JEDI:- A VOTE WITH NO VOTE APPROACH TO DEFEND SDNs FROM MALICIOUS ADMINISTRATORS

Sriram Manohar[1] Aditya Suresh Kumar[1] Rakesh Karanth[1] Vikas Lokesh[1]

Abstract— **Software-defined networking empowers administrators with more flexibility to program their networks. With SDN, network management moves from codifying functionality in terms of low-level device configurations to building software that facilitates network management and debugging at a centralized higher level. However this new paradigm brings with it a number of open security issues. In this paper we discuss the Malicious Administrator Problem where administrators deliberately or unintentionally misconfigure a correctly functioning controller. Inspired from an existing solution Fleet, we propose a controller design JEDI that combats this problem using a vote + validate approach. Our algorithm is more resilient than Fleet as it allows administrators to create configurations independently of each other. Our experiments show that as long as at least 1 administrator has configured the controller correctly, we can usually recover with the correct configuration within seconds.**

I. The Malicious Administrator Problem

Software Defined Networking (SDN) is revolutionizing the networking industry by enabling programmability, easier management and faster innovation. Many of these benefits are made possible by its centralized control plane architecture, which allows the network to be programmed by the application and controlled from one central entity. Routing decisions are installed in switches as flow rules, which match packet fields with an action such as forwarding or dropping. Packets that do not match any rule are sent to the controller, which can decide how to handle the packet and install a flow rule for similar subsequent packets. With the greater centralized and fine-grained control that SDN provides, comes a number of security threats.

SDN management includes set-up of switches and controllers, installation of network applications, administering controllers, managing credentials between various entities of the SDN. Albeit, the complexity of compromising SDN management is high as it needs authentication. However, If it gets compromised, the effect on the network is severe.

[1] states that 50-80% of the network outages are due to human error. It further adds that these mis-configurations are not probably due to incompetence but due to system complexity. System complexity with multiple components and many types of interactions creates an environment where the relationship between actions and outcomes is not always obvious. Thus there is a high probability that a genuine administrator , due to the complexity of the system makes the wrong decision. This problem is harder to detect as the side-effects are subtle and not immediate. This is because in these misconfigurations the worst possible configuration may not be chosen and instead a bad configuration / sub-optimal configuration would be configured. These bad decisions may add up and lead to catastrophic performance issues.

Another possibility is that a malicious administrator can deliberately misconfigure network policies which might degrade the performance of the SDN or can bring it down in the worst case. An unauthorized network application coming from a third-party, could contain malicious code. The restriction on applications to access the controller server or the underlying managed network is very less in SDN. Therefore malicious code can manipulate various configurations on the controller server, shut it down or crash it down once the malware gets installed on it. Installation of faulty or malicious switches can deter the SDN to perform intended tasks in accordance to the network policies.

Thus, the malicious administrator problem can be defined as the situation where a network administrator misconfigures a correctly functioning controller in a way that adversely affects network performance.[2] The word malicious does not necessarily assume malicious intent, since an administrator may accidentally misconfigure the network. The administrator might have accidentally or deliberately misconfigured the network.

II. RELATED WORK

In this paper we propose a new design of the SDN Controller to combat this Malicious Administrator Problem that protects the SDN from administrators who deliberately or unintentionally misconfigure routing rules. Thus the literature survey is organized as follows:-

Since we develop a new distributed SDN Controller In [A] we do a survey on papers that propose new controller designs. This helped us understand the thought process behind the evolution of the SDN Controller. In [B] We move on talk about the problem of Routing Misconfiguration as at the center of our controller design. We then describe a series of Data Plane and Control Plane Attacks associated with SDNs in [C] and [D]. In section 3 , we describe Fleet [11] - the architecture that has inspired us to build Jedi.

A. Development of the SDN Controller

One of the fundamentals of SDN is the logical centralization of the control plane. It fosters flexibility thanks to the centralized network state view and common management interface with the network devices. On the other hand, this approach introduces some drawbacks and requires a specific design in terms of control network performance, scalability

[1]Department of Computer Science and Engineering, University of California San Diego {srmanoha, asureshk, rkaranth, vlokesh }at eng.ucsd.edu

and fault tolerance. Most of the open source controllers currently available are focused on function more than on scalability and fault tolerance. This subsection provides a review on the literature about SDN distributed architectures that address the scalability and fault tolerance issues. ONIX [7] provides an environment on top of which a distributed NOS can be implemented with a logically centralized view. The distributed Network Information Base (NIB) stores the state of network in the form of a graph; the platform is responsible for managing the replication and distribution of the NIB, the applications have to detect and resolve conflicts of network state. Scalability is provided through network partitioning and aggregation (hiding the details), control over the consistency of the state is provided. In regard to the fault tolerance, the platform provides the basic functions, while the control logic implemented on top of ONIX needs to handle the failures. ONIX has been used in the B4 network, the private WAN [8] that inter-connects Googles data centers around the planet, using the OpenFlow protocol to control the forwarding devices. The large geographical dimension is addressed through a hierarchical architecture in which multiple sites are considered as logical Autonomous Systems. Within each site, a modified version of the ONIX platform is deployed. A logically centralized application with a global view provides the overall coordination across the multiple sites, providing Traffic Engineering functionality. This high level design is similar to our ICONA solution, but ICONA is not tailored to a specific use case, providing a reusable framework on top of which it is possible to build specific applications (for example Traffic Engineering). The Kandoo [9] architecture faces the scalability issue by creating hierarchical controllers architecture: the so called root controller is logically centralized and maintains the global network state; the bottom layer is composed by local controllers in charge of managing a restricted number of switches and actions that do not require the network-wide state. Applications, which require a global network view, are managed by the root controller that consequently delegates the requests (e.g., flow installation in the devices) to the local controllers. Applications with a local scope are managed directly by the local controller. The Kandoo architecture does not focus on the distribution/replication of the root controller. Fault tolerance management either in the data and control plane are not considered. A local controller cannot always manage a fault (e.g., link failure between two switches controlled by different local controllers). The consequence is that local controllers should notify the root one and wait before reacting. ElastiCon [9] and Pratyaastha [10] aim to provide an elastic and efficient distributed SDN control plane to address the load imbalances due to static mapping between switches and controllers and spatial/temporal variations in the traffic patterns. Fleet [11], PANE [12] and STN [13] design robust and distributed SDN control planes, which consider the problem of conflicting network-policy updates, each one from a different point-of-view. Fleet aims to resolve the malicious administrator problem that arises in a multi-domain administrative environment. PANE considers as scenario the participatory networks, in which the control plane provides a configuration API to its users, applications and end-hosts. Finally, STN focuses on how to solve concurrency issues that arise from the concurrent execution of control modules on different physical nodes

B. *ROUTING MISCONFIGURATIONS*

The problem of routing misconfigurations has been well studied in the context of interdomain routing protocols like BGP. For example, researchers have investigated the problem of modeling network devices to conduct reachability analysis [3,4]. The router configuration checker (rcc) uses constraint solving and static analysis to find faults in BGP configurations [5]. Their system detects faults leading to invalid routes and invisible routes. In this space, our work is perhaps most closely related to header space analysis, a static analysis approach to detecting network misconfigurations.

C. *HOST AND SWITCH BASED ATTACKS*

We do not focus on Data Plane attacks in this paper.This subsection is just to given a high level overview of the potential security threats in the data plane layer. OpenFlow mandates that packets not matching a flow rule must be sent by the switch to the controller. In spite of the control and data plane separation, this protocol requirement opens up possibilities for malicious hosts to tamper with network topology and data plane forwarding, both of which are critical to the correct functioning of the SDN. Specifically, malicious hosts can (i) forge packet data that would then be relayed by the switches as $PACKET\_IN$ messages, and subsequently processed by the controller, (ii) implement denial of service (DoS) attacks on the controller and switches, and (iii) leverage side-channel mechanisms to extract information about flow rules. Compromised soft switches can not only initiate all the host-based attacks but also trigger dynamic attacks on traffic flows passing through the switch, resulting in (i) network DoS, and (ii) traffic hijacking or re-routing. This has been enunciated in [6].

D. *CONTROL PLANE THREATS*

Control plane includes network applications policies and signalling traffic between switches and the controller for administering the managed network. In SDN, network applications policies can contradict each other. Lack of their prioritization leads to an unexpected behaviour of the managed network. For example, the set action modifies the flow rules inside the flow tables. This action can be used for honeypot, quarantine a host, or captive portals application. However, a malicious application can exploit this action to bypass the firewall policies implemented by other applications through modification of flow headers.

III. FLEET'S APPROACH

Fleet[11]is the first controller architecture that protects SDNs from malicious insiders. Fleet proposes a single configuration approach where if there are k are malicious adminstrators out of a total of n adminstrators, n-k administrators need to agree on a single configuration for it to be accepted. Figure 1 gives an overview of the Fleet architecture. The

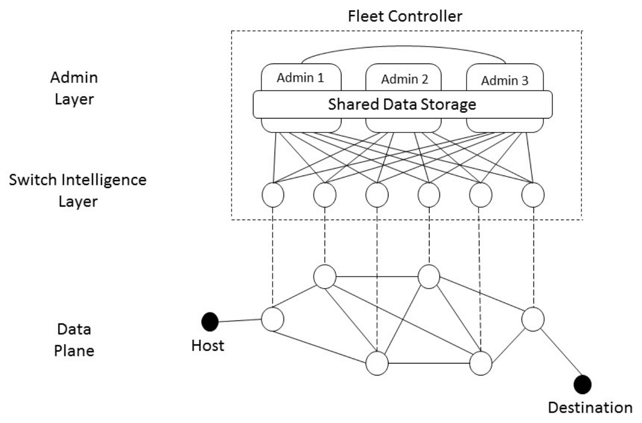

Fig. 1. Fleet Design

Fleet Controller consists of an administrator layer with shared storage and a switch intelligence layer. These layers are logically part of the controller, but physically separated. Administrators upload their configurations to machines in the administrator layer, which coordinate the selection of one or more network configurations. The selected configurations are then translated into flow rules by the switch intelligence layer and pushed to the switches flow tables. The model proposed by fleet for the single configuration approach involves each of the network administrators proposing their configuration to circumvent the detected link failure and all the other administrators are required to vote for or against the proposal within a designated interval of time termed epoch. If n-k adminstators vote for the proposal then its accepted. This approach does not allow k malicious adminstrators to push a bad configuration as long as k is less than n-k.

IV. JEDI DESIGN

The objective of Jedi is to address the malicious administrator problem, which is defined as preventing up to $n-1$ malicious administrators out of n total administrators from adversely affecting routing/forwarding, recovery from failures, or availability in the network. The parameter $n-1$ allows the network to be highly resilient. The parameter $n-1$ guarantees that as long as at least 1 administrator is non-malicious the Controller cannot be mis-configured. While the number n of total administrators does not necessarily need to be small, we assume that even for very large networks the number of network administrators will not exceed 10.

A. Adversary Model

The adversary in this problem is a group of up to $n-1$ possibly colluding malicious administrators whose goal is to reduce network availability. To accomplish this goal, administrators may accidentally or deliberately misconfigure their controller with policies that cause undesired flow rules to be pushed to switches.

B. Assumptions

We make the following assumptions about the Jetty administrator.

- Besides the controller configuration, an administrator cannot influence the controller.
- Administrators are time synchronized
- Administrators have the same view of the Network Topology.
- Administrators sharing the same routing policy if not malicious

C. Desired Properties of Jedi

Rapid Recovery and Protocol Independence Goals of Jedi and Fleet are the same. In this context Indistinguishably changes and we add a new property called Guarantee.

- Indistinguishability: Outside of link failures and subsequent recovery, the network configuration should be the same as long as there is atleast 1 non-malicious administrator.
- Rapid recovery: In the event that a link fails, the controller should find and install a new configuration in the order of seconds.
- Protocol independence: Our approach should not rely on details of the underlying routing protocols.
- Guarantee: Irrespective of the vote, the best possible network configuration computed by the administrators must be applied to the network. The control plane must guarantee that the configuration applied is the best configuration it received wrt some metric.

D. Jedi Design

Figure 2 gives an overview of the Jedi architecture.

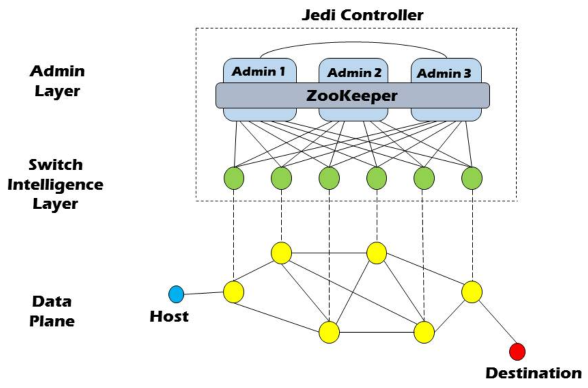

Fig. 2. Jedi Design

The crux of the idea behind the design is to create n routing configurations, one for each administrator, and allow the switch intelligence to probabilistically select and validate one of the configurations for each flow based on a set of metrics. As vote, each of the administrators will send a tuple consisting of a path and a confidence score. The confidence score for a path is based on a metric. The metrics that we used for our evaluation are

- Number of hops
- Bottleneck bandwidth
- Latency.

The switch intelligence layer accepts the path and the confidence score. A validation is then performed on the vote wherein the confidence score is verified on the path.

We consider 3 types of administrators.

- Type 0 The Deliberately Malicious Administrator - This class of administrators deliberately misconfigure the controller with a bad path. Along with the bad path, they send a confidence score better than that of the confidence score of the best possible path.
- Type 1 The Unintentionally Malicious Administrator - This class of administrators unintentionally misconfigure the controller with a bad path. However they report the confidence score of the corresponding path.
- Type 2 The Good Administrator - This class of administrators report the best possible alternate path in case of a link failure. They report the corresponding confidence score along with the path.

The model is a probabilistic model wherein the switch intelligence layer will choose a random administrator and ask for a vote. The administrator chosen will send a (path, confidence) as vote where the confidence of the path is computed in accordance with some metric. The metric could be latency, bandwidth, number of hops, packet loss or any combination of these. Coming up with a reasonable combination metric with appropriate weightage given for each metric in itself is a research question.

The switch intelligence layer accepts the vote and validates the confidence score by evaluating the metric on the path proposed. Three scenarios can occur during evaluation.

- Case 1 - The Best Case - The administrator was a Type 2 administrator. He sends the correct path that had the best possible confidence score. The switch intelligence layer validates this score and promulgates this score to all other administrators. Only administrators who claim to have a better score than the score advertised by the Switch Intelligence Layer will be allowed to vote in the next epoch.
- Case 2 - A worse case - The administrator here is a Type 1 administrator. He has misconfigured the network and has hence ended up computing a suboptimal path. The confidence score he reports is the same as that of the path he computes. This leads to two possibilities. If this administrator votes after the TYPE 2 admin or after a TYPE 1/TYPE 0 admin who has a path with a better confidence score than his path, the switch intelligence layer will not accept his vote. This property of the switch intelligence layer leads to faster convergence of the controller after link failure.
- Case 3 - The worst case - The administrator here is a Type 0 administrator. He has deliberately misconfigured the network and has hence ended up selecting the worst possible path. However, when he reports a confidence score he reports a score better than the best possible confidence score. 2 scenarios can happen here.
  - What would be the outcome if the first person to cast his vote were a type 0 administrator? In this case, the switch intelligence layer accepts the vote and does a validation on the confidence score. Once it determines that the score reported is incorrect - it stores the path and the correct score as the current configuration. It promulgates the correct confidence score of the bad path to all other administrators and accepts votes only if they claim to have a better confidence score than what was advertised by the malicious administrators.
  - The malicious administrator votes after a Type 2 (Good Guy) administrator. The switch intelligence layer will again in this case need to verify if the score is valid because the score claimed by the malicious administrator is better than the current best possible score. However once the switch intelligence layer realizes that the score is incorrect it will discard the vote and retain the current best path and the best confidence score

During the voting epoch the administrator's configuration is verified by the switch intelligence layer to determine if the proposed configuration actually matches up to the confidence score. This is how we figure out if the administrator is up to something fishy. We don't rely on the intuition of the other administrators to help figure out faulty configurations. Also a configuration is evaluated only if the administrator predicted configuration score is higher than the currently evaluated configuration score which belongs to some administrator. Thus if the network has a decent number of non-malicious administrators the chances are that we won't have to do a lot of validations. This leads to a faster conclusion of the mitigation process. Thus our Voting Approach is no Voting. It is a factual verification of the administrator supplied configurations to conclude the process faster.

E. Comparison of Jedi and Fleet

- Jedi is more resilient and allows administrators to create configurations independently of each other. That is, the administrators do not need to agree on a fixed configuration.
- Jedi tolerates up to n-1 malicious administrators whereas Fleet only tolerates up to as k administrators where k is less than n-k.
- Jedi requires more changes to the existing SDN architecture than Fleet.
- The Switch Intelligence layer of Jedi needs to maintain more information because it stores each administrators configuration.

V. IMPLEMENTATION

The network topology for the experiment was implemented using Mininet, OpenFlow and POX which is a Python based SDN Controller. The topology consisted of two end-hosts A and B interconnected by switches. We used Mininet OVS-OFCTL switches operating in Kernel mode and the controller used for the flow rules distribution was a remote controller operating in out-of-band mode. ZooKeeper[20] was used to achieve distributed consensus, facilitate administrator voting and to implement shared data storage for the administrators and the switch intelligence layer to store the necessary configurations and the confidence score. Python scripts were written to

- Simulate random link failures in the network
- Simulate the functionality of the three classes of administrators
- Submission of the desired configuration and the confidence score by each administrator to the switch intelligence layer
- The Voting process
- The functionality of the switch intelligence layer which includes - notifying administrators to participate in voting, validating admin config, confidence score and storing the validated configuration with the correct confidence score in ZooKeeper

The open source Kazoo library was used to help these Python scripts interact with ZooKeeper whenever they needed to make use of the services of ZooKeeper[21][22]. All the experiments that we conducted had in common the following three phases and code was written to handle these three cases

- Random link failure
- The Voting process
- Mitigation of link failure

A. *RANDOM LINK FAILURE*

The experiments begin with the simulation of a random link failure between the two hosts A and B which hinders end to end communication. The controller detects this link failure and informs the Switch Intelligence Layer.

B. *THE VOTING PROCESS*

Upon being informed about a link failure the switch intelligence layer is designed to initiate our Voting Protocol which involves intimating all the adminitrators about the problem. The administrators then

- Prepare an alternate configuration and the associated confidence score to mitigate this link failure.
- Wait for their turn to submit these to the switch intelligence layer for validation

We used ZooKeeper to help us with the voting process as it provides an out of the box solution for distributed consensus. The reasons for using ZooKeeper are detailed below

C. *WHY ZOOKEEPER?*

The author's of the original Fleet[11] paper were reluctant in sharing their code which forced us to use ZooKeeper as we neither had the time nor the sufficient knowledge to implement even a basic version of the consensus model among the administrators that the original paper talks about. Using ZooKeeper also helped us eliminate certain components that we think were unnecessary in the original model evaluated by Matsumoto et al. Some of these components are

- The need for Shared Data Storage - We believe one could use ZooKeeper's node and ephemeral node[21] facilities to store the configuration and the associated confidence score rather than have a dedicated data storage layer. Our model uses ephemeral nodes as these are active only as long as the process that created it remains active. This provides us with the advantage that we don't have to worry about "cleaning up" (Involves deletion of stale configuration and confidence score) the system every time it goes down or comes back up.
- Allowing a single administrator to submit a configuration and confidence score during a voting epoch - Using ZooKeeper one can implement this functionality easily as all it involves is acquire lock on a shared object. The shared object in our model was an ephemeral node and we used the library functions available in the Kazoo library[22] to acquire lock on this ephemeral node. The administrator who acquires the lock first is allowed to vote while the others wait for their turn. ZooKeeper is strictly ordered which implies that the administrator wait times are determined based on the time they made a move on the lock object and is not random each time the lock becomes available.
- Need for polynomial interpolation and shared secret - The authors of the original paper use polynomial interpolation on Schnorr signatures[23] to create a distributed threshold signature scheme which makes use of a verifiable secret sharing scheme to ensure that the administrator creating the private key cannot cheat. That is, they use it primarily to prevent multiple votes from being cast by a single administrator and to prevent spoofing. These techniques are heavily dependent on the number of bits used in the RSA private keys. We believe one can use ZooKeeper to efficiently solve the previous mentioned problems. ZooKeeper allows communication over SSL/TLS. We prevent the problem of multiple votes and spoofing by authenticating each administrator through a CA signed certificate and also keep a track of the administrators who have already voted to prevent them from casting their votes again.

D. *MITIGATION OF LINK FAILURE*

The switch intelligence layer verifies each administrator supplied configuration and at the end of the voting cycle has short-listed one configuration that has the best score. The flow rules mentioned in this configuration file are sent to the controller which takes care of populating these in the desired switches.

VI. Evaluation

In each of the following experiments carried out, a random network graph is generated on the fly with hosts: source and destination, and up to 25 switches. These Network Graphs are of two different types: Sparse Graph and Dense Internet-like graphs as inspired from [3]. The probability of existence of link between two switches in Sparse Graph is 20% whereas it is 80% in Dense Graphs. Once the graph is established, the best source-destination path is chosen, appropriate flow rules are added to the concerned switches and traffic is pushed onto the network. A random link on the current path is deleted dynamically and the switch intelligence layer starts the voting process to chose the best possible alternate source-destination path. Once the voting cycle is completed, the best validated alternate path is chosen and the appropriate flow rules are added to the concerned switches and network traffic resumes. Choice of an appropriate metric to evaluate paths in the network is in itself a research question and largely depends on the type of applications supported by the network. For example, a file transfer application might give maximum weight age to Bandwidth whereas a Video Chat application might give maximum weight age to minimum latency. Here, we experiment on three types of metrics which were chosen in succession:

1. Maximum Bottleneck Bandwidth
2. Minimum Latency
3. Minimum number of hops

Figure 3 shows a plot of total time with respect to the number of malicious administrators when the chosen metric is maximum bottleneck bandwidth. The total number of administrators is fixed to 10, and the number of malicious administrators sequentially increases from 1 to 9. Here, the total time = link failure detection time + admin path recovery time + vote time + new path establishment time. We observe that for both Dense and Sparse Graphs, the total time increases as the number of malicious administrators increase because the malicious administrators tend to take excessive time to come up with an appropriate confidence score to their proposed incorrect path as a best effort to spoof the underlying switch intelligence layer. Figure 4 and 5 show a similar graph when the chosen metric is minimum latency and minimum number of hops respectively.

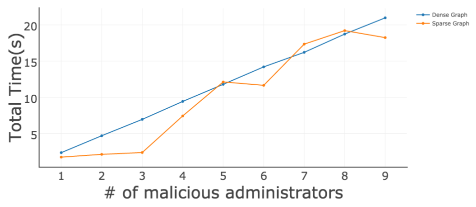

Fig. 3. Total Time v/s No. of Malicious Admins when the metric is maximum bottleneck bandwidth

Figure 6 shows a plot of consensus time with respect to the number of malicious administrators. Here, the consensus time = time taken by Zookeper to poll the

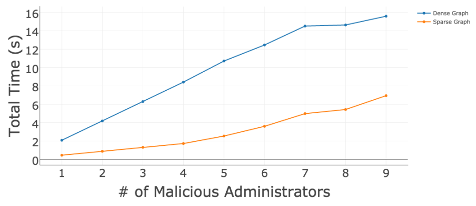

Fig. 4. Total Time v/s No. of Malicious Admins when the metric is minimum latency

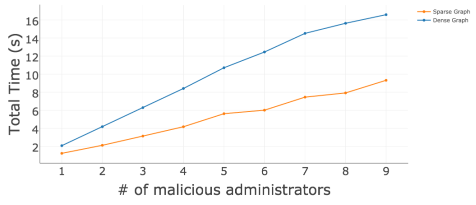

Fig. 5. Total Time v/s No. of Malicious Admins when the metric is minimum number of hops

admins + time taken to validate each path. As observed in the graph, the consensus time remains more or less constant with respect the number of malicious administrators because each proposed path approximately takes the same amount of time to get validated and those paths which do not get validated are not accepted by the Switch Intelligence Layer. Similar results are observed when the metrics are maximum bottleneck bandwidth and minimum number of hops respectively.

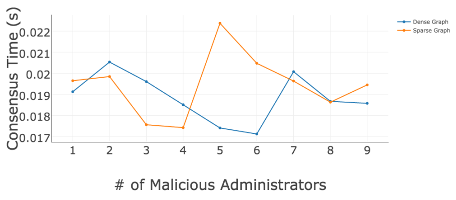

Fig. 6. Consensus Time v/s No. of Malicious Admins

Figure 7 shows a plot of consensus time with respect to the total number of administrators. This is an Anomalous graph depicting the probabilistic nature of Jedi, where the number of malicious administrators(k) is chosen randomly among the total number of administrators(n).

VII. LIMITATONS AND FUTURE WORK

We have evaluated and applied metrics on the static Network to determine a new configuration in the event of failure. We have evaluated the design based on individual metrics like shortest paths, bottleneck bandwidth and latency.

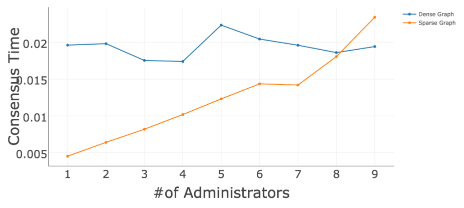

Fig. 7. Consensus Time v/s Total no. of Malicious Admins

Deciding on the correct metric in itself is a research question. In future we would like to decide the best metric based on applying and evaluating configurations on the real network. For eg, if bandwidth intensive applications dominate the network traffic, bandwidth would be chosen as the correct metric and if latency dependent applications dominate the network traffic we would want latency to be the dominant metric. However a problem with this approach is that traffic keeps changing which makes it all the more complex to decide alternate routing paths in the event of network failures.

In future we plan to develop a method and series of metrics for evaluating routing paths in our approach. In particular, we plan to investigate which metrics are most easily collected in SDNs and which most accurately describe the network performance based on the needs of the application. With such a method, we can implement and test the performance of this approach. Since Jedi's approach only requires one non-malicious administrator, it would be interesting to explore the trade offs between maintaining availability and tolerating a greater number of malicious administrators. This is because as the number of malicious administrators increase, the total convergence time increases.

A malicious administrator who successfully attacks routing may be able to ex-filtrate data from a host in the network to a remote location. While this paper focuses on preventing the manipulation of network configurations, we believe that preventing the ex-filtration of information, be it information about the network, personal data, passwords, or business policies, is an important research challenge that must be addressed in current networks, particularly those operating in commercial or corporate environments.

VIII. CONCLUSIONS

We have proposed a new controller design for SDNs to combat Malicious Insider Threats. We used Zookeeper to implement our distributed Voting protocol and to provide Shared Data Storage where Administrators can upload their configurations and associated meta data. Our design allows up to n-1 administrators to be malicious. In the event of a link failure, as long as there is at least one good administrator the best configuration would be applied on the Network. This approach is more resilient than Fleet[11] the only available state-of-the art solution to this problem.

ACKNOWLEDGMENTS

We wish to thank Prof. George Porter and Rahul Bhalerao for their advice, guidance and encouragement in the UCSD graduate networking class, CSE 222A. We also thank Steve Matsumoto for clarifying a lot of our doubts wrt Fleet nitty-gritties.

REFERENCES

[1] Juniper Networks. Whats behind network downtime?, 2008.
[2] W.-K. Chen, Linear Networks and Systems (Book style). Belmont, CA: Wadsworth, 1993, pp. 123135.
[3] E. Al-shaer, W. Marrero, A. El-atawy, and K. Elbadawi. Network Configuration in A Box: Towards End-to-End Verification of Network Reachability and Security. In Proceedings of the IEEE International Conference on Network Protocols, 2009
[4] G. Xie, J. Zhan, D. Maltz, H. Zhang, A. Greenberg, G. Hjalmtysson, and J. Rexford. On Static Reachability Analysis of IP Networks. In Proceeding of IEEE INFOCOM, 2005.
[5] N. Feamster and H. Balakrishnan. Detecting BGP Configuration Faults with Static Analysis. In Proceedings of theUsenix Symposium on Network Systems Design and Implementation, 2005.
[6] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann, Sphinx: Detecting security attacks in software-defined networks, in NDSS, 2015.
[7] Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, and Scott Shenker. Onix: A distributed control platform for large-scale production networks. In Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation, 2010.
[8] S. Jain, et al, B4: Experience with a Globally-Deployed Software Defined WAN in SIGCOMM, 2013
[9] AA Dixit et al. ElastiCon: an elastic distributed sdn controller. Proceedings of the tenth ACM/IEEE symposium on Architectures for networking and communications systems, 2014.
[10] A Krishnamurthy et al. Pratyaastha: an efficient elastic distributed sdn control plane. Proceedings of the third workshop on Hot topics in in software defined networking 2014
[11] S Matsumoto et al. Fleet: defending SDNs from malicious administrators. Proceedings of the third workshop on Hot topics in software defined networking, 2014
[12] AD Ferguson et al. Participatory networking: An API for application control of SDNs. ACM SIGCOMM Computer Communication Review, 2013
[13] M Canini et al. A Distributed and Robust SDN Control Plane for Transactional Network Updates. The 34th Annual IEEE International Conference on Computer Communications (INFOCOM 2015)
[14] W. D. Doyle, Magnetization reversal in films with biaxial anisotropy, in 1987 Proc. INTERMAG Conf., pp. 2.2-12.2-6.
[15] G. W. Juette and L. E. Zeffanella, Radio noise currents n short sections on bundle conductors (Presented Conference Paper style), presented at the IEEE Summer power Meeting, Dallas, TX, June 2227, 1990, Paper 90 SM 690-0 PWRS.
[16] J. G. Kreifeldt, An analysis of surface-detected EMG as an amplitude-modulated noise, presented at the 1989 Int. Conf. Medicine and Biological Engineering, Chicago, IL.
[17] J. Williams, Narrow-band analyzer (Thesis or Dissertation style), Ph.D. dissertation, Dept. Elect. Eng., Harvard Univ., Cambridge, MA, 1993.
[18] N. Kawasaki, Parametric study of thermal and chemical nonequilibrium nozzle flow, M.S. thesis, Dept. Electron. Eng., Osaka Univ., Osaka, Japan, 1993.
[19] J. P. Wilkinson, Nonlinear resonant circuit devices (Patent style), U.S. Patent 3 624 12, July 16, 1990.
[20] https://zookeeper.apache.org/
[21] https://zookeeper.apache.org/doc/trunk/zookeeperOver.html
[22] https://kazoo.readthedocs.org/en/latest/
[23] Schnorr signature - http://web.stanford.edu/class/cs259c/lectures/schnorr.pdf