A Smart Network Repository Based on Graph Database

Amar Abane[1], Abdella Battou[1], Mheni Merzouki[1], and Tao Zhang[1]

National Institute of Standards and Technology, Gaithersburg MD, USA
{first.last}@nist.gov

Abstract. To address the increasing complexity of network management and the limitations of data repositories in handling the various network operational data, this paper proposes a novel repository design that uniformly represents network operational data while allowing for a multiple abstractions access to the information. This smart repository simplifies network management functions by enabling network verification directly within the repository. The data is organized in a knowledge graph compatible with any general-purpose graph database, offering a comprehensive and extensible network repository. Performance evaluations confirm the feasibility of the proposed design. The repository's ability to natively support 'what-if' scenario evaluation is demonstrated by verifying Border Gateway Protocol (BGP) route policies and analyzing forwarding behavior with virtual Traceroute.

Keywords: network management · network data · repository · knowledge graph · SDN · Neo4j · control plane verification · data plane analysis · database-defined networks

1 Introduction

In the current era of rapidly growing network complexity and service diversity, preventing and detecting undesired behaviors is still a daunting task despite the increasing number of network management tools. One core challenge lies in the fragmented nature of network data. It is dispersed across various sources, management functions, and tools, which often lack a uniform representation for optimized usage. This scattering inevitably breeds inconsistency, reducing the utility of data and complicating its processing. Furthermore, when operators evaluate configuration changes using verification tools, it necessitates the transfer of network data from its original storage, such as databases or configuration file repositories, to the analytical tool. This approach further requires supplementary logic within the management software to process the data and integrate results, making conventional network repositories based on traditional databases [5] and distributed maps [1] inefficient.

To tackle these challenges, we present a novel repository to uniformly store and organize network data. This repository is grounded in a graph database

(GDB) which allows analysis of network data within the storage system. Creating a repository capable of addressing the intricate nuances of multi-technology network architectures, such as Beyond 5G (B5G) [6], necessitates a model that can represent these varied concepts uniformly. It's crucial for the model to align with various management scenarios by retaining the relationships in topology, connectivity, and configuration, bridging the gap between low-level network elements and high-level network intent [3]. In addition, the model must remain comprehensible and easily expendable to accommodate additional data, protocols, and technologies. Therefore, the proposed network repository is designed as a Knowledge Graph (KG) integrating semantics into the stored data. Its compatibility with general-purpose GDB allows for the construction of the KG with a systematic procedure developed in this paper. Performance assessments validate the KG's suitability as an effective network repository, underscoring its potential to enhance network management workflows.

By integrating *what-if?* scenario verification for control plane and data plane without additional code, the proposed repository simplifies the development logic of management functions, paving the way for a smart network repository. We showcase this ability through a proactive verification of route policies for Border Gateway Protocol (BGP) and an analysis of network forwarding behavior with a virtual Traceroute process. The smart repository code is publicly accessible and the features presented in this paper are reproducible[1].

This paper is organized as follows. Section 2 reports on recent approaches for storing and handling data for network management and discusses the differences with the repository proposed in this paper. Section 3 presents the design and construction of the proposed network repository and its knowledge graph schema. Section 4 provides an evaluation of the repository and discusses the implemented functionalities. Section 5 concludes the paper and discusses limitations.

2 Related Work

By using a KG to describe network data, it is possible to not only represent the network's physical and logical attributes but also uncover relationships across various dimensions within the network [4]. Many studies consider KGs to store different kinds of network data. Authors in [10] use a KG to represent network element attributes and topology of the physical network on which a ML-based technique is used to extract topology relationships between network elements. Our approach shares the same idea of representing network data using a KG. However, the proposed repository goes beyond network topology and includes device configuration, network services, routing behavior, and data plane. In [2], a solution is proposed to simplify network management workflow using a unified graph-based model for device, network, and service models. Data is represented through multiple graphs without detailing how the graphs are organized, making the model difficult to understand and reproduce. Our approach considers

[1] https://github.com/usnistgov/smart-network-repository

the same kinds of network data but uses a single graph to represent them, and defines an understandable and reproducible method to build and extend the repository. KnowNet [4] builds a KG which captures and connects network data of a Software Defined Networking (SDN) controller. Applications exploit the KG to detect and respond to network issues, and derive new knowledge. Typical management applications are demonstrated, but the collection and organisation of the data is not specified and the KG is internally implemented in the tool without a standard interface to interact with it. SeaNet [9] proposes an autonomic network management solution for SDN driven by a KG and a telecommunication network ontology [8]. Our KG differs from SeaNet's as it supports the representation of configuration-based networks with a distributed control plane such as BGP, in addition to the network logical configuration and data plane. Nevertheless, the proposed KG and construction procedure consider general networking concepts and can be easily adapted to SDN networks.

The authors in [3] present a network repository design that enables an efficient representation of multiple network topology abstractions for various management purposes and share valuable insights from their experiences in curating a comprehensive and evolving topology taxonomy. They propose a Multi-Abstraction-Layer Topology (MALT) representation that supports all network management phases, including design, deployment, configuration, operation, measurement, and analysis. MALT is implemented as a specialized data processing layer on top of a SQL database, unlike our repository which is directly implemented in a native GDB.

Closest to our approach is [7] where the authors put forward the idea that SDN control is fundamentally about how data is represented. They introduce a straightforward data representation of the network that includes its topology, forwarding, and control aspects, all of which are accessible to applications via a single SQL interface. Their system represents the SDN network control infrastructure within a PostgreSQL database. Here, custom SQL queries are utilized to articulate various network abstractions. A significant hurdle in implementing this method was to coordinate on a relational database multiple abstractions that have a collective impact on the same network data.

Note that none of the network repository approaches described above provides a native support of *what-if* scenarios evaluation.

3 Proposed Repository

3.1 Overview

The repository's KG comprises three types of network information: operational, behavioral, and temporary. The operational information is the network state including devices and ports with physical and logical topology, connectivity, and configuration such as VLAN. It also includes data plane information such as routing information base (RIB) and forwarding information base (FIB), and access control list (ACL). Behavioral information is inserted to model the execution of routing protocols such as BGP, or the forwarding process such as Traceroute. Temporary information represents configuration changes not yet applied to the network. This makes verification and consistency checks easier to implement, by searching for particular data patterns in the knowledge graph such as two conflicting IP addresses attached to the same interface, or a BGP peer configured on an interface that is not longer enabled. This also makes it easy to extract configuration changes to apply to network elements.

We discuss in the following the repository design from the network data acquisition, to the KG schema and its systematic creation process. We then describe the design of *what-if* scenario evaluation for BGP and Traceroute.

3.2 Data Acquisition

The construction of the data repository is a multistage process that starts by collecting data from network elements in a non-structured vendor-dependent format.

In this design, we distinguish three categories of non-structured network data: (i) dynamic data representing topology and connectivity updates consisting on port/link status, device status, and BGP peer status changes. This information can be collected through NETCONF push notifications as they change frequently and must be collected as soon as they occur. (ii) static data representing device configuration that does not change frequently such as hostname, BGP router ID and ASN, and VLANs. This data is collected periodically (via NETCONF, RESTCONF, or CLI) and reinserted in the graph. (iii) semi-static data represents device configuration that may change during a day or over the week (see network data alterations measured in [2]), such as ACLs and BGP sessions and route policies. This data is also collected periodically, but more frequently than static data and only the changed/new attributes are updated in the graph.

Collected data is first converted to a vendor-neutral format, which is then processed to create the KG as described later.

3.3 Knowledge Graph Organizing Principle

For the construction of a KG flexible enough to accommodate a range of various networking concepts, it is imperative for its data to be easily queried and understood by users and developers. Consequently, the organization of data within the KG adheres to a set of intuitive principles, outlined below.

We first introduce basic graph database concepts used in this paper. A *node* refers to a graph vertex, which can have one or multiple labels identifying the node type. A *relationship* refers to the edge connecting two vertices. A *property* is a value pair representing an attribute of a node. Each node and relationship can have multiple properties. A *path* is a sequence of nodes and relationships going from one node to another. A *path-pattern* is a path containing node labels instead of specific nodes in the graph; it is used to express the shape of the requested data.

The graph is constructed from the physical layer up to the data plane. Each inserted node must be connected to an existing node to ensure that no dangling node exists. The network data is organized in the graph according to three main patterns defining an intuitive representation of the physical network and the logical protocol stack. First, devices and their interfaces, and the physical links that connect them are represented by the *Host-LTP-Link pattern*, where LTP stands for Link Termination Point. Second, protocol endpoints are represented in their respective layers according to their implementation. That is, link-layer protocols run on top of physical interfaces, and IP-layer protocols run on top of link layer. Endpoints of the same layer that are expected to interact can be connected with a relationship expressing the protocol information; for example, IP endpoints on top of two physically linked interfaces would be connected with a relationship corresponding to an ARP table entry. We refer to this pattern by *Layer-CTP-Connection*, where CTP stands for Connection Termination Point. Depending on the layer represented, the CTP has a type such as Ethernet, IPv4, TCP, etc. Third, device configurations and data plane information must be reachable from the device via a path of one or more hops. When a configuration or a data plane information is an independent fact, it is represented by a node connected to its parent node with a relationship. For example *"a FIB route in a host"* is represented with a FibRoute node connected with a "HAS_FIB" relationship to the Host node. When a configuration or a data plane information connects two independent configurations, it is represented by a relationship between the two nodes it refers to; for example, *"an interface is member of a VLAN"* should be represented "VLAN_MEMBER" relationship between the interface and the VLAN nodes.

Figure 1 shows the KG schema generated by the GDB. It represents the existing node labels and their relationships.

3.4 Knowledge Graph Construction

To illustrate the KG creation process, we consider a simple network from which the following data is collected: device properties (hostname, BGP router ID, etc.), ports properties (name, MTU, speed, etc.), VLAN configuration, LLDP neighbors, IPv4 addresses, BGP peers, and FIB routes.

Data collected from devices is converted to a vendor-neutral format grouped by device, which is then processed to produce facts. A fact is a map representing a network information to insert in the graph; either through a new node and new relationship(s), or only a new relationship between two existing nodes. Facts are organized according to their type and insertion order. For example, facts representing the network devices and their properties are to insert before facts representing interfaces, which are to be inserted before BGP peers configuration, and so on. The fact types are given in their insertion order as follows: devices, physical ports, Ethernet endpoints, IPv4 endpoints, VLAN L3 interfaces, physical links, IP neighbors connectivity, ACL rules, BGP peers, and FIB routes.

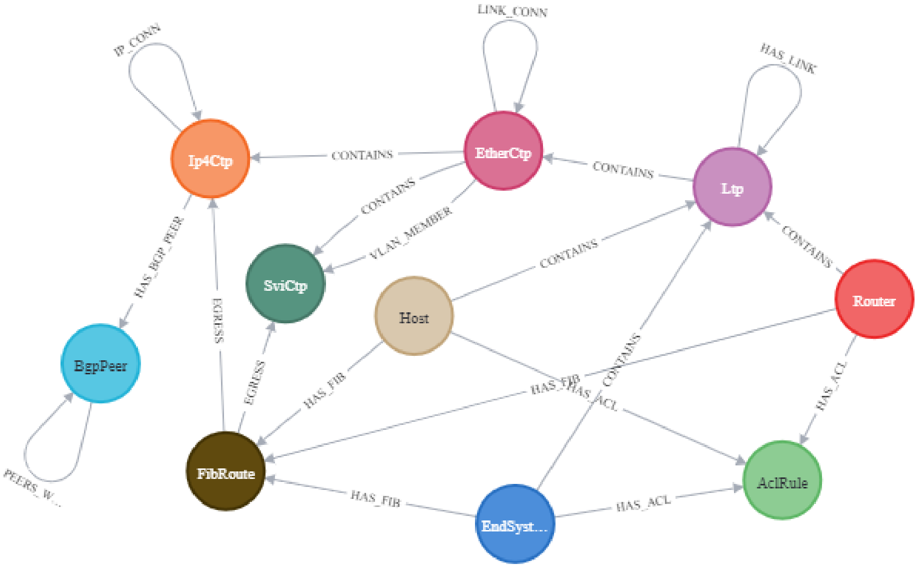


Fig. 1: KG schema visualization

Finally, the facts are converted to GDB queries. A query can insert one or multiple aggregated facts (see Section 4).

Figure 2 shows an example of parsing a BGP peer configuration from CLI output to the GDB query. The facts step is not shown since the fact is mapped directly to the GDB query.

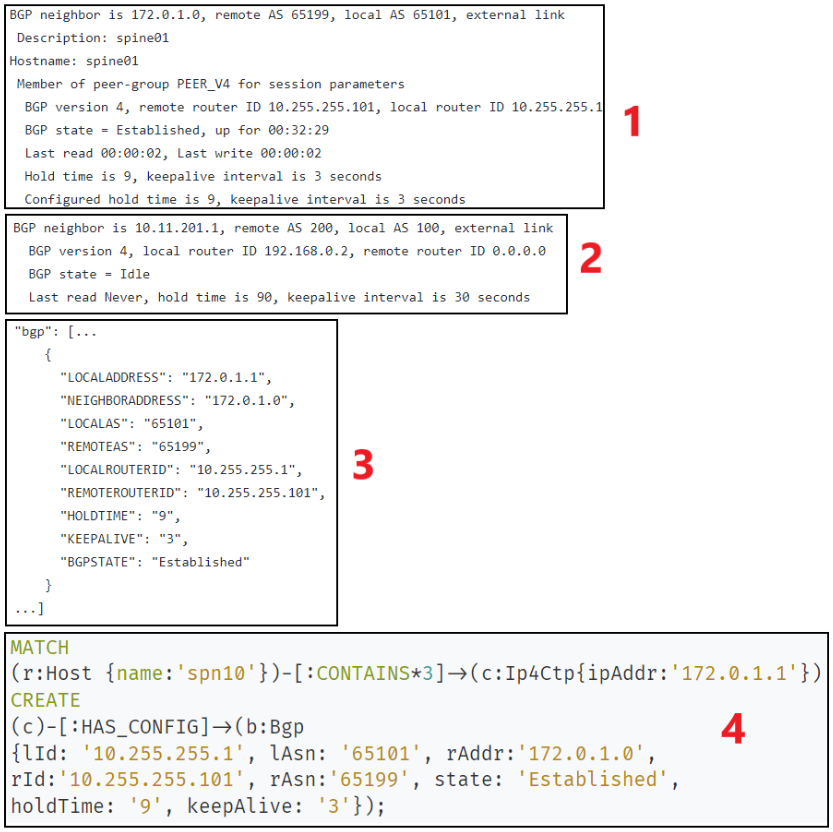
The construction algorithm[2] creates a KG representing the network operational state, with topology, configuration and data plane information. Not all network data is discussed in this paper due to space limitation. However, information such as link aggregation and sub-interfaces can be easily represented in the KG using the organizing principles introduced above.

The complete graph creation process is executed only once. After that only portions of the graph are recreated periodically for static and semi-static network data as described above. Dynamic network state changes are received individually via notification, converted to a fact and inserted in the graph.

3.5 ***What-if* support**

The proposed repository natively supports *what-if* scenario evaluation through simple simulations. We selected BGP and Traceroute to showcase this capability. BGP route policies are known to be intricate and prone to errors, which is why some of the most significant internet outages are caused by misconfigured route policies that inadvertently leak or accept routes they shouldn't. Traceroute

[2] Not provided here due to space limitations. The reader can refer to the smart repository code for the complete KG creation process.



```
BGP neighbor is 172.0.1.0, remote AS 65199, local AS 65101, external link
 Description: spine01
Hostname: spine01
 Member of peer-group PEER_V4 for session parameters
  BGP version 4, remote router ID 10.255.255.101, local router ID 10.255.255.1
  BGP state = Established, up for 00:32:29
  Last read 00:00:02, Last write 00:00:02
  Hold time is 9, keepalive interval is 3 seconds
  Configured hold time is 9, keepalive interval is 3 seconds
```

```
BGP neighbor is 10.11.201.1, remote AS 200, local AS 100, external link
  BGP version 4, local router ID 192.168.0.2, remote router ID 0.0.0.0
  BGP state = Idle
  Last read Never, hold time is 90, keepalive interval is 30 seconds
```

```
"bgp": [...
    {
      "LOCALADDRESS": "172.0.1.1",
      "NEIGHBORADDRESS": "172.0.1.0",
      "LOCALAS": "65101",
      "REMOTEAS": "65199",
      "LOCALROUTERID": "10.255.255.1",
      "REMOTEROUTERID": "10.255.255.101",
      "HOLDTIME": "9",
      "KEEPALIVE": "3",
      "BGPSTATE": "Established"
    }
...]
```

```
MATCH
(r:Host {name:'spn10'})-[:CONTAINS*3]->(c:Ip4Ctp{ipAddr:'172.0.1.1'})
CREATE
(c)-[:HAS_CONFIG]->(b:Bgp
{lId: '10.255.255.1', lAsn: '65101', rAddr:'172.0.1.0',
rId:'10.255.255.101', rAsn:'65199', state: 'Established',
holdTime: '9', keepAlive: '3'});
```

Fig. 2: Simplified BGP configuration parsing example. 1: CLI output from a SONiC device, 2: CLI output from a OCNOS device, 3: Vendor-independent format, 4: GDP query

provides detailed information about the paths taken by a specified flow through the network, which is a powerful capability for exploring and testing network behavior.

The BGP process is a sequence of queries that modify the KG with nodes and relationships that describe BGP behavior. A BGP peer created on an interface of a router is represented by BgpPeer node connected to the corresponding IPv4 CTP node with a 'HAS_BGP_PEER' relationship. Two BGP peers that established a session are connected with a 'PEERS_WITH' relationship. A BGP route update is represented with BgpUpdate node and contains the announced prefix, origin Autonomous System (AS), next hop, AS path[3]. BgpUpdate is connected to the BGP peer through which it is sent (resp. received) via a 'BGP_UPDATE_TO_SEND' (resp. 'RECEIVED_BGP_UPDATE') relationships. A clause of the route policy is represented with a Filter node connected to the device to which it belongs with a 'HAS_FILTER' relationship. A filter can be inbound or outbound, and contains a prefix and/or origin AS to match the route, a priority, a permit or deny action, and optional values for community and local preference to modify the route.

The Traceroute process is similar to that of BGP. The queries start by creating a Packet node connected to the source LTP from which it originates (Usually the loopback interface of a device). The packet contains flow information such as destination IP address, destination port, and protocol. The packet is virtually forwarded by finding the best matching route at each node, applying the inbound and outbound ACL rules, and sending the packet to the output LTP. Each forwarding step creates a relationship corresponding to the action between the packet and the corresponding node: LTP, ACL rule, and FIB route. This process continues until the destination is reached, no route is found for the packet, or the packet is blocked by an ACL rule.

4 Evaluation

The proposed repository is implemented on Neo4j, a popular GDB with an SQL-like query language.

To evaluate the feasibility of the proposed network repository, we measure its time and space efficiency, and we demonstrate two *what-if* scenarios by verifying BGP route policies and analyzing forwarding with a virtual Traceroute.

4.1 Performance Evaluation

When handling network data with a KG, the most critical evaluation criteria are the size of the graph on disk, the time to execute a query on the repository, and the graph creation time.

We emulate a leaf-spine network with 13 leaves, 10 spines, and 4 servers each connected on a different VLAN. The links between spines and leaves are Layer 3.

[3] Not all BGP features are included in this demonstration.

The routing is realized with BGP. All nodes use SONiC as the network operating system.

The repository is implemented with Neo4j 4.4.25 Enterprise Edition, and the evaluation is performed on a server with a Core i5 8th generation CPU with 16 Gb of RAM.

The processing stage takes about 500 ms and includes the generations of facts and queries, with each fact mapped to one query. The graph creation takes about 106 seconds for a total number of 8466 queries representing 8466 facts, which corresponds to an average execution time of 13 ms per query. The execution time of queries is stable during the graph creation process because, as per the creation procedure, the queries do not need to process a large portion of the graph to insert a node/relationship. Consequently, the number of queries executed to create the graph can be reduced by aggregating multiple related facts and insert them in one query. Given the KG organizing principle (Section 3.3) and the creation process (Section 3.4), the more is known about the represented network concepts (e.g., topology, routing protocols, etc.) the more facts can be aggregated. After aggregating the facts of ports and Ethernet/IPv4 endpoints with each device creation query, the number of queries dropped by 39.6% (3359 queries) lowering the total graph creation time to 66 seconds while producing the same graph. Adding the BGP peers to this aggregation dropped the initial queries number by 42.6% (3610 queries) lowering the graph reaction time to 63 seconds. Finally, the aggregation of ACL rules facts belonging to the same ACL table, reduced the number of queries by 43.9% and the graph creation time to 61 seconds.

The KG of this network contains 7974 nodes and 17193 relationships, with a size on disk of around 3 MB. Therefore, on average, for each created node 3 relationships are created. Once the graph is established, the average query execution time is no more than 29 ms. This query execution time can represent the time to insert a notification about a dynamic network state change.

Note that a GDB system such as Neo4j is capable of supporting hundred millions of nodes and relationships.

4.2 *What-if* scenario with BGP

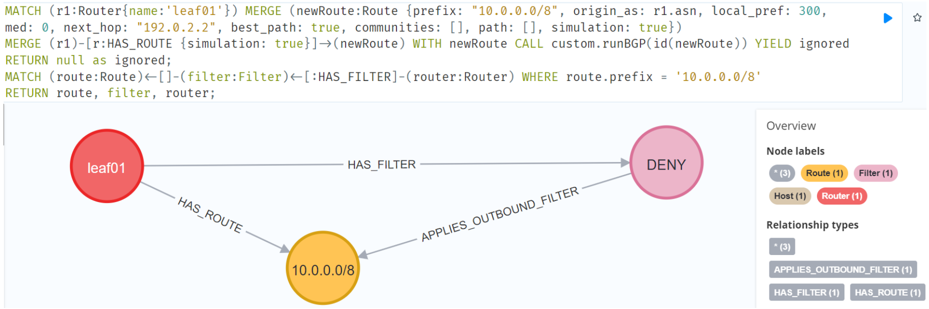
We consider an example leaf-spine network with 4 leaves, 4 spines, and 8 servers each connected on a different VLAN. The topology includes 2 border routers and 2 firewalls[4]. For simplicity, we assume route updates are allowed only between leaf01, leaf02, spine01, and spine02.

In the following, we run the simulated BGP behavior to analyse two examples of BGP route policies configuration, before applying them to the network. The filters in routes are configured in such a way that on only routes for prefix 10.11.200.0/24 are accepted and propagated.

In the first example, we mimic the advertisement of a route for the prefix 10.0.0.0/8 from leaf01 that will not be propagated. Figure 3 shows the queries

[4] This network and "what-if" examples are available in the smart repository code

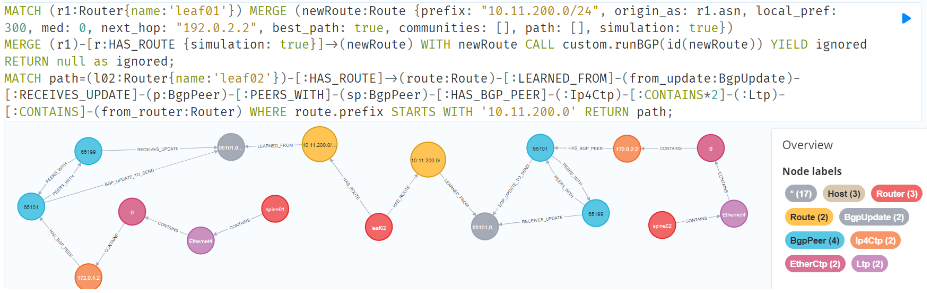
that runs the *what-if* scenario and inspect the results to find where the route has been blocked.



```
MATCH (r1:Router{name:'leaf01'}) MERGE (newRoute:Route {prefix: "10.0.0.0/8", origin_as: r1.asn, local_pref: 300,
med: 0, next_hop: "192.0.2.2", best_path: true, communities: [], path: [], simulation: true})
MERGE (r1)-[r:HAS_ROUTE {simulation: true}]→(newRoute) WITH newRoute CALL custom.runBGP(id(newRoute)) YIELD ignored
RETURN null as ignored;
MATCH (route:Route)←[]-(filter:Filter)←[:HAS_FILTER]-(router:Router) WHERE route.prefix = '10.0.0.0/8'
RETURN route, filter, router;
```

Fig. 3: Blocked route

In the second example, we mimic the advertisement of a route for the prefix ”10.11.200.0/24 from leaf01 that will be propagated to all routers. Figure 4 shows the queries that runs the scenario and find the propagation path of the routes installed at leaf02.



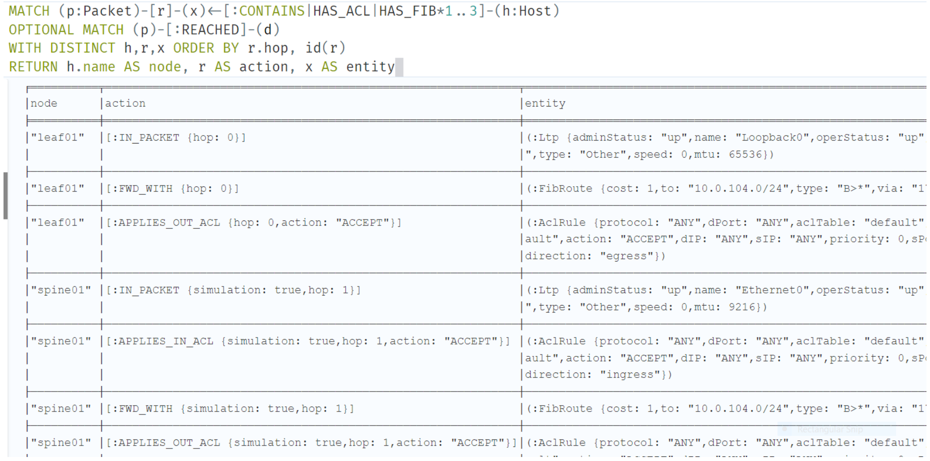
```
MATCH (r1:Router{name:'leaf01'}) MERGE (newRoute:Route {prefix: "10.11.200.0/24", origin_as: r1.asn, local_pref:
300, med: 0, next_hop: "192.0.2.2", best_path: true, communities: [], path: [], simulation: true})
MERGE (r1)-[r:HAS_ROUTE {simulation: true}]→(newRoute) WITH newRoute CALL custom.runBGP(id(newRoute)) YIELD ignored
RETURN null as ignored;
MATCH path=(l02:Router{name:'leaf02'})-[:HAS_ROUTE]→(route:Route)-[:LEARNED_FROM]-(from_update:BgpUpdate)-
[:RECEIVES_UPDATE]-(p:BgpPeer)-[:PEERS_WITH]-(sp:BgpPeer)-[:HAS_BGP_PEER]-(:Ip4Ctp)-[:CONTAINS*2]-(:Ltp)-
[:CONTAINS]-(from_router:Router) WHERE route.prefix STARTS WITH '10.11.200.0' RETURN path;
```

Fig. 4: Propagation path of a route

4.3 *What-if* scenario with Traceroute

In this example, we simulate the forwarding process to find the path taken by leaf01 router to reach the DNS Server 10.0.104.104 (server04). The following is the query that runs the *what-if* scenario and Figure 5 shows the results.

```
CALL custom.traceroute('leaf01', '10.0.104.104', '10.0.104.0/24', 33434, 'UDP')
YIELD ignored RETURN ignored
```



```
MATCH (p:Packet)-[r]-(x)←[:CONTAINS|HAS_ACL|HAS_FIB*1..3]-(h:Host)
OPTIONAL MATCH (p)-[:REACHED]-(d)
WITH DISTINCT h,r,x ORDER BY r.hop, id(r)
RETURN h.name AS node, r AS action, x AS entity
```

```
|node      |action                                                     |entity
|"leaf01"  |[:IN_PACKET {hop: 0}]                                      |(:Ltp {adminStatus: "up",name: "Loopback0",operStatus: "up"
|          |                                                           |",type: "Other",speed: 0,mtu: 65536})
|"leaf01"  |[:FWD_WITH {hop: 0}]                                       |(:FibRoute {cost: 1,to: "10.0.104.0/24",type: "B>*",via: "1
|"leaf01"  |[:APPLIES_OUT_ACL {hop: 0,action: "ACCEPT"}]               |(:AclRule {protocol: "ANY",dPort: "ANY",aclTable: "default"
|          |                                                           |ault",action: "ACCEPT",dIP: "ANY",sIP: "ANY",priority: 0,sP
|          |                                                           |direction: "egress"})
|"spine01" |[:IN_PACKET {simulation: true,hop: 1}]                     |(:Ltp {adminStatus: "up",name: "Ethernet0",operStatus: "up"
|          |                                                           |",type: "Other",speed: 0,mtu: 9216})
|"spine01" |[:APPLIES_IN_ACL {simulation: true,hop: 1,action: "ACCEPT"}] |(:AclRule {protocol: "ANY",dPort: "ANY",aclTable: "default"
|          |                                                           |ault",action: "ACCEPT",dIP: "ANY",sIP: "ANY",priority: 0,sP
|          |                                                           |direction: "ingress"})
|"spine01" |[:FWD_WITH {simulation: true,hop: 1}]                      |(:FibRoute {cost: 1,to: "10.0.104.0/24",type: "B>*",via: "1
|"spine01" |[:APPLIES_OUT_ACL {simulation: true,hop: 1,action: "ACCEPT"}]|(:AclRule {protocol: "ANY",dPort: "ANY",aclTable: "default"
```

Fig. 5: Traceroute results

5 Conclusion

This paper demonstrated a smart network repository based on a general-purpose native GDB. The repository can represent multiple levels of network abstraction, including topology, connectivity, configuration, and forwarding, in an organized and comprehensible manner. A systematic method is introduced to construct the repository's KG which contributes to the common understanding of network data, and creates a interoperable and extensible network repository, thus increasing the accessibility and usability of the network data.

The feasibility of a general-purpose GDB as a network repository was evaluated, establishing it as a viable solution for Network Management Systems (NMS) and SDN controllers.

The repository's potential extends beyond just storing and managing network data. We assessed its potential for running simple simulations to quickly evaluate what-if scenarios through queries. This capability, demonstrated for BGP route policy and Traceroute, opens up new avenues for smart network repositories that natively support analysis and verification.

Disclaimer

Any mention of commercial products or reference to commercial organizations is for information only; it does not imply recommendation or endorsement by NIST, nor does it imply that the products mentioned are necessarily the best available for the purpose.

References

1. Berde, P., Gerola, M., Hart, J., Higuchi, Y., Kobayashi, M., Koide, T., Lantz, B., O'Connor, B., Radoslavov, P., Snow, W., Parulkar, G.: Onos: Towards an open, distributed sdn os. In: Proceedings of the Third Workshop on Hot Topics in Software Defined Networking. p. 1–6. HotSDN '14, Association for Computing Machinery, New York, NY, USA (2014). https://doi.org/10.1145/2620728.2620744, https://doi.org/10.1145/2620728.2620744
2. Hong, H., Wu, Q., Dong, F., Song, W., Sun, R., Han, T., Zhou, C., Yang, H.: Netgraph: An intelligent operated digital twin platform for data center networks. In: Proceedings of the ACM SIGCOMM 2021 Workshop on Network-Application Integration. p. 26–32. NAI'21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3472727.3472802, https://doi.org/10.1145/3472727.3472802
3. Mogul, J.C., Goricanec, D., Pool, M., Shaikh, A., Turk, D., Koley, B., Zhao, X.: Experiences with modeling network topologies at multiple levels of abstraction. In: 17th Symposium on Networked Systems Design and Implementation (NSDI) (2020), https://www.usenix.org/conference/nsdi20/presentation/mogul
4. Quinn, R., Kunz, J., Syed, A., Breen, J., Kasera, S., Ricci, R., Van der Merwe, J.: Knownet: Towards a knowledge plane for enterprise network management. In: NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium. pp. 249–256 (2016). https://doi.org/10.1109/NOMS.2016.7502819
5. Sung, Y.W.E., Tie, X., Wong, S.H., Zeng, H.: Robotron: Top-down network management at facebook scale. In: Proceedings of the 2016 ACM SIGCOMM Conference. p. 426–439. SIGCOMM '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2934872.2934874, https://doi.org/10.1145/2934872.2934874
6. Vilalta, R., Muñoz, R., Casellas, R., Martínez, R., López, V., de Dios, O.G., Pastor, A., Katsikas, G.P., Klaedtke, F., Monti, P., Mozo, A., Zinner, T., Øverby, H., Gonzalez-Diaz, S., Lønsethagen, H., Pulido, J.M., King, D.: Teraflow: Secured autonomic traffic management for a tera of sdn flows. In: 2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC6G Summit). pp. 377–382 (2021). https://doi.org/10.1109/EuCNC/6GSummit51104.2021.9482469
7. Wang, A., Mei, X., Croft, J., Caesar, M., Godfrey, B.: Ravel: A database-defined network. In: Proceedings of the Symposium on SDN Research. SOSR '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2890955.2890970, https://doi.org/10.1145/2890955.2890970
8. Zhou, Q., Gray, A.J.G., McLaughlin, S.: Toco: An ontology for representing hybrid telecommunication networks. In: Hitzler, P., Fernández, M., Janowicz, K., Zaveri, A., Gray, A.J., Lopez, V., Haller, A., Hammar, K. (eds.) The Semantic Web. pp. 507–522. Springer International Publishing, Cham (2019)
9. Zhou, Q., Gray, A.J.G., McLaughlin, S.: Towards A knowledge graph based autonomic management of software defined networks. CoRR **abs/2106.13367** (2021), https://arxiv.org/abs/2106.13367
10. Zhu, Y., Chen, D., Zhou, C., Lu, L., Duan, X.: A knowledge graph based construction method for digital twin network. In: 2021 IEEE 1st International Conference on Digital Twins and Parallel Intelligence (DTPI). pp. 362–365 (2021). https://doi.org/10.1109/DTPI52967.2021.9540177