reachability Check application builds a separate KG needing a totally different KG from the other applications and is represented in Fig. 7b.

If a common KG can effectively aggregate the overlapping nodes and relationships, the efficiency of the KG handling can be even better. However, this paper does not aggregate multiple KGs because such an attempt needs an appropriate aggregation algorithm, which is beyond the scope of this research.

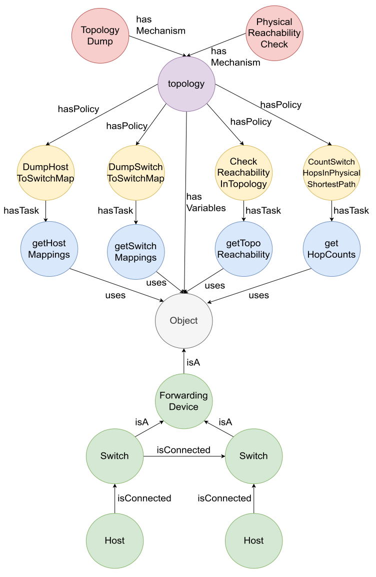
Each KG has nodes that define the entities, the policies, and its mechanism that contain the logic to construct, execute a query and process the response to extract the desired information. An important point worth noting here is that each application has its specific KG requirement, and the size of the KG can vary accordingly based on the number of entities and their relations specified in the template and the network topology.

C. IMPLEMENTATION OF DTN APPLICATIONS

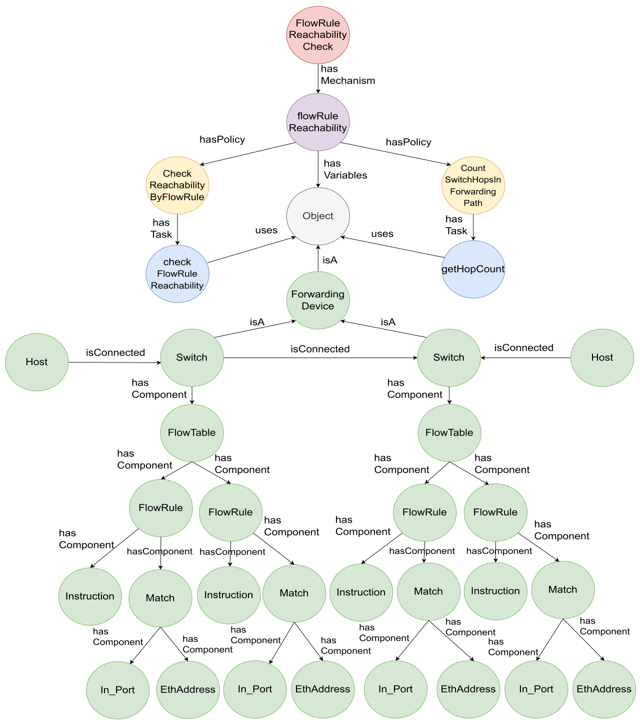
Applications run the required cypher queries on the KG built for their desired intent and process the response for parsing its outputs. For example, the Topology Dump application defines two policies, 'DumpHostToSwitchMap' policy uses the function 'getHostMappings()', which uses the MATCH queries to retrieve the information between all connected

TABLE 3. Number of variables, mechanisms, and policies in the template and number of nodes in KG for the 3 topologies.

Application	Variables	Mechanisms	Policies	Number of nodes in KG Linear	Abilene	Geant
Topology Dump Physical Reachability Check	6	1	2	13	29	69
Flow Rule Reachability Check	20	1	2	134	2146	20906



(a) Topology Dump and Physical Reachability Check



(b) Flow Rule Reachability Check

FIGURE 7. Knowledge Graphs for different applications.

host and switch nodes from the KG based on the subgraph matching. The process is repeated for all the pairs of connected hosts and switches in the topology. The second policy, 'DumpSwitchToSwitchMap', uses the function 'getSwitchMappings()' to perform a MATCH query to search on all the connected switch nodes in the KG and retrieve the port information from the query response. The other applications are implemented similarly.

VI. EVALUATION

The proposed DTN architecture was evaluated mainly by measuring the time duration for building a KG and querying a KG by executing different DTN applications in different network topologies.

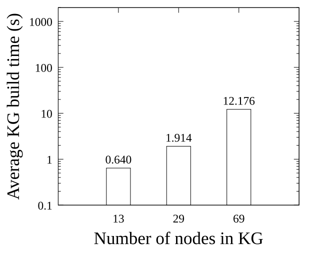
As summarised in Table 3, the combinations of an application and a network topology introduce different numbers of variables, mechanisms, and policies in the corresponding template as well as different sizes of KG. This evaluation used the DTN applications that were described in the previous sections: Topology Dump, Physical Reachability Check, and Flow Rule Reachability Check. The three network topologies were used: Linear topology with 3 switches, Abilene [38] network with 11 switches, and Geant [39] network with 31 switches where 1 host is attached to each switch. Note that Topology Dump and Physical Reachability Check share the same template and the KG. The evaluation was performed on a workstation with 32 GB RAM capacity, Intel Xeon(R) W-2133 3.6 GHz, 12 core CPUs, and 2 threads per core.

A. KG BUILD TIME

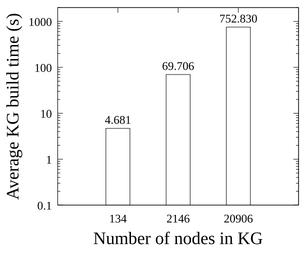
The time taken to build a KG was measured against the varying size of KG in terms of the number of nodes due to the combination of application and network topology. The total build time is measured as the time elapsed between reading a template at DTN Manager and completing the construction of the corresponding KG inclusive of the communication delay with the SDN Controller through REST APIs.

Fig. 8a and 8b show the average KG build time of five attempts using the two applications: Physical Reachability Check and Flow Rule Reachability Check. In both applications, the KG build time becomes longer as the size of the KG increases. The KG build time ranged in the scale of seconds and minutes in this experiment. These results indicate that ad-hoc construction of a KG may not be realistic, and the reuse of an existing KG can reduce the delay to deploy a new DTN application if available. Also, note that the KG build times for Topology Dump and Physical Reachability Check were identical due to the shared KG.

While building the complete KG is a one-time process and takes time, the time taken for adding a new node or updating



(a) Physical Reachability Check



(b) Flow Rule Reachability Check

FIGURE 8. Average KG build time vs Number of nodes in KG.

an existing node in the KG is very short. In our evaluation setup, the average time to add a new node was 18 ms with little deviation for all the KGs in all three topologies.

B. QUERY RESPONSE TIME OF DTN MANAGER AND APPLICATION QUERIES

The query response time is defined as the time taken for the response of a cypher query to the corresponding KG to be returned. Fig. 9 compares the query response time for the DTN Manager queries and the application queries against different applications involving the different numbers of nodes in a KG.

The query response time for application queries is shorter compared to the DTN Manager queries because application queries involve only MATCH read operation, while the DTN Manager queries involve CREATE and UPDATE operations which are basically the read and write operations in the KG. From Fig. 9a and Fig. 9b, it is observed that the query response time is almost the same since the two graphs correspond to the two applications that share the same KG. In Fig. 9c, the average query response time for the application queries is 5 ms for approximately 21 thousand nodes in the KG. This implies that the execution of an application in a given size of KG is realistic for practical use.

C. IMPACT OF QUERY DEPTH

In this evaluation item measures the impact of Query Depth on the query response time. Query Depth is defined as the number of nodes involved in a single cypher query. For example, the following cypher query

```
MATCH (h2:Host{mac:'8E:23:04:6F:48:2F'})      \
-[r2:isConnected]->(s2:Switch)                \
return s2,properties(r2)
```

contains Host and Switch as nodes in the KG. This query is used to find the switch to which the host with MAC address “8E:23:04:6F:48:2F” is connected and has a query depth of 2 since 2 nodes are mentioned in this query. Consider another cypher query as follows

```
MATCH (s1:Switch{id:'00000000002'})-          \
[hc:hasComponent*]->(f:FlowTable)-            \
[:hasComponent]->(f1:FlowRule)-               \
[:hasComponent]->(m1:Match)-                  \
[:hasComponent]->(e1:EthAddress)              \
where e1.dst='8E:23:04:6F:48:2F'              \
and e1.src='6A:05:72:74:85:42'                \
return f1,m1,e1
```

This query is used to check whether there is a matching flow rule in the switch with id ’00000000002’ for the given pair of source and destination MAC addresses. The nodes involved in this query are Switch, FlowTable, FlowRule, Match and EthAddress. Hence the query depth of this cypher query is 5.

The query response time is measured against the query depth for the three applications executed in the three topologies. Fig. 10 depicts that the query response time increases as the query depth or the number of nodes in the KG increases. Fig. 10c exhibits that the query response time is approximately 11 ms when the number of nodes in KG is around 21 thousand, and the query depth is 5.

D. APPLICATION EXECUTION TIME

In this evaluation metric, we measured the time taken to completely execute an application against the total number of queries executed for an application. The application execution time is the time duration to execute an application involving a certain number of queries executed to a corresponding KG after the DTN Manager completely builds its KG.

The application execution time is the total sum of the time that the policies took to get executed for each application. In Topology Dump, 2 policies, *DumpHostToSwitchMap* and *DumpSwitchToSwitchMap* policies are executed. Similarly, *CheckReachabilityInTopology* and *CountSwitchHopsInPhysicalShortestPath* policies are executed in Physical Reachability Check, and *CheckReachabilityByFlowRule* and *CountSwitchHopsInForwardingPath* policies are executed in Flow Rule Reachability Check respectively.

Fig. 11 shows that the application execution time increases as the total number of queries executed per application increases. The three different applications exhibit different average execution times, which depend on how each application works even though the same KG may be shared. In Fig. 11c, the average application execution time for Flow Rule Reachability Check in Geant topology is 141.425 seconds, which involves approximately 31 thousand query executions, including the ones with a query depth of 5. Also, even after the necessary responses are returned, they have to be processed to return the application response by checking whether the two entities in the network have a consistent forwarding path in between.

E. ANALYSIS OF QUERY THROUGHPUT OF DTN MANAGER AND APPLICATION QUERIES FOR EVEN LARGER NETWORKS

The query throughput is defined as the number of queries executed against a KG in a certain length of time. A script, which simulates a large network, is executed to measure the query throughput for very large KGs instead of emulating the live network using mininet because of resource limitations. The script is implemented for the Flow Rule Reachability Check application in a k-fat tree topology for simulating a simple data center network. The script takes the value of

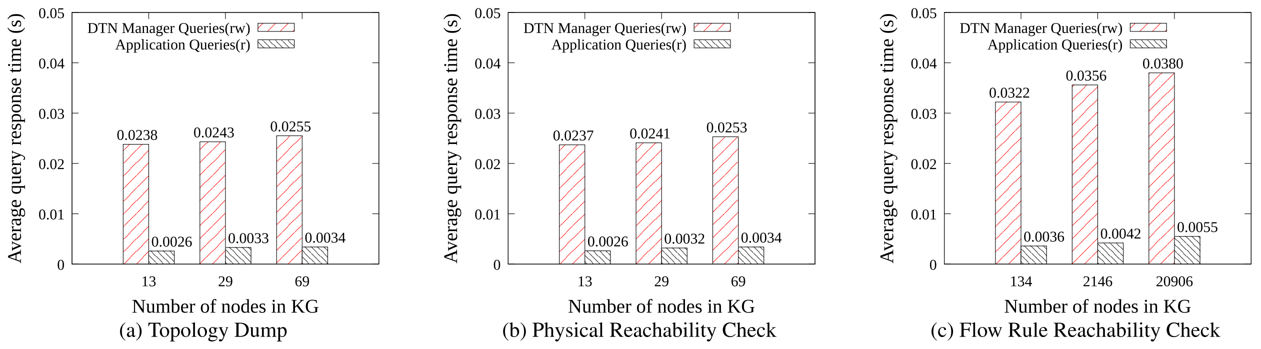


FIGURE 9. Average query response time vs Number of nodes in KG.

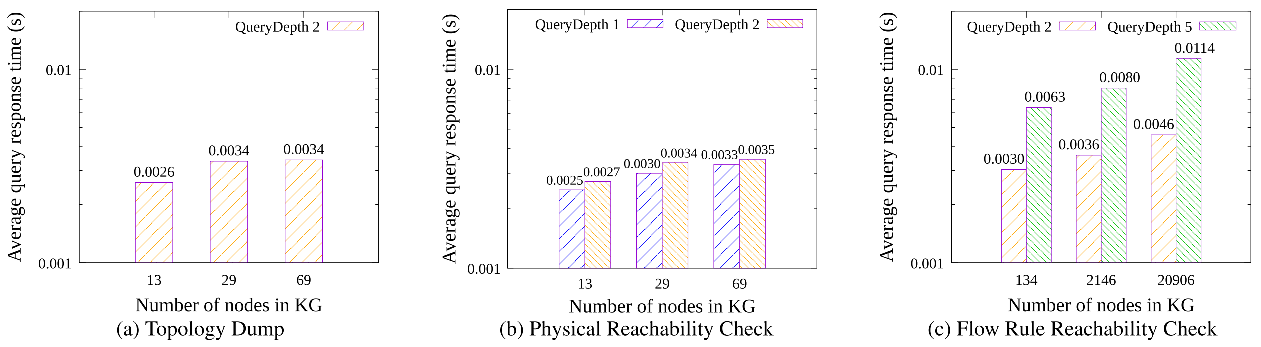


FIGURE 10. Average query response time w.r.t Query Depth.

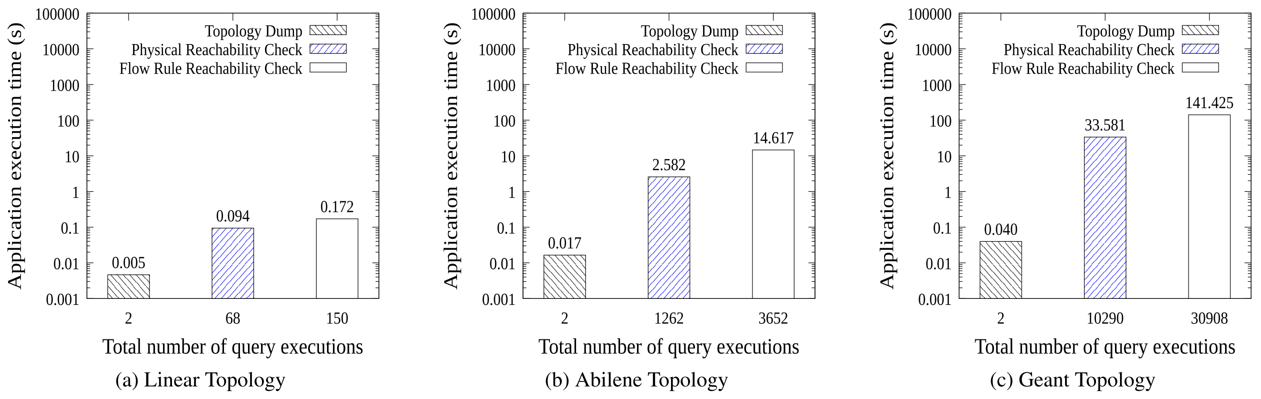


FIGURE 11. Application execution time vs Total number of query executions.

“k” as input to generate the number of nodes in the KG on the scale of one thousand, ten thousand, and hundred thousand. In a k-fat tree topology, for a given value of k, there are $(k/2)^2$ core switches, $(k^2/2)$ edge switches, $(k^2/2)$ aggregation switches, and $(k^3/4)$ servers. The script records the number of queries issued per second during its execution. This experiment measures 1) the throughput of the DTN Manager queries (CREATE and MATCH queries) and 2) the query throughput of application queries (MATCH queries).

Fig. 12a shows the average throughput, and Fig. 12b shows the average response time of the DTN Manager and Application queries for the nodes in KG with the scale of one thousand, ten thousand, and hundred thousand for the different values of k=4, 6, and 8 respectively of the k-fat tree topology.

In Fig. 12a, the throughput of the DTN Manager queries is less than that of the application queries since the DTN Manager query involves write operation as well, whereas application queries are read-only. Also, the average throughputs of both DTN Manager queries and application queries decrease with the increase in the number of nodes in the corresponding KG since query response time is largely proportional to the number of nodes in KG. However, as seen in both Fig. 12a and Fig. 12b, neither the query throughput nor the response time degrades drastically even when the size of the KG increases to a different scale order. This fact

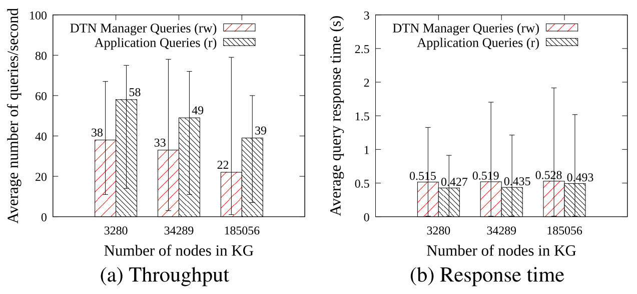


FIGURE 12. **Average query throughput and average query response time w.r.t number of nodes in KG.**

emphasizes that KG is good as a storage for the virtual twin of a large network.

F. KEY TAKEAWAYS

Response time in various angles is important to measure the performance and scalability of the DTN architecture. The experiment was conducted using the PoC implementation with the built-in applications of the proposed DTN architecture. To summarize the results, the query response time to a KG is largely proportional to the size of the KG and the query depth. Hence, the size of the KG and query depth should be considered when a KG is constructed to maintain the query response time to be reasonably short.

In addition, reading or adding an entity to a KG can be done with a reasonably short response time compared with the construction of a KG. Considering the importance of reflecting the dynamic changes in the live network to its virtual twin, using KG for the DTN purpose is realistic and scalable, given the result of the query throughput experiment.

VII. DISCUSSION ON LIMITATIONS AND FUTURE SCOPES

A. KG BUILD TIME AND SCALABILITY

The KG build time increases as the network size increases and can affect the setup process of a DTN for a very large network. Currently, the proposed architecture uses Neo4j, which is proprietary software, as the implementation of KG. Therefore, optimizing the KG build time in the current implementation can be unrealistic. On the other hand, partitioning a large network topology into smaller ones may help to constrain the size of a network to reduce the KG build time or parallelise the KG construction processes. However, the way to partition a network can be computationally complex for minimising the total time and resource consumption of such a KG construction approach. Also, the maintenance of multiple KGs can introduce additional operational overhead as well as issues of data security and consistency.

1) ENABLING BROADER DATA SOURCES AND PROTOCOLS

The current design of DTN uses the data collected from the ONOS-based SDN Controller via the REST API. If a high-resolution virtual twin is desired, much broader data should be collected from the live network and supporting broader types of SDN Controllers is important. Also, KG Data Modeller can be extended to support more data sources and information exchange protocols such as SNMP, Netconf, etc.

2) SECURITY AND PRIVACY

A DTN inherits the logic, parameter, network topology, etc., that the SDN Controller of a live network maintains. While the security of the SDN Controller has already been discussed [40], [41], [42], the preparedness of security of the DTN architecture needs more detailed discussions. Expecting any input from an SDN Controller may be malicious, an anomaly of input data from the SDN Controller should be detected instead of unconditionally incorporating such data to the DTN storage. In addition, the DTN architecture can naturally collect the data from user devices connecting to the live network. User privacy should still be properly preserved, but it can also be a challenge when the DTN is used for enabling the virtual twin of an IoT for home, mobility, healthcare, etc.

3) SDN ACTUATION FROM DTN

The built-in features of DTN implementation work in the read-only mode. However, a DTN application can change the network behaviour by interacting with the SDN Controller through the REST API. However, actuating an SDN-based network through a DTN application should be executed safely. Network state verification before and after the SDN actuation should be done in the SDN. Also, potential inconsistencies in the decision making logic between the SDN Controller and the DTN Manager need to be mitigated or eliminated without disturbing the ongoing network operation. The issue of DTN-SDN inconsistency will likely happen to the data representation based DTN approaches that do not reproduce another network as a virtual twin.

4) SCOPE OF AI/ML INTEGRATION

Applying Artificial Intelligence (AI) / Machine learning (ML) techniques on the data stored in KG is a very interesting direction for various use cases, including predictive analysis and causality analysis of broad phenomena in a live network as well as optimisation of performance and resource usage in the network [43], [44], [45]. Together with network verification and actuation, AI/ML can benefit the network operation to behave optimally with cost-effectiveness.

VIII. CONCLUSION

This paper proposed a data representation based DTN architecture for SDN-based networks integrating Knowledge Graph (KG) as its data storage together with the concept of Template. The use of KG is highly promising to set and retrieve the contextual understanding of the state of the live network in a scalable manner, which is important to synchronize DTN and SDN with high accuracy and frequency, given the dynamic nature of a live network. The concept of Template introduces flexibility to define the entities and intent logic about the contextual information in the live network and determines how the KG needs to be built. The PoC implementation of the proposed architecture integrated an ONOS-based SDN Controller and Neo4j-based KG with the built-in DTN applications for practical use. In the evaluation, the PoC implementation exhibited reasonably good performance in terms of the query response time and throughput and produced several insights, including the limitations and potential challenges as discussed in §VII. As a future work, the key research directions of our interest are the safe actuation of SDN from DTN application, the integration of AI/ML to the DTN Manager, the security and privacy in operating a DTN, and the further improvement of DTN scalability.

REFERENCES

[1] C. Zhou, H. Yang, X. Duan, D. Lopez, A. Pastor, Q. Wu, M. Boucadair, and C. Jacquenet, "Concepts of digital twin network," Internet-Draft, Draft-ZHOU-NMRG-DigitalTwin-Network-Concepts-03, Internet Eng. Task Force, 2021, p. 15. [Online]. Available: https://datatracker.ietf.org/doc/draft-zhou-nmrg-digitaltwin-network-concepts/03/

[2] Y. Wu, K. Zhang, and Y. Zhang, "Digital twin networks: A survey," *IEEE Internet Things J.*, vol. 8, no. 18, pp. 13789–13804, Sep. 2021.

[3] C. Jeffrey Mogul, D. Goricanec, M. Pool, A. Shaikh, D. Turk, B. Koley, and X. Zhao, "Experiences with modeling network topologies at multiple levels of abstraction," in *Proc. 17th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, Santa Clara, CA, USA: USENIX Association, Feb. 2020, pp. 403–418.

[4] J. Deng, Q. Zheng, G. Liu, J. Bai, K. Tian, C. Sun, Y. Yan, and Y. Liu, "A digital twin approach for self-optimization of mobile networks," in *Proc. IEEE Wireless Commun. Netw. Conf. Workshops (WCNCW)*, Mar. 2021, pp. 1–6.

[5] P. Almasan, M. F. Galmés, J. Paillisse, J. Suárez-Varela, D. Perino, R. D. López, A. A. P. Perales, P. Harvey, L. Ciavaglia, L. Wong, V. Ram, S. Xiao, X. Shi, X. Cheng, A. Cabellos-Aparicio, and P. Barlet-Ros, "Digital twin network: Opportunities and challenges," 2022, *arXiv:2201.01144*.

[6] A. Fuller, Z. Fan, C. Day, and C. Barlow, "Digital twin: Enabling technologies, challenges and open research," *IEEE Access*, vol. 8, pp. 108952–108971, 2020.

[7] Y.-W.-E. Sung, X. Tie, S. H. Y. Wong, and H. Zeng, "Robotron: Top-down network management at Facebook scale," in *Proc. ACM SIGCOMM Conf.*, Aug. 2016, pp. 426–439.

[8] H. Hong, Q. Wu, F. Dong, W. Song, R. Sun, T. Han, C. Zhou, and H. Yang, "NetGraph: An intelligent operated digital twin platform for data center networks," in *Proc. ACM SIGCOMM Workshop Netw.-Appl. Integr.*, Aug. 2021, pp. 26–32.

[9] M. Kherbache, M. Maimour, and E. Rondeau, "Network digital twin for the industrial Internet of Things," in *Proc. IEEE 23rd Int. Symp. World Wireless, Mobile Multimedia Netw. (WoWMoM)*, Jun. 2022, pp. 573–578.

[10] H. H. Liu, Y. Zhu, J. Padhye, J. Cao, S. Tallapragada, N. P. Lopes, A. Rybalchenko, G. Lu, and L. Yuan, "CrystalNet: Faithfully emulating large production networks," in *Proc. 26th Symp. Operating Syst. Princ.*, Oct. 2017, pp. 599–613.

[11] G. Bonofiglio, V. Iovinella, G. Lospoto, and G. Di Battista, "Kathará: A container-based framework for implementing network function virtualization and software defined networks," in *Proc. NOMS IEEE/IFIP Netw. Oper. Manage. Symp.*, Apr. 2018, pp. 1–9.

[12] D. Pediaditakis, C. Rotsos, and A. W. Moore, "Faithful reproduction of network experiments," in *Proc. ACM/IEEE Symp. Architectures Netw. Commun. Syst. (ANCS)*, Oct. 2014, pp. 41–52.

[13] M. Shin, K. Nam, and H. Kim, "Software-defined networking (SDN): A reference architecture and open Apis," in *Proc. Int. Conf. ICT Converg. (ICTC)*, Oct. 2012, pp. 360–361.

[14] A. Leivadeas and M. Falkner, "A survey on intent-based networking," *IEEE Commun. Surveys Tuts.*, vol. 25, no. 1, pp. 625–655, 1st Quart., 2023.

[15] T. A. Khan, A. Muhammad, K. Abbas, and W.-C. Song, "Intent-based networking platform: An automated approach for policy and configuration of next-generation networks," in *Proc. 36th Annu. ACM Symp. Appl. Comput.*, Mar. 2021, pp. 1921–1930.

[16] M. Van Den Brand, L. Cleophas, R. Gunasekaran, B. Haverkort, D. A. M. Negrin, and H. M. Muctadir, "Models meet data: Challenges to create virtual entities for digital twins," in *Proc. ACM/IEEE Int. Conf. Model Driven Eng. Lang. Syst. Companion (MODELS-C)*, Oct. 2021, pp. 225–228.

[17] A. Rasheed, O. San, and T. Kvamsdal, "Digital twin: Values, challenges and enablers from a modeling perspective," *IEEE Access*, vol. 8, pp. 21980–22012, 2020.

[18] S. Ji, S. Pan, E. Cambria, P. Marttinen, and P. S. Yu, "A survey on knowledge graphs: Representation, acquisition, and applications," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 2, pp. 494–514, Feb. 2022.

[19] C. El Houssaini, M. Nassar, and A. Kriouile, "A cloud service template for enabling accurate cloud adoption and migration," in *Proc. Int. Conf. Cloud Technol. Appl. (CloudTech)*, Jun. 2015, pp. 1–6.

[20] *DTN-Project*. Accessed: Jun. 23, 2023. [Online]. Available: https://github.com/DTN-Project/DTN-KG

[21] F. Tao, H. Zhang, A. Liu, and A. Y. C. Nee, "Digital twin in industry: State-of-the-art," *IEEE Trans. Ind. Informat.*, vol. 15, no. 4, pp. 2405–2415, Apr. 2019.

[22] A. Canedo, "Industrial IoT lifecycle via digital twins," in *Proc. Int. Conf. Hardw./Softw. Codesign Syst. Synth. (CODES+ISSS)*, Oct. 2016, p. 1.

[23] T. Sanislav, G. D. Mois, and S. Folea, "Digital twins in the Internet of Things context," in *Proc. 29th Telecommun. Forum (TELFOR)*, Nov. 2021, pp. 1–4.

[24] X. Wang, H. Song, W. Zha, J. Li, and H. Dong, "Digital twin based validation platform for smart metro scenarios," in *Proc. IEEE 1st Int. Conf. Digit. Twins Parallel Intell. (DTPI)*, Jul. 2021, pp. 386–389.

[25] T. Clemen, N. Ahmady-Moghaddam, U. A. Lenfers, F. Ocker, D. Osterholz, J. Ströbele, and D. Glake, "Multi-agent systems and digital twins for smarter cities," in *Proc. ACM SIGSIM Conf. Princ. Adv. Discrete Simulation*, May 2021, pp. 45–55.

[26] L. Zhao, G. Han, Z. Li, and L. Shu, "Intelligent digital twin-based software-defined vehicular networks," *IEEE Netw.*, vol. 34, no. 5, pp. 178–184, Sep. 2020.

[27] K. Ding, H. Han, L. Li, and M. Yi, "Research on question answering system for COVID-19 based on knowledge graph," in *Proc. 40th Chin. Control Conf. (CCC)*, Jul. 2021, pp. 4659–4664.

[28] H. Wang, X. Miao, and P. Yang, "Design and implementation of personal health record systems based on knowledge graph," in *Proc. 9th Int. Conf. Inf. Technol. Med. Educ. (ITME)*, Oct. 2018, pp. 133–136.

[29] K. Zhu, J. Zhou, X. Guo, F. Li, and H. Yang, "Review on knowledge graph and its application in power dispatching," in *Proc. IEEE Int. Conf. Power, Intell. Comput. Syst. (ICPICS)*, Jul. 2021, pp. 431–434.

[30] A. Syed, B. Anwer, V. Gopalakrishnan, and J. Van der Merwe, "DEPO: A platform for safe DEployment of POlicy in a software defined infrastructure," in *Proc. ACM Symp. SDN Res.*, Apr. 2019, pp. 98–111.

[31] *ONOS REST Services*. Accessed: Jun. 23, 2023. [Online]. Available: https://wiki.onosproject.org/display/ONOS/Appendix+B%3A+REST+API

[32] *YAML Ain't Markup Language*. Accessed: Jun. 23, 2023. [Online]. Available: https://yaml.org/

[33] S. Timón-Reina, M. Rincón, and R. Martínez-Tomás, "An overview of graph databases and their applications in the biomedical domain," *Database*, vol. 2021, May 2021, Art. no. baab026.

[34] Y. Cheng, P. Ding, T. Wang, W. Lu, and X. Du, "Which category is better: Benchmarking relational and graph database management systems," *Data Sci. Eng.*, vol. 4, no. 4, pp. 309–322, Dec. 2019.

[35] Z. J. Zhang, "Graph databases for knowledge management," *IT Prof.*, vol. 19, no. 6, pp. 26–32, Nov. 2017.

[36] *Neo4j Graph Database*. Accessed: Jun. 23, 2023. [Online]. Available: https://neo4j.com/xproduct/neo4j-graph-database/

[37] *Cypher Query Language*. Accessed: Jun. 23, 2023. [Online]. Available: https://neo4j.com/developer/cypher/

[38] *Abilene Network*. Accessed: Jun. 23, 2023. [Online]. Available: https://web.archive.org/web/20120324103518/http://www.internet2.edu/pubs/200502-IS-AN.pdf

[39] *GÉANT The World's Most Advanced International Research Network*. Accessed: Jun. 23, 2023. [Online]. Available: http://www.music.mcgill.ca/~ich/classes/mumt301_11/network/Topology_Oct_2004.pdf

[40] M. Kamal, S. Amin, F. Ferooz, M. J. Awan, M. A. Mohammed, O. Al-Boridi, and K. H. Abdulkareem, "Privacy-aware genetic algorithm based data security framework for distributed cloud storage," *Microprocessors Microsyst.*, vol. 94, Oct. 2022, Art. no. 104673.

[41] S. Scott-Hayward, S. Natarajan, and S. Sezer, "A survey of security in software defined networks," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 623–654, 1st Quart., 2016.

[42] M. B. Jiménez, D. Fernández, J. E. Rivadeneira, L. Bellido, and A. Cárdenas, "A survey of the main security issues and solutions for the SDN architecture," *IEEE Access*, vol. 9, pp. 122016–122038, 2021.

[43] A. Lakhan, M. A. Mohammed, O. I. Obaid, C. Chakraborty, K. H. Abdulkareem, and S. Kadry, "Efficient deep-reinforcement learning aware resource allocation in SDN-enabled fog paradigm," *Automated Softw. Eng.*, vol. 29, no. 1, p. 20, Jan. 2022.

[44] A. Mestres, "Knowledge-defined networking," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 47, no. 3, pp. 2–10, Sep. 2017.

[45] M. Bahnasy, F. Li, S. Xiao, and X. Cheng, "DeepBGP: A machine learning approach for BGP configuration synthesis," in *Proc. Workshop Netw. Meets AI ML*, Aug. 2020, pp. 48–55.

DEEPU RAJ RAMACHANDRAN POTTI received the B.Tech. degree in computer science and engineering from the College of Engineering, Thiruvananthapuram, in 2014. He is currently pursuing the M.Tech. degree with the Indian Institute of Technology Hyderabad. He is also a Research Assistant with the Indian Institute of Technology Hyderabad. From 2014 to 2019, he was a Senior Applications Engineer with Oracle. From 2019 to 2021, he was a Senior Software Developer with IBM. His current research interests include computer networks, software-defined networks, and applying artificial intelligence/machine learning in the computer networks.

TAHIR AHMED SHAIK received the B.Tech. degree in computer science and engineering, in 2020, and the M.Tech. degree in computer science and engineering from the Indian Institute of Technology Hyderabad, in 2022. His research interests include networks and security, artificial intelligence, and software-defined networks.

ANISH HIRWE received the Ph.D. degree from the Indian Institute of Technology Hyderabad (IIT Hyderabad), India. He is an Assistant Professor with the Department of Computer Science and Engineering, Indian Institute of Technology Palakkad. His research interests include network function virtualization, software-defined networks, and network management.

PRAVEEN TAMMANA received the Ph.D. degree from the University of Edinburgh, in 2018. He is currently an Assistant Professor with the Computer Science Department, Indian Institute of Technology Hyderabad (IIT-Hyderabad). Before IITH, he was a Postdoctoral Researcher with the Computer Science Department, Princeton University, USA. He has published papers in top networked systems conferences, such as NSDI, OSDI, and SOSR. His research interests include the intersection of systems, networks, and security. He received the Best Paper Award at ACM SIGCOMM SOSR 2020.

KOTARO KATAOKA received the B.A. degree in environmental information and the master's and Ph.D. degrees in media and governance from Keio University, in 2002, 2004, and 2010, respectively. Currently, he is an Associate Professor with the Department of Computer Science and Engineering, Indian Institute of Technology Hyderabad. His research interests include internet architecture, software-defined networking, network functions virtualization, blockchain, and online Japanese education.