Inferring and Querying the Past State of a Software-Defined Data Center Network

Jonathan Sherwin
Department of Computer Science
Munster Technological University
Cork, Ireland
jonathan.sherwin@mtu.ie

Cormac J. Sreenan
School of Computer Science and Information Technology
University College Cork
Cork, Ireland
cjs@cs.ucc.ie

***Abstract*—Software-Defined Networking (SDN) is used widely in Data Center Networks (DCNs) to facilitate the automated configuration of network devices required to provide cloud services and a multi-tenant environment. The resulting rate of change presents a challenge to a DCN operator who needs to be able to answer questions about the past state of the network. We describe our work in addressing this need, and how an ontological approach was taken to build a topological and temporal model of a DCN, which could then be populated using control-plane data captured in a message log. Sophisticated queries applied against the populated model allow the DCN operator to gain insight into the effects of historical automated configuration changes. We have tested our model for accuracy against a network from which a message log was captured, and we have demonstrated how queries have been formulated to retrieve useful information for the DCN operator.**

Keywords— Software-Defined Networking, Data Center Networks, Network Management, Ontologies, OpenFlow

I. Introduction

Data Center Networks (DCNs) have provided a natural environment for the application of Software-Defined Networking (SDN) [1]: the required frequency of changes to network configuration or policy, particularly in large cloud or multi-tenant DCNs, make it essential to effect those changes through software with full automation where possible. The frequency of changes to network configuration is driven by on-demand cloud-services, through which users can increase or decrease their resource requirements, and new tenants can sign-up or existing tenants leave.

Automation of DCN configuration tasks through SDN results in the state of the network changing constantly, meaning that the DCN operator no longer has certainty about, for example, which hosts can communicate with each other, or what paths packets will take over any duration of time. While these questions are easily answered for the current state of a network, with tools as simple as ping and traceroute, they are not so easily answered for past states of the network.

In previous work, we described how a historical log of control messages captured from a software-controlled DCN could be used to reconstruct the network state as it was at any point in time during its history. We introduced our system, LogSnap [2], to demonstrate the use of this approach to quickly and accurately reproduce a network in a historical state, allowing a DCN operator to review and test the replicated network, and to understand its behaviour at a specific time in the past.

In this paper, we build on our previous work, to satisfy other questions that a DCN operator might have about historical events on a network, including those that require analysing network state over a time interval rather than just at a specific past point-in-time. *Our key contributions described in this paper are methods allowing the resolution of sophisticated operator queries about past events on a DCN.* These queries can include temporal parameters, and provide temporal results.

In comparison to the state of the art, our contributions represent a novel enhancement to techniques for analyzing SDN networks based on historical information. The challenges in achieving these contributions have been to, firstly, infer a topological and temporal model of a DCN using key information identified and extracted from network control messages, and, secondly, provide meaningful and useful answers to high-level temporal (and other) queries posed by the DCN operator.

The rest of this paper is structured as follows: In Sections II and III, we provide further detail on our motivation and research challenges. We describe our methodology and design in Section IV. Section V presents our Implementation and Evaluation. Section VI is Related Work, and Section VII contains the Conclusions and Future Work.

II. Motivation

Why would a DCN operator want to query the past state of their network? Reasons include auditing and compliance, troubleshooting, verification, and testing 'what-if' scenarios. An example of auditing would be to check that the forwarding rules in place on a DCN at a particular time match the policy specified by the operator. For compliance purposes, it might be necessary to show that forwarding behaviour was within the terms of a service level agreement with a tenant of the data center - as stated in [3], "In a multi-tenant environment, the ability to securely contain and isolate tenant traffic is a fundamental requirement". Having the ability to query past network state can be useful when troubleshooting one-off or intermittent problems. A DCN operator might want to verify reachability, isolation or other network properties for time points or intervals

We acknowledge the support of SFI CONNECT, grant 13/RC/2077.
Jonathan Sherwin is funded by Munster Technological University.

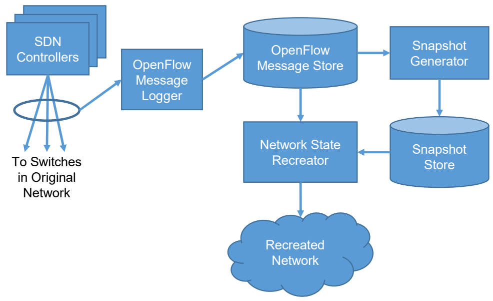

Fig. 1: LogSnap Architecture Diagram (previous work)

in the past. 'What-if' queries could be used to plan for failures, e.g. what flows and tenants would have been affected if a specific switch or link failed at a particular point in time? NetFlow / IPFIX [4, 5] and other protocols have been used by Network Management Systems to collect data-plane flow statistics to give network operators some visibility into historical traffic patterns. These protocols do not take advantage of the extra information contained in control-plane messages. Other researchers have described their approach for recording and playing back OpenFlow packets on a campus LAN [6].

In a multi-tenant DCN, it is difficult to look back to see exactly what happened at a time of interest, let alone to see why it happened. When LogSnap was designed (Fig. 1), the aim was to capture control-plane messages from a software-controller DCN, record those messages to a log, and use the log to reproduce the network as it was at any single point in time. The reproduced network could be used by a DCN operator to test and examine the state of the original network as it was at that point in its (possibly quite distant) history. While this can provide a means to answering some questions the operator might have, for example regarding reachability, isolation, and paths, there are many other questions that could be answered with the information contained in the message logs if an appropriate query mechanism was provided. This paper presents our work on such a mechanism.

Table I lists some queries of interest to a DCN operator, presented in a natural language style. For each query, a single time point or a time interval might be provided as a parameter.

The answer to Query #1 might be that switches are communicating with the SDN controller as expected, or it might reveal that one or more switches that the operator knows to be physically present in the DCN were not 'active' for some time – i.e. not connected to the SDN controller and therefore not receiving instructions regarding flow rules to allow data-plane packets to be forwarded by those switches.

TABLE I. SAMPLE QUERIES

#	Query	Query Type
1	What switches were active in the network, and when?	Enumerating elements
2	For what hosts were flow-rules installed in switch flow-tables, but for which no packets were observed in the DCN?	Statistics for elements
3	What path would a specified packet follow through the network?	Path determination

Applying Query #2 might yield evidence that the network policy (a high-level description of how the network should be configured to meet organisational or customer requirements) may need to be reviewed to remove out-of-date requirements.

The intention behind Query #3 is to provide insight into how a packet with specific values in its header fields would be forwarded through the DCN at a time of interest, based on the topology and set of configured flosw-rules at that time.

All three queries could lead to more questions that may or may not be answerable solely from the contents of the message log, but they should provide concrete information on which to base further investigation by the DCN operator, and may lead to the identification of other sources of information that could be recorded to augment the picture provided by the control-plane message log.

III. Research Challenges

The first challenge in addressing the goals above is to identify an appropriate method for modelling the DCN. The model must represent the topology of the network and the state of the switch flow-tables, and must change over time to reflect the evidence gathered from control-plane messages. An SDN controller maintains a model of the network, but it reflects the controller's *current* view of the network and does not retain historical information. Furthermore, the controller's model is designed to meet the functional requirements of the controller, not to act as a faithful record of the state of the network, including situations where the controller's view was incorrect.

The second challenge is to demonstrate how to formulate and apply queries against the model to provide meaningful information for DCN operators. The task requires the expression of informal, natural language questions in terms of the concepts and relationships in the model. The sample queries from Table I provide a basis for such a demonstration, and will be referred back to in the sections below.

A third challenge relates to the accuracy of the control-plane message log that currently provides the input data from which queries are answered. There is the possibility that some messages may not have been captured, partly due to the passive, unintrusive method used by LogSnap to collect control-layer information. While our current work is based on message logs recorded by LogSnap, it could use any source of OpenFlow messages. It is worth noting that any packet-capture based source of such messages incurs the risk of missing data if, for example, the capture utility being unable to keep up with peak control-traffic flow. Other reasons for a message log appearing to be incomplete include a device such as a switch failing, or a software process such as a controller crashing. Furthermore, the timestamps on the messages are not necessarily the exact times at which the events occurred that are described by the messages. For example, when a switch tells a controller that it has removed a rule from a flow table, some time may have elapsed between when the rule was removed and when the switch sent the message to the controller. Or when a switch acknowledges a controller request to add a flow-rule to a flow-table, it might be assumed that the rule was added at some point in time between when the switch received the add_flow request and when it sent the acknowledgement (although other researchers have shown

[7] that the flow-rule may be added after the switch sends the acknowledgement). These considerations need to be taken into account in any solution that aims to provide meaningful answers to DCN operator queries regarding the past state of the network.

IV. Methodology and Design

A. Query System Requirements

Providing a method to query the past state of a DCN depends on a number of requirements being satisfied. Firstly, a model of the network must be created that captures the topology, and the changes in the topology over time. By topology, we mean the links between switches, between switches and hosts, and connections between switches and controllers. By changes in topology over time, we mean 'what is the lifespan of each node, link and connection, and relevant state information for those elements?', bearing in mind that the lifespan of each element can have different start and end points, and indeed can be discontinuous – for example, if an element such as a host is removed from the network and later re-added. The relevant state information differs for each element, but includes, for example, flow-rules and flow-tables on switches, each of which may have their own lifespans, constrained within the lifespan of the device, but otherwise independent.

The second requirement is an approach to encoding queries. It must be possible to query every element in the model – with temporal ('when?'), topological ('where?'), and other query attributes. The sample queries in Table I contain such attributes.

As a third requirement of any proposed method, in light of the considerations discussed in section III, some context should be available for query answers to assess their veracity.

B. Ontological Approach

The concept of ontology originates from the domain of philosophy, and seeks to classify objects and explain their structure. An ontological approach has been used in many disciplines to create frameworks and taxonomies.

Ontologies in Computer Science are most closely associated with the semantic web [8] and AI [9]. An ontology formally encodes relationships between concepts. Those concepts can represent categories of physical or abstract items. Ontologies have been used successfully for SDN-related research efforts heretofore. One use-case for an ontological approach is to provide a mapping between overlapping sets of terminology used by different groups within the same domain. For example, creating a formal mapping between SDN controller requests and legacy network device configuration languages. Another use-case is the construction of a framework to define concepts for which instances can be constructed using data pulled from different sources – and this approach has been applied to combine collected network monitoring data to mine for troubleshooting purposes. For more, see Section VI.

Our use case for an ontological approach is to facilitate the construction of a logical framework based on control-plane messages, through which the challenge of inferring a topological and temporal model of a DCN can be met. A less formal approach could have been taken, however the ontological approach has the advantage of making consistency and correctness easier to achieve, especially important when dealing with large sets of data. Moreover, ontologies can be extended and enhanced without impacting previously working code – for example, to accommodate changes in future versions of OpenFlow. Lastly, we can query an ontology once it has been populated with data as properties of instances of the concepts defined within the ontology.

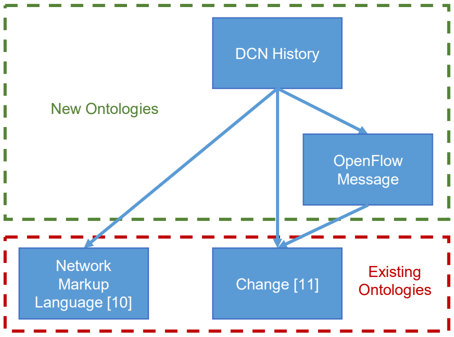

Fig. 2: Hierarchy of Ontologies
(arrows represent dependencies)

For our purposes, we need to represent several groups of concepts ontologically: basic concepts such as control-plane messages, network devices and connections; temporal concepts; and higher level concepts built on the basic ones. We constructed a set of ontologies (Fig. 2) to map from the information contained in control-plane messages to higher level abstractions. Our ontologies are built on existing ontologies where possible, and a rich set of relationships allows us to infer from control messages the existence of entities such as network switches, hosts, links and controllers, and the states of those entities. A benefit of the inference process is that it can reveal where information is missing from the log, i.e. control messages that may not have been captured for reasons as discussed in section III.

Our new ontologies are built on two existing ontologies: the Network Markup Language (NML) Base ontology [10], and the Change ontology [11]. The NML ontology contains concepts for network objects such as nodes, ports, and links, defines properties of these concepts, and relationships between the concepts. Its purpose is to facilitate the description of a traditional computer network, without SDN-specific characteristics. The Change ontology gives us sophisticated temporal concepts, allowing us to describe elements of a network where some properties of those elements may change over time. For example, the IP address of a host may change, while the MAC address stays the same. Or a host may be connected to one switch for an interval, but may be moved and connected to a different switch for a subsequent interval.

The OpenFlow Message ontology describes types of OpenFlow messages and their properties, with the timestamp indicating when a message was captured as an additional property. Part of the ontology is shown in Fig. 3. Although the message ontology is relatively simple, its usefulness is the ability to, having populated it with message instances, link message instances with instances of other ontological concepts. For example, a message instance might be connected to a switch instance to represent the relationship "this is the first message that was captured from switch X", or connected with a flow-rule instance to represent the relationship "this is the message that caused flow-rule Y to be installed in the flow-table of switch Z".

The DCN History ontology describes the network elements that were present over time, and their relationships. These elements are the SDN controller(s) and SDN switches that exchanged control-plane messages, the links that the controller learned of through topology discovery, and hosts that the controller learned of from switches or potentially from other sources. SDN switches contain flow-tables, flow-tables contain flow-rules, and the tables and rules have properties that can change over time. Part of the DCN History ontology is shown later in this section, in Fig. 4.

The OpenFlow Message ontology is defined separately from the DCN History ontology for the reason that, although the message log we have as source data contains OpenFlow messages, our approach can be applied using logs of other control-plane protocol messages (NETCONF, OVSDB, and/or BGP-LS, for example), or more generally any source of network management information. While the OpenFlow Message and DCN History ontologies are separate, they are related – since the OpenFlow messages are exchanged between controllers and switches, which are elements of the DCN. In fact, our approach is to infer the network elements and relationships from the OpenFlow messages.

C. Inferring DCN Elements from Control-Plane Messages

Inferring switches and controllers. OpenFlow messages are sent by switches and controllers, providing direct evidence of the existence of those DCN components. A sequence of messages from a switch or controller indicates that the component existed for the duration of the message sequence. A gap in communication between a switch and controller might indicate that the switch became disconnected from the controller, or was rebooted, or might even have been removed and replaced with a component that presents the same identity. In the latter case, clearly the intention is that the replacement is taking the role of the component it replaced. Notwithstanding, the gap represents an event that should somehow be signified. An SDN controller typically communicates with multiple switches. Where these communication sessions overlap, it is certain that the controller was present from the start of the first overlapping session to the end of the last overlapping session.

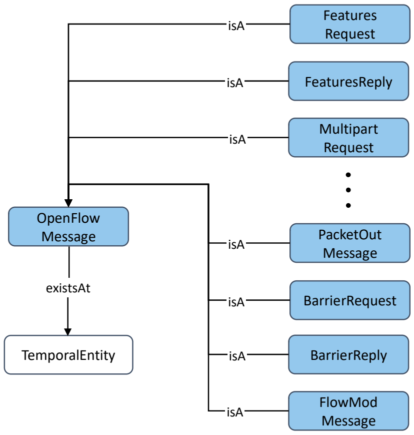

Fig. 3: OpenFlow Message Ontology (Partial)

Inferring inter-switch links. Other topological components are learned of indirectly, from packets contained in OpenFlow PacketIn messages. Some such messages are the result of a topology discovery process, by which a controller instructs switches to output discovery packets through all ports, and to forward any received discovery packets back to the controller. The topology discovery process is re-run periodically (e.g. every 3 to 5 seconds, depending on the controller). A switch will also inform the controller via a PortStatus message if a port changes state (up or down). The existence and duration of inter-switch links, and the switch-ports connected via those links, can be inferred from these exchanges.

Inferring hosts and host-switch links. Other PacketIn messages in the capture log may contain ARP packets that can be used to infer the existence of hosts and their links to switches. The usual response of an SDN controller to receipt of an ARP request is to install one or more flow-rules to allow the communication to progress. These flow-rules will have a lifespan that we can use as evidence of the continued existence of the host if at least one of the flow-rules is host-specific. The capture log should contain the FlowMod request sent from the controller to instruct the switch to install the flow-rule. It should also contain a subsequent FlowRemoved message from the switch to the controller indicating that the flow-rule was removed (if the controller requested this behaviour when it first installed the flow-rule). A switch will remove a flow-rule at the request of the controller, or if the flow-rule had an idle-timeout or hard-timeout value that expired. SDN controllers periodically request statistics information about flow-rules from switches, and from the replies recorded in the capture log we can identify the point at which a flow-rule stopped matching packets (i.e. the host stopped sending packets) as being within a time interval. In the event that there was no initial ARP request, or at least none recorded, the statistics information indicating that a flow-rule started matching packets can support the inference of a host's existence. Furthermore, a host will often communicate with multiple other hosts, and combining the information gathered about those multiple communication sessions helps to build the picture of the full lifespan of the host in the DCN.

Inferring flow-rules and flow-tables. Individual flow-rule details (match criteria, actions, priority) can be extracted directly from the FlowMod messages instructing switches to add those rules. Flow-rule statistics can be gathered from MultipartReply messages sent by switches in response to requests from controllers, and can also be used to infer the existence of flow-rules for which the FlowMod messages might have been missed. A flow-rule will normally be explicitly removed by a controller when it is no longer required, and, as mentioned above in relation to inferring hosts, FlowRemoved messages are a further indicator of the end of the lifespan of a flow-rule. It can be assumed that a switch has at least one flow-table when it initially connects to a controller. For later versions of OpenFlow, the controller can query the switch's flow-table configuration and request re-configuration via TableFeatures messages.

Inference dependencies. The inference of any DCN element depends on the message ontology having been populated with message instances. Furthermore, the inference of some DCN elements depends on the prior inference of other elements, as outlined above and as detailed in Table II.

TABLE II. INFERENCE DEPENDENCIES

DCN Element	Depends On
Switch	Control-plane messages
Controller	Switch; control-plane messages
Port	Switch; control-plane messages
Inter-switch link	Port; control-plane messages
Flow-rule	Switch; control-plane messages
Flow-table	Switch; flow-rule
Host	Switch; control-plane messages
Host-link	Port; host

D. *Representing Change Over Time in a DCN History*

Our DCN model requires the extra dimension of time, to represent how long a DCN element such as a switch was present in a network, the lifespan of each link connecting that switch to other switches, the duration for which each flow-rule is present in the flow-table of the switch, and so forth. We apply Krieger's 4D approach [12] to representing change over time in our DCN History ontology, following the design pattern described in [10]. We believe this to be the first application of the 4D approach to a model of an SDN. Applying Krieger's 4D approach allows us to re-use in our ontology concepts from the atemporal Network Markup Language ontology.

When representing an instance of a switch, for example, some switch properties are constant, others change. 4D splits the representation into two classes: a perdurant class with properties that are invariant, and a manifestation class that can have multiple temporal instances with individual property values. An invariant property of an OpenFlow switch is its DPID (DataPath Identifier), hence that property belongs to the switch perdurant. Since the IP address of a switch can change, as can the TCP or UDP port from which the switch connects to a controller, those are properties of switch manifestations. Another property of a manifestation is the time interval for which it existed.

Our temporal entities can be nested: A switch manifestation contains a flow-table. A flow-table contains flow-rules, of which each flow-rule has perdurant and manifestation parts. The priority and match-criteria of a flow-rule cannot change, but the other properties, e.g. instructions, timeouts and counters, can. The instructions and timeouts of an existing rule can be modified by a controller via a FlowMod message. The 4D representation of a switch, with nested 4D representation of a flow-rule, is illustrated in Fig. 4.

E. *Expressing DCN History Queries*

The sample queries presented in Table I can now be expressed in terms of the ontological concepts discussed so far. A temporal parameter is assumed (a single time-point or an interval), but can be omitted.

Query #1 'What switches were active in the network?' can be expressed more formally as:

Find the set $\{P\}$ of switch manifestations, where the start and end points of the manifestations are within time interval $[x..y]$

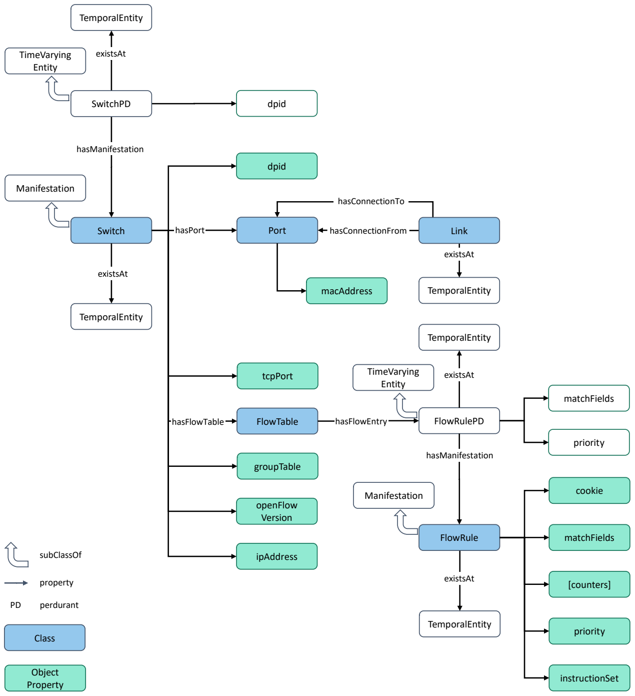

Fig. 4: DCN History Ontology (Partial)

Query #2 'For what hosts were flow-rules installed in switch flow-tables, but for which no packets were observed in the DCN?' can be expressed as:

Find the set $\{Q\}$ of flow-rules, where for each q an element of $\{Q\}$ there is no host manifestation h, where the IP Address property of h equals the IP address match property of q or the MAC Address property of h equals the MAC Address property of q, and where the start and end points of q are within time interval $[x..y]$

Query #3 'What path would a specified packet follow through the network?' is a graph problem, but not a typical 'best path' problem because each OpenFlow switch forwards packets based on its configured rules and their priorities, and the actual destination is not known in advance. The desired destination may be in the packet header, but the actual destination depends on the configured flow-rules. The query might be expressed in terms of our defined concepts as follows:

Given an injection point p representing the switch port on which a packet entered the network, and the set of packet header field values $\{R\}$, where each element of R is a tuple of field name and value (f, v), and a time-point t, find the sequence $[S]$ where each item s in $[S]$ is a switch except for s_{last}, which may be a switch, a host or a controller; s_1 is the switch to which p belongs, and s_i is the switch, host or controller to which switch s_{i-1} will forward a packet based on the highest priority flow-rule u present in the flow-table on switch s_{i-1} with a set of match criteria $\{M\}$ consisting of field name and value tuples (n, o) where $\{M\}$ matches $\{R\}$, if any such rule exists at time t.

The semi-formal queries above must be encoded to be applied against the populated ontological model.

V. Implementation and Evaluation

The objective of our implementation is to demonstrate that the challenges presented in Section III have been met – i.e. that our inferred model of a DCN topology matches the original network and can track the changes in the original network accurately; that queries can be applied against the model to provide useful answers to DCN operators; and that the answers provide context to allow the impact of inexact data (or an incomplete message log) on their accuracy to be assessed.

The implementation workflow is shown in Fig. 5. The tools used at each numbered step in the workflow are as follows:

Step 1. We created our ontologies with Protégé [13], an actively developed authoring tool supporting evolving standards and widely used in the ontology development community.

Step 2. The ontologies we created were exported from Protégé as OWL (Web Ontology Language) files expressed in Turtle syntax, then imported into AllegroGraph [14], a graph database used in this work for its triple-store capabilities. While Protégé can act as a triple-store using in-memory storage, or combined with a backend database to store and access large amounts of data, AllegroGraph is designed to be high performance and massively scalable – properties essential for handling the quantity of data extracted and inferred from the control-plane message log of a multitenant DCN.

Step 3. Python code was written to populate the message ontology in AllegroGraph with instances using data read from the OpenFlow message log stored in ElasticSearch [15] by LogSnap. Both AllegroGraph and ElasticSearch provide Python APIs. While ElasticSearch stores full OpenFlow message details, only key data is written to AllegroGraph to populate the ontology with the required triples.

Step 4. Modules to infer temporal instances of network elements were written in AllegroCL Common Lisp, which interfaces with AllegroGraph, and Allegro Prolog, which is embedded in AllegroCL. Where possible, inference logic was encoded as Prolog rules to allow consistency checking, however the Allegro Prolog implementation is limited so additional Lisp code provided the required inferencing functionality.

Step 5. Queries were encoded in Prolog and SPARQL [16], executed from within Lisp modules – or SPARQL queries executed from Python if the results were to be charted. SPARQL is a W3C-standard semantic query language supported by AllegroGraph and accessible via Lisp and Python. Our experience has been that SPARQL allowed us to write more expressive queries than Prolog, although the queries were not readily composable. Results were visualised with Plotly.

Steps 1 to 5 can be repeated: If an ontology is changed or added to (Step 1), then the other 4 steps should be repeated to obtain new answers to queries on the updated ontology. If a new or updated message log is available, then Step 3 onwards should be re-done. If the inference modules are modified, the restart point is Step 4. Step 5 can be repeated indefinitely, and if new queries are created previous steps do not need to be re-done.

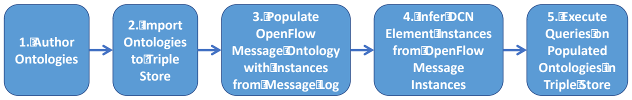
1. Author Ontologies
2. Import Ontologies to Triple Store
3. Populate OpenFlow Message Ontology with Instances from Message Log
4. Infer DCN Element Instances from OpenFlow Message Instances
5. Execute Queries on Populated Ontologies in Triple Store

Fig. 5: Workflow Diagram

A. *Inferring DCN Elements – Implementation Issues*

In Section IV, it was stated that the lifespan of a controller instance could be inferred from a set of sessions between switches and the controller that overlap in time. The task of identifying sets of overlapping sessions was recognised as lending itself to being formulated as a graph problem. The nodes of the graph correspond to switch-controller sessions. An edge connects two nodes if the two corresponding sessions overlap in time and the identity of the controller is the same for the two sessions (matching IP addresses and port numbers). A group of nodes is a 'connected-component' if all nodes in the group have at least one path to every other node in the group. Each connected-component of the graph contains the nodes corresponding to the overlapping switch-controller sessions that represent the lifespan of the controller instance. The start point of the controller instance is the earliest session start point in the set, and the end point of the controller instance is the latest session end point in the set. The 'graph' library available via Quicklisp [17] provides a 'connected-components' function. Fig. 6 illustrates how each identified connected-components group of switch-initiated sessions with a controller relates to, and can be used to infer, a controller instance.

Our implementation currently assumes one flow-table per switch; and that controllers will be notified when a flow-rule is removed, as can be specified when the flow-rule is first added.

B. *Implementing Queries*

The triple-store is ready to be queried once the ontology is populated with OpenFlow message instances, and DCN elements inferred. We implemented queries including the three given in Table I, and revisited in Section IV. Queries #1 and #2 translate to SPARQL. Query #2 requires more search terms and logical operations, being more complex. SPARQL queries are constructed dynamically with optional parameters – for example to add a time filter to reduce the search interval, if necessary. Query #3 required a combination of SPARQL and Lisp.

Adding new queries currently requires additional code – in SPARQL, Prolog, Lisp, and/or Python. New concepts can be added to the ontologies (for example 'Failing Switch', a concept based on 'Switch' but taking other criteria into account, such as switch-controller session length and frequency), and logic added

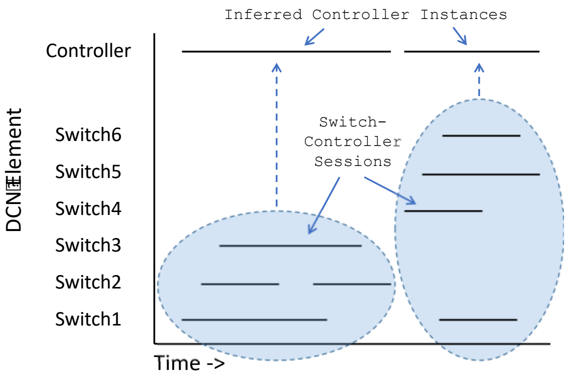

Fig. 6: Inferring Controller Instances from Switch-Controller Sessions (illustration purposes only)

as an additional inference module to infer instances of those concepts. Following the appropriate steps in the workflow (Fig. 5), the new concepts, instances, and their properties are made available for use in queries and as a basis for other concepts.

C. *Evaluation – Accuracy of the Inferred DCN Elements*

To evaluate the accuracy of our inference of DCN elements, we ran experiments using OpenFlow message logs captured on emulated networks with different DCN topologies, including: a traditional three-layer topology, as used historically in data-centers; a spine-leaf topology, used on its own or as a building block in modern multi-tenant data-centers; and a fat-tree (k=6) topology. For a range of DCN topologies, see [18].

The verification message logs are from experiments where light traffic was generated (using DCT2Gen's [19] TrafficGen utility) on the DCN for two minutes. The traffic profile specifies 10MB TCP transfers between pairs of hosts scheduled to start one per second, with 80% rack-local / 20% inter-rack traffic. For each message log, the ontologies were populated with message instances (Fig. 5, step 3) – and the remaining workflow steps followed. Query results were charted where appropriate.

Fig. 7 shows a composite inferred topology from the spine-leaf DCN message log. The chart is 'composite' because it is the result of a query for all DCN elements inferred within the time range covered by the message log, even if these were not present at the same time. The time range can be narrowed to query the topology for shorter windows of time. For comparison, Fig. 8 is a logical diagram of the original network - consisting of 6 leaf switches and 6 spine switches, connected in a folded-clos arrangement, with 20 hosts connected to each leaf switch. Clearly, the inferred network topology contains the same hosts, switches and links as the original network.

In addition to message logs, we had periodic dumps of flow-table contents from the DCN switches on the original networks. By querying the populated ontologies for the flow-rules that were inferred to be present at the time of each periodic dump, and comparing the two sets of flow-rules, we verified that both matched for each network. The results confirmed the accuracy of our inferred DCN elements.

D. *Evaluation – Accuracy of Query Results*

The sample queries initially presented in Table I, and discussed further in Section IV were applied against the populated ontologies. The results of applying queries #1 and #3 against the ontology for the spine-leaf network are shown in Figs. 9 and 10. For query #2, the result was that there were no flow-rules installed for hosts that were not active, which was as expected since the SDN controller was operating in reactive mode, and only installing flow-rules as switches reported packets for new data-plane flows.

In Fig. 9, the inferred set of DCN switches can be seen, identified by DPID, and their active lifespans for which they were connected to the controller. The chart shows the switches did not all connect to the controller at exactly the same time, and each switch had a single instance with an uninterrupted lifespan. This corresponds with observations from the original DCN.

Fig. 10 shows the inferred path for a packet sent from host 10.0.3.11 through the link on which it was connected to a switch. The header field values for the packet are 10.0.3.11's MAC address as source, host 10.0.4.11's MAC address as destination, an EtherType value of 0x800 (for an IP datagram. A transmission time for the packet was specified, selected based on when a flow was scheduled in the TrafficGen profile to be active between the two hosts on the original network. Since the destination host is in a different rack, the configured flow-rules establish the path shown in Fig. 10 across several switches to deliver matching packets to their destination.

VI. Related work

Ontologies have been employed for network management for quite a number of years, and more recently have been applied to aspects of managing a Software-Defined Network.

ReasoNet [20] maintains an ontology representing the current state of an SDN, checking this state for correctness against rules that are encoded in the ontology. Requests from ReasonNet-aware apps on the SDN controller are subject to conflict resolution. The OpenFlow ontology, reasonet-schema, is available online. The researchers in [22] describe their application of machine learning to data collected using the northbound interface of an SDN controller, to identify symptoms and causes of faults. Domain knowledge is encoded in their SDNDL (SDN Description Language) ontology.

Neither reasonet-schema nor SDNDL have any temporal dimension to them to represent changes in network topology or flow tables over time. They do not describe SDN control-plane concepts in enough detail to be able to, for example, relate events on a network with the control-plane messages that were exchanged to communicate the occurrence of the events. Neither ontology describes the various OpenFlow message types.

We have created ontologies that contain concepts to describe OpenFlow control-plane messages and their contents, as well as concepts to describe a network topology, flow-tables and events. Temporal topological instances of controllers, switches and hosts, and temporal instances of flow-rules are inferred from OpenFlow messages in a log of packets from exchanges between controllers and switch in a DCN.

ForenGuard [21] uses a non-ontological approach to identifying root causes of forwarding problems, monitoring and analysing related previous data-plane and control plane activities. However it is controller-specific, and focussed on resolving specific security-related issues.

VII. Conclusions and Future Work

The challenges in this work were to find an appropriate method for modelling a DCN, to demonstrate how to formulate and apply queries against the model, and to ensure that context information is available to assess the accuracy of the query results. We described how an ontological approach supported the development of a logical framework, enabling construction of a topological and temporal model of a DCN. We listed sample queries in natural language, expressed them in terms of the concepts and relationships defined in the historical model of the DCN, and outlined how the model could be extended. We described how we tested our model's accuracy, having instantiated the ontologies from a control-plane message log. We included graphical representations of the results of applying

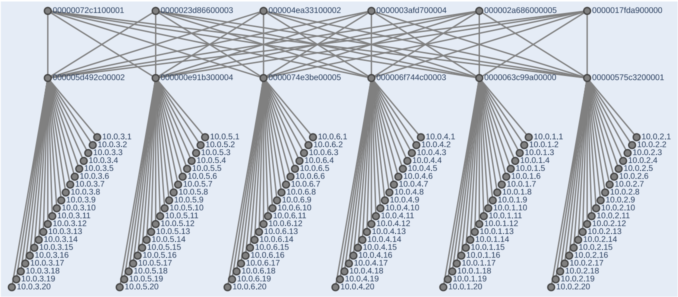

Fig. 7: Composite Inferred Topology for a Spine-Leaf DCN - switches identified by OpenFlow DPID, hosts by IP address.

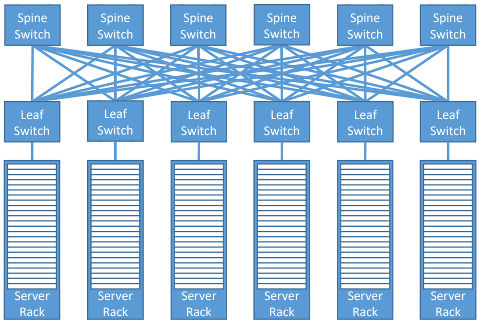

Fig. 8: Logical Diagram of Original Spine-Leaf DCN

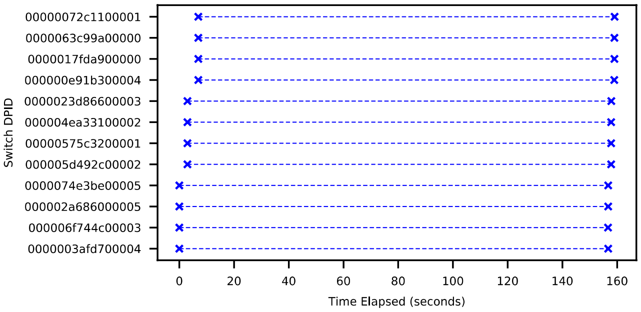

Fig. 9: Inferred Switch Instance Lifespans (Query #1) for Spine-Leaf DCN

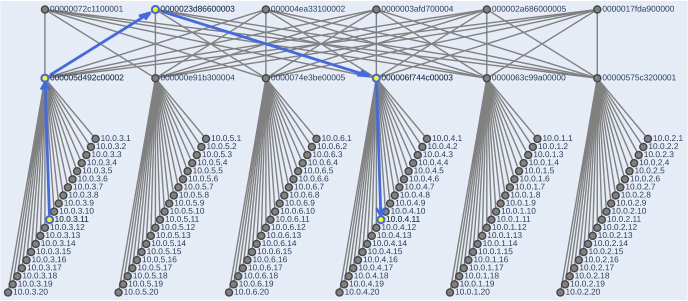

Fig. 10: Inferred Path for Packet 'X' (Query #3) on Spine-Leaf DCN

the queries to the populated ontologies. The queries can return detail to show the data points and relationships used to formulate the answers, in order to assess their veracity.

For the future, we plan to extend the DCN model with higher level abstract concepts, their properties, and relationships to the current set of concepts. We will work on an extensible, modular query system allowing a DCN operator construct new queries.

REFERENCES

[1] J. Son and R. Buyya, "A taxonomy of software-defined networking (SDN)-enabled cloud computing," *ACM Computing Surveys (CSUR),* vol. 51, no. 3, pp. 1-36, 2018.

[2] J. Sherwin and C. J. Sreenan, "LogSnap: Creating Snapshots of OpenFlow Data Centre Networks for Offline Querying," in *10th International Conference on Network of the Future (NoF 2019)*, 2019.

[3] *Cisco Virtualized Multi-Tenant Data Center, Version 2.2 Design Guide*. Cisco Systems, 2013.

[4] B. Blaise, "RFC 3954: Cisco Systems NetFlow Services Export Version 9," 2004.

[5] B. Blaise, B. Trammel, and P. Aitken, "RFC 7011: Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information," ed: Internet Engineering Task Force, 2013.

[6] A. Wundsam, D. Levin, S. Seetharaman, and A. Feldmann, "OFRewind: enabling record and replay troubleshooting for networks," presented at the Proceedings of the 2011 USENIX Annual Technical Conference, Portland, OR, 2011.

[7] M. Kuźniar, P. Perešíni, and D. Kostić. *What you need to know about SDN flow tables, Lecture Notes in Computer*, vol. 8995, pp. 347-359, 2015.

[8] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," *Scientific american,* vol. 284, no. 5, pp. 34-43, 2001.

[9] B. Chandrasekaran, J. R. Josephson, and V. R. Benjamins, "What are ontologies, and why do we need them?," *IEEE Intelligent Systems and their applications,* vol. 14, no. 1, pp. 20-26, 1999.

[10] J. van der Ham, F. Dijkstra, R. Lapacz, and A. Brown, "The Network Markup Language (NML) A Standardized Network Topology Abstraction for Inter-domain and Cross-layer Network Applications," presented at the TNC2013, 2013.

[11] M. Katsumi and M. Fox, "A Logical Design Pattern for Representing Change Over Time in OWL," *Proceedings of the 8th Workshop on Ontology Design and Patterns (WOP 2017)*.

[12] H.-U. Krieger, "Where temporal description logics fail: Representing temporally-changing relationships," in *Annual Conference on Artificial Intelligence*, 2008: Springer, pp. 249-257.

[13] J. H. Gennari *et al.*, "The evolution of Protégé: an environment for knowledge-based systems development," *International Journal of Human-computer studies,* vol. 58, no. 1, pp. 89-123, 2003.

[14] *AllegroGraph. Accessed: 2021-07-10. [Online]. Available: https://allegrograph.com/products/allegrograph/*

[15] *What is the ELK Stack?* Accessed: 2021-07-10. [Online]. Available: https://www.elastic.co/elk-stack

[16] C. Buil-Aranda *et al.*, "SPARQL 1.1. Overview. W3C Recommendation.." Accessed: 2021-07-10. [Online]. Available: https://www.w3.org/TR/sparql11-overview/

[17] *Quicklisp beta*. Accessed: 2021-07-10. [Online]. Available: https://www.quicklisp.org/beta/

[18] T. Chen, X. Gao, and G. Chen, "The features, hardware, and architectures of data center networks: A survey," *Journal of Parallel and Distributed Computing,* vol. 96, pp. 45-74, 2016.

[19] P. Wette and H. Karl, "DCT2Gen: A traffic generator for data centers," *Computer Communications,* Article vol. 80, pp. 45-58, 2016.

[20] C. Rotsos *et al.*, "ReasoNet: Inferring Network Policies Using Ontologies," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, 25-29 June 2018 2018, pp. 159-167, doi: 10.1109/NETSOFT.2018.8460050.

[21] F. Benayas, Á. Carrera, M. García-Amado, and C. A. Iglesias, "A semantic data lake framework for autonomous fault management in SDN environments," *Transactions on Emerging Telecommunications Technologies,* p. e3629, 2019, doi: 10.1002/ett.3629.

[22] H. Wang, G. Yang, P. Chinprutthiwong, L. Xu, Y. Zhang, and G. Gu, "Towards Fine-grained Network Security Forensics and Diagnosis in the SDN Era," Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, Toronto, Canada, 2018.