GOX: Towards a Scalable Graph Database-Driven SDN Controller

Fetia Bannour
ENSIIE & SAMOVAR, Evry, France
fetia.bannour@ensiie.fr

Stefania Dumbrava
ENSIIE & SAMOVAR, Evry, France
stefania.dumbrava@ensiie.fr

Alex Danduran--Lembezat
ENSIIE, Evry, France
alex.danduran–lembezat@ensiie.fr

***Abstract*—The Software-Defined Networking (SDN) paradigm relies on decoupling the control and data planes, and logically centralizing SDN control to enable direct network programming via open interfaces. New abstractions are thus needed in a bid to rethink the traditional networking approach and create new opportunities for management and automation. We demonstrate the GOX controller, proposing a *novel graph abstraction* of the network topology in real time using the scalable Neo4j graph database. Our proof-of-concept was evaluated for a forwarding application designed for GOX. Compared to POX's model, GOX shows better performance and scalability on synthetic topologies and real-world topologies from the Internet Topology Zoo.**

I. Introduction

Traditional networks are grappling with the explosion of demand, mobile trends, and Cloud Computing. They are ill-suited to meet the requirements of today's users, enterprises, and carriers, given their static nature, vertically-integrated design, and distributed control logic. The Internet has thus evolved into an ossified bundle of closed devices and hardware-centric protocols, and has become extremely hard to manage [1]. SDN [2] breaks the vertical integration of legacy networks, separating the network's control logic from the underlying hardware. Thus, it promises to improve network management, automation and innovation. In SDN, network intelligence is logically centralized in software controllers that ensure a globally consistent network view to be leveraged by applications [3]. The controllers maintain the network state and handle communications between applications and devices through open Application Programming Interfaces (APIs).

That said, SDN introduces programmatic interfaces and provides them with abstractions to improve resource utilization, and ease application development and network management. There has been a growing interest in designing proper network topology *abstractions* that abstract the complexity of the underlying hardware. So far, extensive efforts have been devoted to developing a suitable abstraction for the Southbound API (allowing interactions between the control and data planes), as OpenFlow has become its widely-accepted standard [4].

But, little attention has been paid to studying abstractions for the *Northbound API*, which allows the communication between the control and application planes, and the *East-Westbound API*, which ensures the communication between the controllers in a physically-distributed SDN control plane [1]. Particularly, standardizing these interfaces partly relies on providing a proper abstraction of the network topology (the global network view). The representation of this abstraction is key to rethinking the network architecture and management.

Recent works have shown that graph database models offer performance and scalability benefits for today's modern applications [5]. Inspired by these works, we leverage graph data models to build a dynamic graph-based network representation and implement it using modern graph database techniques.

Specifically, we propose a proof-of-concept methodology for building a graph model abstraction of the network which stores the network state in real time. Our graph-oriented controller, which we refer to as GOX, augments the popular and developer-friendly POX controller [6] with a graph-based module implemented with the scalable Neo4j graph database [7]. We implement a custom forwarding application and use it to showcase GOX's superior performance compared to POX.

The goal of this demo is to illustrate the GOX system which allows users to engage in the following scenarios:

- *visualization* and *exploration* of a network topology in the Neo4j graph database system;
- *execution* of a network application leveraging the expressiveness of the Cypher graph query language;
- *inspection* of the performance benefits of using our data-centric SDN controller model over the POX baseline.

II. Related work

Previous works on SDN management explored the concept of leveraging graph databases [8], [9]. Souza et al. [5] introduce the NML semantic network model implemented in Neo4j. They encode the NetGraph SDN primitives as queries in SQL and Cypher (Neo4j's query language), showing the superior efficiency of path queries [10] on synthetic topologies. Unlike our network model, NML does not use the full expressiveness of Neo4j's property graph model, as edges are unattributed. Also, the method is not implemented in a controller and is only tested on synthetic topologies. Ravel [11] and Gavel [12], [13] are proof-of-concept controllers that implement an SDN architecture within a database. For Ravel, network abstractions correspond to SQL views, while Gavel treats them as graph instances. While Gavel is shown to perform better than Ravel, it is not compared with other SDN controller platforms.

III. GOX System Architecture

A. Workflow Diagram

The interactions between the controller, the graph database, and the SDN applications are captured in Fig. 1.

1) Processing network updates: SDN network updates (host joining or switch disconnecting) trigger GOX events. These can either be processed using topology information from the graph database, or can modify the latter. For example,

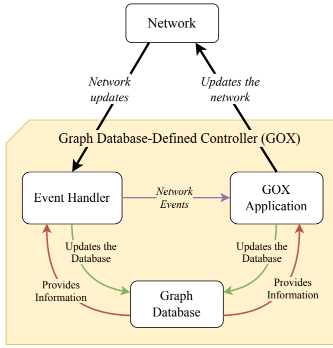


Fig. 1: Workflow diagram of the GOX controller

when a host joins the network, the event handler checks whether it is registered and, if not, it adds it to the database.

2) GOX applications: Network events can be sent to GOX applications. These need to know the network's current state, as recorded in the database providing them with information. Based on the introduced logic, applications might need to update the database, to account for how they will update the network. For example, a forwarding application might require information about a particular path between two hosts from the database. If it wants to create/modify that path, it should modify/update the database accordingly and send packets to the switches on that path to install the new forwarding rules.

B. Network Modeling

We abstract network topologies (interconnected hosts and switches) using the *property graph model*, i.e., a directed multi-graph, with property lists attached to nodes and edges. We represent hosts and switches as nodes and their links as

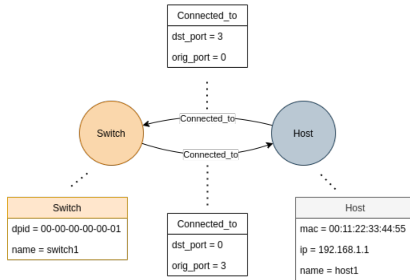


Fig. 2: Example of a simple network under our network model

edges. To identify nodes, we use the DataPath ID (DPID) for switches and the MAC address for hosts. Links contain port information and, since Neo4j relationships are oriented, we distinguish the origin and destination ports, as in Fig. 2.

IV. GOX System Design

A. GOX : A Novel Graph-Oriented POX

GOX leverages the key advantages of graph databases for modeling network abstractions and designing applications. Indeed, representing applications as graph queries greatly *simplifies programmability*, *reduces orchestration complexity* (as all applications access the same network state), and ensures *efficient execution*, due to their custom evaluation algorithms.

1) GOX's Database Component: The *gox_db* component adds/deletes hosts, switches, links and extracts property values.

2) GOX's Network Component: The *gox_network* component is crucial to adapting GOX's topology model to the network dynamicity. As such, it responds to events raised by components like `openflow.discovery` and `host_tracker`, and updates the database accordingly. When `openflow.discovery` raises a `LinkEvent`, it means a connection update (created/removed) between a pair of switches is detected. *gox_network* catches that event and recovers the stored information: the DPID of both switches and the ports on which the link is added/removed. The `ConnectionUp` and `ConnectionDown` events are raised when a switch is connected/disconnected. From these, we can extract the switch's DPID and update the database, using the *gox_db* component. The `host_tracker` raises a `HostEvent` when the status of a host changes (joined, left, moved). Then, GOX can retrieve the host's MAC address, the DPID of the switch it is connected to, and the port.

B. Use case : A GOX Forwarding Application

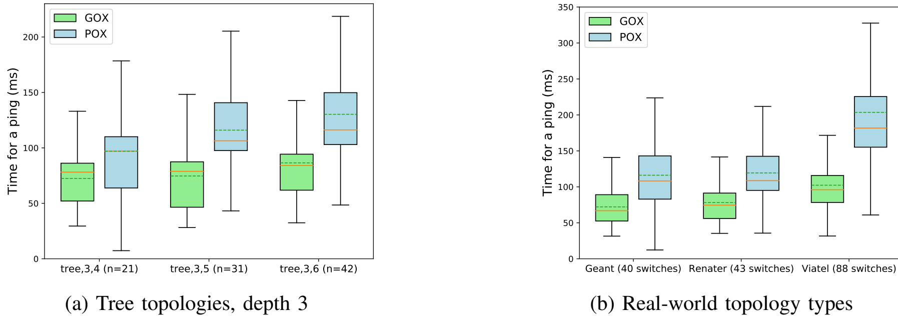
To study GOX's performance, we developed an application leveraging Neo4j's graphs. It forwards packets between hosts, along the shortest-path (Dijkstra's algorithm). Initially, no forwarding rules are installed. When switches send packets to the controller, the `PacketIn` event is triggered and GOX retrieves the switch's DPID, the source MAC address, the target host, and the transfer type. If the sending and receiving hosts are unknown, it broadcasts the packet until it learns the location of the destination host. Since GOX uses POX's `host_tracker`, if a host is inactive for a while, it times out and gets deleted. When hosts are known, we execute a Cypher query (see Listing 1) to create the path storing the switch DPIDs and ports, linking the sending and receiving hosts. Our application sends OpenFlow messages to path switches, using their DPIDs. Since POX's flow system uses OpenFlow's packet-matching pattern, we can define the action to be taken when a packet is matched. We communicate the identified destination port for each path switch, installing the path. This can be used by further packets, without calling the controller.

```
MATCH (h1:Host {mac: "da:25:b5:f7:9a:3e"})
MATCH (h2:Host {mac: "95:d5:e5:d0:b7:4f"})
MATCH
p = shortestPath((h1)-[:Connected_to*]->(h2))
WITH h1, h2, p,
[n in nodes(p)[1..-1]| n.dpid] AS switches,
[r in relationships(p)[1..] | r.orig_port] AS
    out_ports,
[r in relationships(p)[..-1]| r.dst_port] AS
    in_ports
MERGE (h1)-[p1:Path_to]->(h2)
ON CREATE SET p1.switches=switches,
              p1.out_ports=out_ports,
              p1.in_ports=in_ports
```

Listing 1: Cypher query example for our application

V. Demonstration Overview

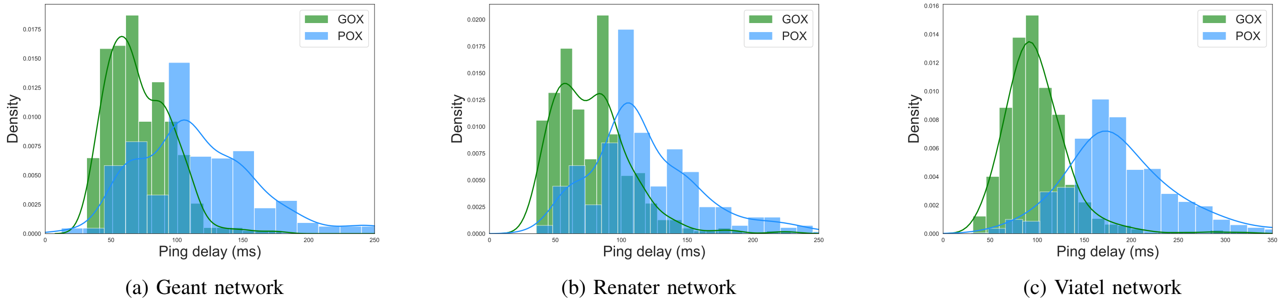
We demonstrate the superior performance of GOX compared to POX through several experiments. These experiments are performed on a VM running an Ubuntu 20.04.2 LTS



(a) Tree topologies, depth 3

(b) Real-world topology types

Fig. 3: Forwarding performance for various topology types (with an increasing number of nodes (n))



(a) Geant network

(b) Renater network

(c) Viatel network

Fig. 4: Forwarding delay distributions for different real-world topologies

server, with 8 GB of RAM and 8 CPU cores. We used POX's "gar-experimental" version, Mininet 2.3.0, and Neo4j Server 4.1.10. Our evaluation measures the Round-Trip Time for host message exchanges, by allowing a random host to ping all others. Such ICMP requests rely on our application, which creates paths between hosts in the controller's graph database and installs them in the switches' flow tables.

On small network topologies, from (tree_2_2) to (tree_2_6), GOX's performance is comparable to POX's. However, on larger, depth-3 tree topologies, ranging from (tree_3_2) to (tree_3_6), GOX outperforms POX (see Fig. 3a). We also notice that GOX's performance is relatively more steady as the network size increases, highlighting the better scalability of our model. These experiments also reveal that the controllers' performance depends on the network topology type and size.

On real-world topologies (Geant, Renater, and Viatel), with up to $\approx$ 90 switches, GOX's mean ping time is lower than POX's (see Fig. 3b). The percentage-wise time gain with GOX is up to $\approx$ 50% for the Viatel topology (88 switches). Also, as seen in Fig. 4, the mean and standard deviation parameters for a ping delay distribution are lower for GOX. This highlights the stability of our approach on larger, real-world networks, confirming the observations on tree topologies.

For both synthetic and real-world topologies, GOX has a superior performance on larger networks and a more predictable forwarding time. Its scale-out capabilities come from the use of a graph database, as these are optimized for large topologies.

VI. Conclusion

We demonstrate the GOX graph-oriented SDN controller and the inherent advantages of implementing it with the Neo4j graph database. Its evaluation, on a custom forwarding application, is shown to yield better performance results compared to POX. Our experiments confirm that graph databases offer a natural real time representation of the network topology for the SDN control plane. Together with the reduced complexity and latency of graph queries for network applications, this shows the promise of leveraging such technologies to improve SDN management and automation. In future work, we aim to leverage the property graph model for controller interoperability. GOX is accessible at https://github.com/ExcessDrive/GOX.

References

[1] F. Bannour, S. Souihi, and A. Mellouk, "Distributed SDN control: Survey, taxonomy, and challenges," *IEEE Commun. Surv. Tutorials*, vol. 20, no. 1, pp. 333–354, 2018.

[2] M. He, M. Huang, and W. Kellerer, "Optimizing the flexibility of SDN control plane," in *NOMS*. IEEE, 2020, pp. 1–9.

[3] V. Huang, G. Chen, P. Zhang, and et al., "A scalable approach to SDN control plane management: High utilization comes with low latency," *IEEE Trans. Netw. Serv. Manag.*, vol. 17, no. 2, pp. 682–695, 2020.

[4] D. E. Sarmiento, A. Lebre, L. Nussbaum, and A. Chari, "Decentralized SDN control plane for a distributed cloud-edge infrastructure: A survey," *IEEE Commun. Surv. Tutorials*, vol. 23, no. 1, pp. 256–281, 2021.

[5] T. D. Souza, C. E. Rothenberg, M. A. S. Santos, and L. B. de Paula, "Towards semantic network models via graph databases for SDN applications," in *EWSDN*. IEEE Computer Society, 2015, pp. 49–54.

[6] Pox controller. [Online]. Available: https://github.com/noxrepo/pox/

[7] Neo4j. [Online]. Available: https://neo4j.com/

[8] P. S. Rivera, M. Hayashida, J. Griffioen, and Z. Fei, "Dynamically creating custom SDN high-speed network paths for big data science flows," in *PEARC*. ACM, 2017, pp. 59:1–59:4.

[9] B. Halder, M. S. Barik, and C. Mazumdar, "A graph based formalism for detecting flow conflicts in software defined network," in *ANTS*. IEEE, 2017, pp. 1–6.

[10] A. Bonifati and S. Dumbrava, "Graph queries: From theory to practice," *SIGMOD Rec.*, vol. 47, no. 4, pp. 5–16, 2018.

[11] A. Wang, X. Mei, J. Croft, M. Caesar, and B. Godfrey, "Ravel: A database-defined network," *SOSR*, 2016.

[12] O. L. Barakat, D. Koll, and X. Fu, "Gavel: Software-defined network control with graph databases," in *ICIN*. IEEE, 2017, pp. 279–286.

[13] O. L. Barakat, D. Koll, and X. Fu, "Gavel: A fast and easy-to-use plain data representation for software-defined networks," in *IEEE Trans. Netw. Serv. Manag.*, vol. 16, no. 2, 2019, pp. 606–617.