Link Validation and Usability in a Database-Defined Network

Keith Hudock
Department of Computer & Information Sciences
Temple University
Philadelphia, PA, USA 19121
Email: keith.hudock@temple.edu

Thomas Smith
Department of Computer & Information Sciences
Temple University
Philadelphia, PA, USA 19121
Email: thomas.robert.smith@temple.edu

Anduo Wang
Department of Computer & Information Sciences
Temple University
Philadelphia, PA, USA 19121
Email: adw@temple.edu

***Abstract*—Software-defined networks (SDN's) rely on a centralized controller in order to manage the nodes contained within a network. The validation of nodes found in the network and connections between them is vital to maintaining a functioning network. As SDN's evolve over time, so should their methods for validation. Most SDN's are run within a command terminal. As this is acceptable in a research environment, efforts must be made to implement a user friendly design for it to be used in the real world.**

***Index Terms*—Ravel, controller, Software-Defined Networks, application, links**

I. Introduction

Ravel is an SDN controller. Like most other SDN's, Ravel runs through a Linux command terminal. At this moment in time, since traditional network management is still widely used, SDN's are still being developed to make it easier to manage a network. With this in mind, using a terminal interface can be intimidating and unclear to someone who has never used a terminal window. Ravel can become more user-friendly by minimizing the amount of files that can load a network topology and by allowing user inputs to determine the size and structure of the network.

What makes Ravel unique is that it is a database-defined controller, meaning that it leverages tables created in PostgreSQL to manage a network. When adding a flow through the routing application, the configuration table (cf) is updated to show each node in the flow and the node following it by using Dijkstra's algorithm to find the shortest path vector between hosts. This process does not validate whether two hosts are connected, and poses a problem when establishing a flow in the network. In order for this problem to be fixed, an application must be developed to correct the error and integrity constraints in tables can automatically detect problems with network links and move flows to the appropriate table.

II. Background

Ravel is built with Python 2 and PostgreSQL. It comes with several base tables that help to manage a network. There is a host table to see all of the hosts in a network, a switches table to see all of the switches in a network, and a nodes table that shows a combination of the host and switches table. Other base tables include the topology table (tp), the configuration table, and the reachability matrix (rm). Tp shows all of the nodes in a network and which node it is connected to. It can detect if there is a host and if the link between nodes is active or not. Cf shows the flow of a network by identifying the shortest path from host to host. Rm stores flows, keeps track of its source and destination, and whether or not the flow passes through a firewall. These tables can be altered through standard SQL commands, PostgreSQL commands, and Python functions used through PostgreSQL. These alterations can be done manually, or automatically through integrity constraints. Integrity constraints can be triggered upon a specific action to a table, such as an insert or a delete. Data can then be changed instead of or also changed based on the parameters of the SQL and Python function. The understanding of integrity constraints and these base tables drives the efforts of this work.

III. Problem Statement

One of the biggest problems that the Ravel system has is a lack of validation for a proper flow between hosts, which comes from a lack of validation of links between nodes in a network.

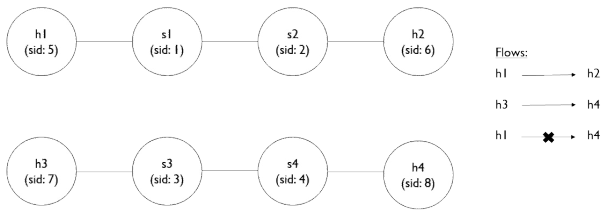

Fig. 1. Test network topology

Fig. 2.

Figure 1 shows a network in which there are four total hosts. The first two hosts are connected to each other (h1 and h2) and the other two (h3 and h4) are connected, but these combinations of connected nodes are not connected to one another. Therefore, a flow can be added from h1 to h2, h3 to h4. but not from h1 to h4. If a flow is added from h1 to h4, an error will not be thrown. Instead, the program will run, but the cf table will not be updated (Fig.2), showing that the flow was not added properly. The problem arises from the fact that there is no way to check for valid links. This can be a major problem if a node were to crash and the problem was not resolved in a timely manner. This can lead to bad flows within the network.

Another problem in Ravel stems from its user interface. While the terminal window works well in an academic environment, it lacks a user interface that makes it easier to run the programs necessary to manage the network. Furthermore, Ravel is heavily dependent on static network topologies. A user has no way of creating their own network topology and must rely on models built in by Mininet. Ravel does support custom network topologies and has some built into the controller (ex. diamond), but these networks are static as well. As networks are constantly changing in a dynamic matter, it is apparent that Ravel must also be able to change dynamically.

IV. Solutions Proposed

A. Link Validation

1) Integrity Constraints: There are several ways to approach correcting the link validation problem. One such solution is by developing one or multiple integrity constraints in order to verify links between nodes. An integrity constraint can be a command that executes upon the action of another table. For example, in Ravel, when a flow is added through the routing application, the reachability matrix is updated. The firewall application works alongside the routing application. When data is inserted into the reachability matrix, an integrity constraint found in the firewall SQL file will execute and determine if the flow added is a whitelisted flow, and if so, add said flow into the firewall policy access control list table. An integrity constraint could be developed in order to detect whether a flow added to the reachability matrix is properly linked and execute as it normally would. If that is not the case, and the flow added is not properly linked, the flow may be added to a separate table that is specifically made to contain bad flows.

2) Develop an Application: Another solution proposed is the development of an application that can add and delete links once loaded onto the Ravel controller. This application will be able to add or delete links between nodes based on the ID number assigned to each node in the network. If a link is already established between two nodes, it will not re-add the link, and return a message stating that the link already exists. The delete function will work similarly to the add function. If a link does not exist and an attempt to delete such a link exists, it will return a message that the link does not exist, therefore can not be deleted. This application would be made to manually repair or create new paths from one end to the other.

B. User Interface

To make Ravel more user friendly , a single file could be made to contain all of the network topologies. Furthermore, these topologies can be made to take parameters, such as the type of topology and size of the network. For example, a k-ary fat-tree topology is a common data

center network topology. There are four layers to a fat-tree topology, the edge, aggregate and core layers. The size of the fat-tree is determined by the number of pods (k) found in the network. The pods can be taken as a parameter entered by a user, and build the topology based on the number of pods the user inputs. To load any other topologies, a user would simply input the name of the topology, and input any other parameters that would be associated with building the topology.

V. Experiments and Results

A. Integrity Constraints

This method of verifying valid links by using integrity constraints proved to be a challenging task. Ravel's table are heavily intertwined with one another. The relationship between tables and developing a constraint to properly work with existing constraints made it difficult to finish a working integrity constraint. This effort was not fruitless, however. After the studying the controller, the most promising method of achieving a working integrity constraint comes from the topology table. The topology table shows each node in the network and each node they are connected to. In the future, a relationship can be developed between the reachability matrix and topology table to check if the flow added to the reachability matrix can be reached by checking the results found in the topology table.

B. Application

1) Functionality: Links.py was developed to add and delete links in the network. The application must be loaded upon loading the desired network topology. Once loaded, the user can add and delete links between nodes by using the addlink or dellink function.

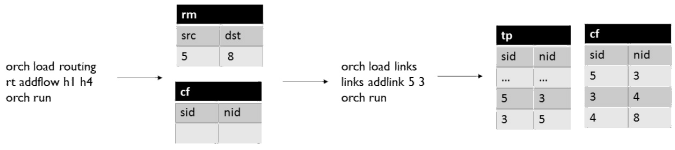

Fig. 3.

Fig. 3 shows the process of adding a flow between h1 and h4 while still not having the properly connected nodes to allow the flow to be established. Once the addlink function is run to connect h1 with s3, the topology table is update to reflect this change in the network, and the configuration table updates properly leaving a working flow between h1 and h4. Fig 4. shows how the network looks with the update link. Since the application utilizes and relies on data found in the topology table, the switch-id (sid) must be used to add or delete links rather than the name given to the node. While this does work regardless of which topology is being used, it does not follow the rules for links for a k-ary fat-tree. Further development could allow for the addition or deletion of multiple links at a time to account for the rules of links in a fat-tree. Another issue with the application is that it cannot detect whether a host is being connected another switch in the network. This affects the ishost column of the topology table. If a link between nodes contains a host, the number 1 should be inserted into the column. Since the application cannot detect if an sid belongs to a host, a 0 will always be inserted into the ishost column. While it does not affect the core functionality of the application, it is ideal to display the correct information in the ishost column.

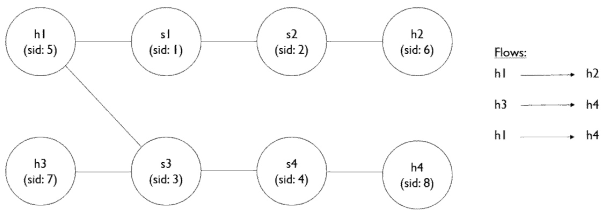

Fig. 4.

2) Performance: Fig. 5 below shows the performance of the addlink and dellink functions in milliseconds over 20 test runs. The timing was calculated by the built in time function found in the Ravel controller. The large spikes in addlink's performance were found when the system was reset and it was executed again. The first execution of the function results in a high performance time and remains low for each successive addition. The dellink function consistently posts a low performance time. It is also worth noting that performance times may be higher if the program could properly detect a host. Since it would need to refer to each node in the network

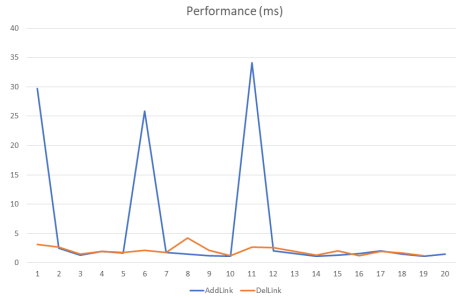

Fig. 5. Performance Graph

to find the host, causing a potential spike in time. At the moment, the times remain low due to the fact that it cannot properly check for a host. More work will need to be done to assure that performance will not degrade once the host detection problem is resolved.

C. Usability

A single file, network.py, was developed to contain all of the network topologies. The file contains a variety of if statements and each one can build a different network topology. It is fairly simple to add topologies to the file. The file contains a parameterized fat-tree topology. Further development is needed as it was planned to add an ISP topology, as more time is needed to develop it.

VI. Conclusion

A link application was developed as a hotfix to the current problem of link validation in the Ravel controller. Loading a network topology was made simpler by making a file that contains multiple topologies that can be created by the user. More topologies can easily be added to the file as the need to do so arises. More development is needed in each application, but the work shows promise that needed improvements can be made in the near future.

Acknowledgment

This research has been supported by the National Science Foundation grant awarded to Temple University Computer and Information Sciences department housed within the College of Science and Technology for Research Experience for Undergraduates (REU) during the summer of 2017. The findings and opinions are one the authors and does not reflect the views/opinions of Temple University or the National Science Foundation.

References

[1] A. Wang and J.Croft, *Automating SDN Composition: A Database Approach*, Santa Clara, California, USA: from SOSR '17

[2] A. Wang, J. Croft, and E. Dragut *Reflections on Data Integration for SDN*, Scottsdale, Arizona, USA: from SDN-NFV Security 17

[3] A. Wang, X. Mei, J. Croft, M. Caesar, and B. Godfrey *Ravel: A Database-Defined Network.*, Santa Clara, California, USA: from SOSR '16

[4] A. Wang, M. Caesar, and B. Godfrey *Ravel: Orchestrating Software-Defined Networks*, Santa Clara, California, USA: from SOSR '15

[5] Mininet: http://mininet.org/.

[6] PostgreSQL: https://www.postgresql.org/