exchange has to be used instead of a broadcast of messages to all controllers.

Neo4j has a Python library that handles the lower level network communication code and provides a RESTful web API which we invoke from our code to perform publish or subscribe functions. Note that each node in our evaluation setup has an IP address, this includes the node running Neo4j, and so the REST API can be invoked on the Neo4j database from any of the controllers and Trust Collector or Policy Ditributor components by using the IP of the Neo4j node.

The number of messages that need to be used for one complete process of our trust calculation is O(N*M) where N is the number of controllers involved in calculating the trust and M is the number of switches in the network, and there exists one instance of the Policy Distributor component and one instance of the Trust Collector component. While the Neo4j database based communication scheme described earlier helps get rid of broadcast messages, each controller (from N number of controllers) has to communicate with each of the switches (from the M number of total switches).

TABLE II. DIFFERENT NETWORK CONFIGURATIONS WE CREATED OF CONTROLLERS AND SWITCHES FOR SCALABILITY EVALUATION OF NUMBER OF MESSAGES. IN EACH CONFIGURATION, NUMBER OF SWITCHES CONTROLLED BY ONE CONTROLLER IS EQUAL TO (NO. OF SWITCHES / NO. OF CONTROLLERS).

Configuration	Controllers	Switches	Malicious controllers
Config 1	1	3	0
Config 2	3	6	1
Config 3	6	12	2
Config 4	9	27	3

Table 2 shows the various configurations of controllers and switches which we created in our evaluation setup. Note that this set of configurations created are different than those created for the earlier evaluation and were shown in Table 1. Figure 4 shows the number of messages that were used to perform one complete process of trust collections for the various configurations mentioned in Table 2. Note that here a message from a component A to component B is defined as one write of a message from a component A and its corresponding read by a component B. As can be seen from the graph, our scheme scales smoothly as the number of controllers and switches involved in the process is increased. For the highest configuration, Config 4, with 9 controllers and 24 switches, the scheme uses less than 250 messages to finish the entire process of trust calculation.

V. DISCUSSION

Our test results presented earlier show that our scheme works correctly and efficiently in weeding out malicious controllers. Since the Trust Collector decides whether a controller is malicious based on aggregate of recommendations from all other controllers, therefore our scheme provides defense against bad mouthing attacks [18, 19].

In bad mouthing attacks, a malicious party provides dishonest recommendations for another good party to malign the name of the good party. But since our scheme does not make a decision of whether a controller is malicious based on recommendation from just one other controller, therefore

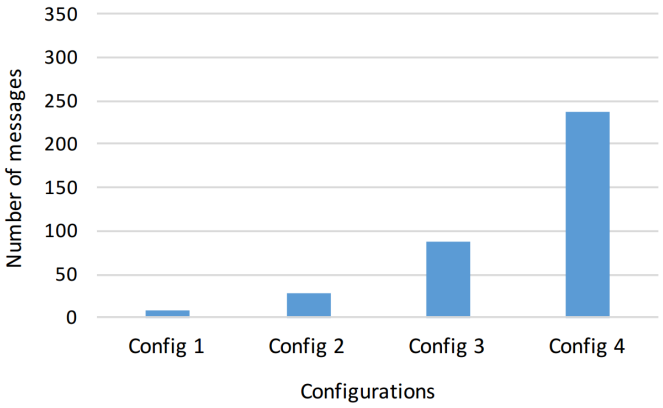

Fig. 4. Shows the number of message exchanges that take place for different network configurations of controllers and switches, the names of the configurations on the X-axis (e.g. Config1, Config2, ..) refer to the configurations in Table 1.

we can provide defense against bad mouthing as long as malicious controllers are not the majority in the total number of deployed controllers. This assumption is reasonable since we can guarantee the number of controllers which would need to become malicious before the network collapses. That is, in a network with N controllers, our scheme is guaranteed to work correctly and identify malicious controllers as long as (N/2)+1 controllers stay uncompromised. This assumption is realistic since majority of controllers is unlikely to become malicious in an instant and if they become malicious one by one over time, then our scheme will identify the malicious controllers at all times when (N/2)+1 controllers are still uncompromised.

Our scheme of collecting ratings and aggregating trust and reputation using a Trust Collector component works more robustly and accurately than delegating trust and reputation management entirely to individual controllers in a distributed environment. This is because we always need a central entity that can aggregate the ratings generated by all the controllers that are part of the distributed environment, and the central entity can then make a decision of whether a given controller is malicious by looking at what the majority of ratings say about that controller. Alternatively, the central entity can also output the result of the ratings to a human operator who can decide whether a given controller is malicious based on both their domain knowledge about the network and also based on the majority of ratings that were received for that controller.

Introducing a central trust managing entity also helps in solving an important dilemma, which is, what happens if majority of rogue controllers vote against a controller which otherwise has installed correct policies? Let us examine a case of distributed trust management, in which controllers find the policy mismatches themselves, and there is no central entity for managing the trust and reputation. There are three controllers in a network, A, B, and C, each managing one switch under them, and connected to other switches too. A network administrator defines one flow rule, i.e. block any traffic originating from IP address 200.0.0.1. The controllers install the flow rules in their respective switches. After sometime, controllers probe the switches to find out whether other controllers installed correct policies in their respective switches. A finds out that

TABLE III. DIFFERENT NETWORK CONFIGURATIONS WE CREATED OF CONTROLLERS AND SWITCHES FOR BAD MOUTHING EVALUATION. IN EACH CONFIGURATION, NUMBER OF SWITCHES CONTROLLED BY ONE CONTROLLER IS EQUAL TO (NO. OF SWITCHES / NO. OF CONTROLLERS). AND CX REFERS TO THE CONTROLLER NUMBER, E.G. C1, C2, ETC. THE BAD MOUTHING COLUMN SHOWS THE DETAILS OF WHICH CONTROLLER(S) BAD MOUTHED WHICH OTHER CONTROLLER(S). THE RESULT COLUMN SHOWS THE FINAL RESULT AFTER THE TRUST CALCULATION ROUND WHICH AGGREGATES TRUST VALUES FROM ALL CONTROLLERS IN THE NETWORK. AS SEE FROM RESULTS, THE POSITIVE OR NEGATIVE MAJORITY RATINGS AFFECT THE RESULT OF WHETHER A GIVEN CONTROLLER IS DEEMED TRUSTED OR UNTRUSTED.

Configuration	Controllers	Switches	#Malicious controllers	Bad mouthing	Result
Config 1	3	6	1	C1 bad mouthed C2	C2 found trusted
Config 2	3	6	2	C1, C2 bad mouthed C3	C3 found untrusted
Config 3	6	12	1	C1 bad mouthed C2 and C3	C2 and C3 found trusted
Config 4	6	12	2	C1, C2 bad mouthed C3 and C4	C3 and C4 found trusted

B and C have (maliciously) installed flow rules in their switches which allow traffic originating from 200.0.0.1. It rates B and C negative. B and C on the other hand rate A as negative. In presence of an automated solution of shutting down or restricting a malicious controller, this will prove to be disastrous. If, however, a human operator has to approve the shutting down or restricting of a malicious controller, then it will be a burden for him to sort through the conflicting ratings of controllers against each other.

Using the Trust Collector for aggregating the opinions about other controllers from each controller not only helps us in ascertaining the validity of recommendations with surety, but it also helps in eliminating broadcasts. If, for example, there are three controllers A, B, and C, then all of the nodes will have to send their reports to each other so that they can perform final trust calculation (since the final step in the trust calculation process inside a controller needs input from other controllers too).

However, by introducing the Trust Collector in between, all controllers send their reports to this central entity, which simply forwards it to individual controllers. While it is true that our scheme introduces this one central point of compromise, the Trust Collector, but we emphasize that it is much easier to guard and protect one component if it can help us have a safe distributed environment of controllers where each controller does not have to guarded very well. As long as majority of controllers are not compromised, our scheme guarantees that the network will keep functioning correctly.

VI. CONCLUSION AND FUTURE WORK

Securing the controllers in SDN is an open problem for research. Researches carried so far do not address the problem of identifying malicious controllers, especially in multi-controller environment. We tackle this problem by employing a trust management scheme in which controllers rate each other on the basis correctness of flow rules installed by them in switches. We do this by introducing a centralized entity which keeps record of policies to be installed, so that controllers can compare them with installed policies for rating purpose. We coded the scheme and implemented it in Ryu controller as prototype and found that the scheme is able to successfully flag a malicious controller. Our next step will be to extensively evaluate our scheme in different network topologies and number of nodes, and also by launching various kinds of attacks on our trust management system.

REFERENCES

[1] H. Kim, N. Feamster, Improving network management with software defined networking, Communications Magazine, IEEE 51 (2) (2013) 114-119.

[2] Hakiri, Akram, et al. "Software-defined networking: challenges and research opportunities for future internet." Computer Networks 75 (2014): 453-471.

[3] Sezer, Sakir, et al. "Are we ready for SDN? Implementation challenges for software-defined networks." Communications Magazine, IEEE 51.7 (2013): 36-43.

[4] Kreutz, Diego, et al. "Software-defined networking: A comprehensive survey." Proceedings of the IEEE 103.1 (2015): 14-76

[5] Kreutz, Diego, Fernando Ramos, and Paulo Verissimo. "Towards secure and dependable software-defined networks." Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking. ACM, 2013.

[6] Prasad, Abhinandan S., David Koll, and Xiaoming Fu. "On the Security of Software-Defined Networks." 2015 Fourth European Workshop on Software Defined Networks. IEEE, 2015.

[7] Liang, Zhengqiang, and Weisong Shi. "PET: A personalized trust model with reputation and risk evaluation for P2P resource sharing." Proceedings of the 38th annual Hawaii international conference on system sciences. IEEE, 2005.

[8] "Ryu SDN framework," https://osrg.github.io/ryu/

[9] "Emulab - Network Emulation Testbed," http://emulab.net/

[10] "Kaminsky DNS Attack," http://dankaminsky.com

[11] Shu, Zhaogang, et al. "Security in Software-Defined Networking: Threats and Countermeasures." Mobile Networks and Applications: 1-13.

[12] Porras, Phillip A., et al. "Securing the Software Defined Network Control Layer." NDSS. 2015.

[13] Nunes, Bruno, et al. "A survey of software-defined networking: Past, present, and future of programmable networks." Communications Surveys and Tutorials, IEEE 16.3 (2014): 1617-1634.

[14] Phemius, Kvin, Mathieu Bouet, and Jrmie Leguay. "Disco: Distributed multi-domain sdn controllers." 2014 IEEE Network Operations and Management Symposium (NOMS). IEEE, 2014.

[15] Tootoonchian, Amin, and Yashar Ganjali. "HyperFlow: A distributed control plane for OpenFlow." Proceedings of the 2010 internet network management conference on Research on enterprise networking. 2010.

[16] Koponen, Teemu, et al. "Onix: A Distributed Control Platform for Large-scale Production Networks." OSDI. Vol. 10. 2010.

[17] "Neo4j Graph Database," http://neo4j.com

[18] S. Buchegger and J. L. Boudec. "Coping with False Accusations in Misbehavior Reputation Systems for Mobile Ad-Hoc Networks."" EPFL tech. rep. IC/2003/31, EPFL-DI-ICA, 2003.

[19] Sun, Yan, Zhu Han, and KJ Ray Liu. "Defense of trust management vulnerabilities in distributed networks." IEEE Communications Magazine 46.2 (2008): 112-119.