Fleet: Defending SDNs from Malicious Administrators

Stephanos Matsumoto
CMU/ETH Zurich
smatsumoto@cmu.edu

Samuel Hitz
ETH Zurich
hitzs@student.ethz.ch

Adrian Perrig
ETH Zurich
aperrig@inf.ethz.ch

ABSTRACT

We present the malicious administrator problem, in which one or more network administrators attempt to damage routing, forwarding, or network availability by misconfiguring controllers. While this threat vector has been acknowledged in previous work, most solutions have focused on enforcing specific policies for forwarding rules. We present a definition of this problem and a controller design called Fleet that makes a first step towards addressing this problem. We present two protocols that can be used with the Fleet controller, and argue that its lower layer deployed on top of switches eliminates many problems of using multiple controllers in SDNs. We then present a prototype simulation and show that as long as a majority of non-malicious administrators exists, we can usually recover from link failures within several seconds (a time dominated by failure detection speed and inter-administrator latency).

1. INTRODUCTION

Software-defined networks (SDNs) separate the control plane and the data plane, with routing decisions made at a centralized controller (which may be physically distributed across multiple machines), and switches simply forwarding packets according to these decisions. 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, however, comes a greater risk of outages resulting from network administrator error. Human error is reportedly responsible for 50 to 80 percent of network outages [6], and an administrator who misconfigures a controller can easily degrade network performance, even if the controller is functioning correctly and no problematic flow rules are installed in switches. We call this the *malicious administrator problem*, where a network administrator misconfigures a *correctly functioning* controller in a way that adversely affects network performance[1].

[1]Our use of the word "malicious" does not necessarily assume malicious intent, since an administrator may accidentally misconfigure the network.

HotSDN'14, August 22, 2014, Chicago, IL, USA.

ACM 978-1-4503-2989-7/14/08 . . . $15.00.
http://dx.doi.org/10.1145/2620728.2620750.

Detecting suboptimal network performance is challenging. Even if one assumes Byzantine controller failures, for example, simply determining what constitutes a "fault" if a controller is functioning properly but misconfigured is difficult to pinpoint. Moreover, enforcing policies that determine what flow rules should and should not be installed in switches is not sufficient, since a flow rule may harm network performance while still conforming to a policy. While previous proposals have followed these approaches (§2), predicting the outcome of a new flow rule is still a challenge.

In this paper, we design Fleet, a controller architecture that makes a first step towards addressing the malicious administrator problem, and present two approaches that can be used with Fleet (§4). The first approach leverages a threshold voting protocol to ensure that the network has a single configuration, while the second approach allows selection among multiple, independently-created configurations from each of the administrators. While the two approaches have fundamental differences, they both leverage logically centralized, physically distributed controllers and make use of digitally signed information. The use of multiple physical controllers creates the need for switches to dynamically associate with these controllers, and digital signatures require switches to be able to verify these signatures. Fleet has a controller layer collocated with switches to provide both of these features at the switches themselves.

In summary, our paper makes the following contributions:

- A definition, adversary model, and list of assumptions for the malicious administrator problem (§3).
- Fleet, which enables two possible approaches to the malicious administrator problem: an easily-deployable single-configuration approach, and a multi-configuration approach more resilient to malicious administrators (§4).
- An argument for Fleet's intermediate "switch intelligence" layer between controllers and switches and how this layer facilitates higher-level switch functionality (§4.1).
- A simulation and evaluation of the single-configuration approach, as well as a discussion on how we could implement our multi-configuration approach (§5).
- A discussion of future work and related problems motivated by Fleet, such as how to maintain data secrecy with malicious administrators (§6).

2. BACKGROUND AND RELATED WORK

Currently known threat vectors to SDNs include attacking the reliability of network forensic data, the authentication between controllers and switches, and vulnerabilities in switch firmware [5], as well as the availability of SDN controllers [10]. One potential mitigation for these attacks is to take a distributed systems approach to prevent compromised controllers from negatively affect-

ing network availability, such as replication and diversity of hardware/software [5]. While replication is necessary to mitigate misbehavior, it is unrealistic that a company would deploy a diversity of hardware and software in their network, since it increases the complexity of maintaining and updating network equipment.

The idea of distributing controllers over several physical machines has also been studied in other work. HyperFlow uses a logically centralized but physically distributed controller in which switches connect to the physically closest part of the controller, which updates the other physical machines on network events via a publish/subscribe system [12]. However, if a HyperFlow controller fails, its switches must be reconfigured to connect to a new controller. ElastiCon addresses this problem by proposing a dynamic migration protocol between controllers and implements a dynamic load balancing system based on this protocol [2].

Flow rules that might adversely affect network availability can be blocked by using security policies to detect and prevent the installation of such rules in switches. FortNOX accomplishes this by enforcing role-based source authentication and security policies on candidate flow rules [7], while VeriFlow does so by dividing the network into equivalence classes based on the domains of the rules to efficiently check adherence to invariant properties in the network [4]. These approaches are orthogonal to the work in this paper and can complement our approach in a real deployment.

2.1 Threshold Signatures

The use of threshold cryptography among controllers has been previously suggested as a possible solution to transmit correct information from controllers to switches [5]. However, to our knowledge no further work has incorporated this technique. Threshold cryptography can be applied to a variety of signature schemes, but in this paper we apply threshold techniques to Schnorr signatures, which are simple and efficient in addition to being secure [8].

Threshold signatures incorporate a secret sharing scheme, most commonly Shamir's secret sharing scheme, which splits a secret among n parties such that at least k of them are needed to reconstruct the secret [9]. The scheme generates n points on a polynomial of degree $k-1$ so that any k points uniquely determines the polynomial using interpolation and a given point on this polynomial (such as its value at $x = 0$) contains the secret. Previous work has applied this technique to Schnorr signatures to create a distributed threshold signature scheme, which can also use a verifiable secret sharing scheme to ensure that the dealer creating the private key shares cannot cheat [11].

3. PROBLEM DEFINITION

Our main objective in this paper is to address the *malicious administrator problem,* which we define as preventing k malicious administrators out of n total administrators from adversely affecting routing/forwarding, recovery from failures, or availability in the network. The parameter k allows the network to be set to a specific level of resilience to malicious administrators. While the number n of total administrators does not necessarily need to be small, we assume that even for large networks the number of network administrators will not exceed 10.

3.1 Adversary Model

The adversary in this problem is a group of k 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. However, besides the controller's configuration, an administrator cannot influence the controller. Thus, for example, an administrator cannot send controller messages that respond to a switch message unless the switch has indeed sent such a message, nor can a controller send arbitrary messages to the switch.

3.2 Assumptions

We assume that switches are preconfigured with the necessary cryptographic keys to authenticate controller messages. Thus a malicious administrator cannot, for example, preinstall other keys during setup. We also assume that each administrator has the same view of the network topology during the protocol. The OpenFlow specification makes similar assumptions, requiring that switches be configurable with certificates authenticating them to the controller and vice versa, and notes that configuration of the flow entries at switch startup is out of scope [3].

With regards to controllers, we assume that the administrators' machines are loosely time-synchronized to within several seconds. We also assume that each administrator has a connection to each switch, and that there is a communication channel that is always available and allows administrators to exchange messages. We can achieve such a channel by for example reserving certain links in the network specifically for inter-administrator communication or by using an entirely separate network to enable this communication.

Finally, we assume that all non-malicious administrators share a common routing policy in the sense that any non-malicious administrator will agree with a configuration proposed by any other non-malicious administrator. We make this assumption on the basis that in a real-world deployment, there will be network-wide guidelines for routing and forwarding depending on the purpose and requirements of the network.

3.3 Desired Properties and Metrics

Our protocol aims to achieve the following properties:

- **Indistinguishability:** Outside of link failures and subsequent recovery, the network configuration should be the same as long as the number of malicious administrators does not exceed k.
- **Rapid recovery:** In the event that a link fails, the controller should find and install a new configuration on the order of seconds.
- **Protocol independence:** Our approach should not rely on details of the underlying routing or cryptographic protocols.

We evaluate our approach using the following metrics:

- **Probability of compromise:** the probability that a group of k malicious administrators can cause the use of a different network configuration than that of the benign administrators.
- **Protocol overhead:** the time and computational overhead of the protocol, measured from the point at which a link failure is detected and when a fix has been proposed.
- **Recovery time:** the time between the moment a link fails and when a fix has been issued.

4. FLEET CONTROLLER DESIGN

We begin with an overview of the Fleet controller architecture, which consists of an administrator layer with shared storage and a switch intelligence layer as shown in Figure 1. These layers are logically part of the controller, but physically separated. In particular, the switch intelligence layer is physically collocated with switches, with each instance of switch intelligence operating on top of a switch. Administrators upload their configurations to machines in the administrator layer, which coordinate the selection of one or more network configurations. The selected configurations

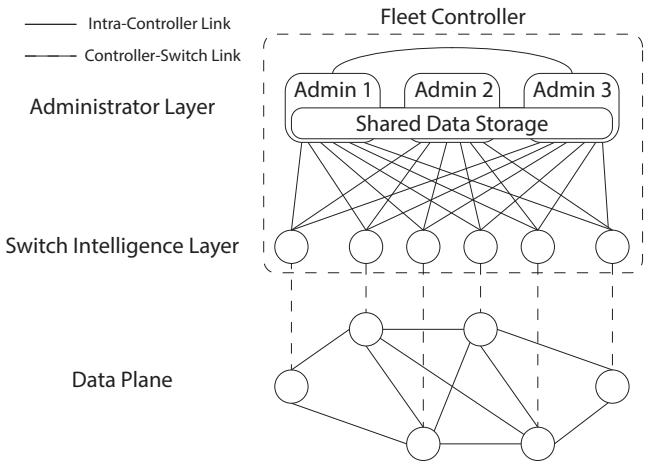

Figure 1: Diagram of a Fleet controller with 3 administrators in a small sample network.

are then verified and translated into flow rules by the switch intelligence layer and pushed to the switches' flow tables.

We then describe two approaches to addressing the malicious administrator problem with the Fleet controller. The first is a straightforward single-configuration approach, requiring a threshold of administrators to agree to a high-level routing configuration that is then pushed to the switch intelligence. We achieve this through a voting protocol in which administrators take turns proposing and voting on high-level routing decisions following network events such as a link or switch failure or flow expiration. Once a proposal is accepted by a threshold number of administrators, it is pushed to the switch intelligence, which handles flows according to these rules.

Our second approach requires more significant changes to be deployable and is not guaranteed to always use non-malicious routes, but can still use correct routes with even a single non-malicious administrator. The crux of the idea is to create n routing configurations, one for each administrator, and allow the switch intelligence to probabilistically select one of the configurations for each flow based on a set of metrics. Malicious administrators can then only disrupt network operation in their own configurations.

4.1 Controller Overview

Fleet utilizes a single, logically centralized controller physically distributed across separate machines, as shown in Figure 1. There are two layers in our controller: an *administrator layer*, which contains a physical machine for each administrator along with a shared data storage system, and a *switch intelligence* layer, which operates on top of switches and mediates communication between switches and the administrator layer. This includes verifying signed messages, translating routing configuration to flow rules, and connecting to a different administrator machine in case one is unresponsive.

The administrator layer consists of a set of physically separated machines to which administrators upload their network configurations and a shared data storage system. We assume that the administrators utilize out-of-band channels to communicate with one another, and use a distributed data storage system such as Oracle Coherence[2] or Hazelcast[3] to ensure that they all maintain a consistent view of the network. While Fleet does not require a one-to-one correspondence between administrators and machines to which they upload configurations, we argue that such a separation prevents the failure of a single machine from disconnecting multiple administrators.

Administrators are responsible for equipping their controller with a routing configuration that determines a set of metrics by which routes should be chosen. A simple configuration, for example, may always select a path with the fewest hops. These configurations may be straightforward and based only on network metrics, or may also incorporate fine-grained network routing policies. For example, in an ISP's network, these guidelines may contain information about business relationships and how flows to or from those entities should be routed. Because these policies may vary widely depending on the network requirements and on the metrics that can be gathered through OpenFlow, we consider the specific encoding of configurations outside the scope of this paper.

The switch intelligence layer consists of a set of switch intelligence instances, one for each switch in the network (though there may be fewer in very large networks). An instance of switch intelligence serves as a basic controller which handles tasks that switches cannot easily perform on their own, such as verifying administrator signatures and translating routing configuration into flow rules. If a switch receives a packet that does not match any of its rules, it passes the packet to its switch intelligence, which pushes an appropriate flow rule according to its routing configuration.

Since switch intelligence runs on top of switches, it does not access administrators' shared data but instead receives necessary information as messages from the administrators. Therefore, the switch intelligence is also responsible for facilitating communication between switches and administrators, connecting to a subset of administrator controllers and dynamically changing this set for load-balancing purposes if necessary. Because the switch intelligence is part of the controller and interacts with the administrators on behalf of the switch, we do not have to change connected administrators using controller-switch messages. Instead, this handover can be coordinated entirely within Fleet while the switch remains connected to a single switch intelligence instance.

The use of a switch intelligence layer presents several challenges in practice. As current switch hardware does not support the functionality enabled by switch intelligence, migrating to the use of this layer would either require an additional device to be installed at each switch, or a new switch entirely. While the switch intelligence layer ensures that a malicious administrator cannot easily compromise many switches from a single physical location, the increased complexity in switches contrasts with the prevailing philosophy of keeping SDN switches simple. For better interoperability with current SDN architectures, we point out that the switch intelligence layer can also be implemented as a single machine that interacts with all switches in the network.

4.2 Single-Configuration Approach

In this approach, we set a threshold t of administrators that must agree on a configuration in order to install it in the network. We define $t = k+1$ so that even if k administrators are malicious, the single configuration cannot be altered without the agreement of at least one benign administrator. Since $t \leq n-k$ (otherwise at least one malicious administrator must agree to reach the threshold), we can see that $n \geq 2k+1$, ensuring that there is always a majority of non-malicious administrators. We then use Shamir's secret sharing to distribute shares of a private key among them, and install the corresponding public key in the switch intelligence instances. The distribution of the private key can be done without a trusted dealer [1], or by using a verifiable secret sharing protocol [11].

When a link failure is detected, an administrator controller can initiate a vote to determine a workaround path for the link. A con-

[2]http://www.oracle.com/technetwork/middleware/coherence/
[3]http://www.hazelcast.com

troller becomes the *initiator* by broadcasting a unique vote identifier, which is then used to seed a generator returning a pseudorandom permutation of the numbers 1 through n, which is called the *proposal sequence*. This permutation can be independently computed by each administrator in order to ensure that the sequence is correct.

The proposal sequence is used to determine the order in which administrators propose fixes for the broken link. The voting time is divided into time periods of a predetermined length called *voting epochs*. In each epoch, an administrator proposes a fix according to its routing policy, and the other administrators have until the end of the epoch to accept or reject the fix based on their own routing policies.

Administrators can then cast a vote for a proposal by signing the proposal according to a threshold signature scheme, producing a signature share that can be independently verified by other administrators. For efficiency, we use a threshold Schnorr signature [11]. If a proposal receives fewer than t votes before the end of the epoch, then it fails and the next epoch begins with a new proposal. If the vote succeeds, then a valid signed routing configuration can be reconstructed from the signature shares and sent to the switch intelligence layer to be implemented in the switches.

We use the above voting protocol instead of the Paxos protocol because controllers still must generate a valid signed configuration. Without a threshold signature scheme, a smaller group of malicious administrators may be able to push a valid malicious configuration to the switch intelligence. The threshold signature scheme ensures that some minimum threshold of agreement is always required to generate a valid configuration, and in particular prevents k colluding malicious administrators from agreeing on a network-wide configuration.

The distribution of votes in this approach implies that each administrator has equal authority in the entire network. Because this may not be true in practice, we also suggest that in such a case the administrators create more than n signature shares and distribute the shares to provide some administrators with more votes than others. In this case the threshold t should be strictly greater than the total number of votes held by any set of k malicious administrators, in order to prevent such a set of creating a valid configuration.

4.3 Multi-Configuration Approach

Our multi-configuration approach is more resilient and allows administrators to create network configurations more independently of each other, but requires more changes to the existing SDN architecture. Therefore, we propose this approach as an avenue of further exploration, describing our preliminary work in this space and noting the challenges of taking this approach.

In the multi-configuration approach, each administrator has its own routing configuration and can construct it independently of the other administrators. While administrators are expected to follow any network policies when creating a configuration, they are free to choose the remaining routes according to any available information. They then send their configurations to the switch intelligence layer, which stores these configurations. Since the number of administrators for a network is assumed to be small (i.e., less than 10), storing and selecting among these configurations should not create a large storage or computational burden on the switch intelligence layer.

These configurations are then deployed alongside each other to create a series of n routing planes, each of which can be used by traffic in the network. Because we consider several routing planes simultaneously in operation, we can relax the requirement that all non-malicious administrators agree on each others' configurations. Additionally, we do not need any sort of consensus among administrators, eliminating the need for a voting protocol and threshold signatures. Because any of the routing planes can be used, we also are able to tolerate up to $n-1$ malicious administrators, since we only need a single good plane in order to ensure network availability.

However, this approach produces other challenges in practice. In particular, who should choose which routing plane to use and how? While we could endhost applications the ability to select routing planes based on their own requirements, this would require significant changes to the SDN architecture. Therefore, we propose that the switch intelligence layer probabilistically choose a routing plane for each flow and adjust these choices if necessary. However, this requires the switch intelligence to have its own metrics for evaluating the performance of a flow. Allowing switch intelligence instances to evaluate metrics in this way would also require changes to current SDNs.

This approach also has several drawbacks compared to the single-configuration approach. For example, the switch intelligence needs to maintain more information because it stores each administrator's configuration. Additionally, since administrators can simply ignore the planes, it can be difficult to detect whether or not an administrator's plane is violating network routing guidelines. Finally, because any of the planes can be used and it is not apparent which path is best by only analyzing the configuration, this approach is reactive rather than preventative. That is, the use of a bad path can only be detected after it is being used, allowing even a single malicious administrator to temporarily reduce network performance (however small that probability may be).

Since a single malicious administrator can cause an undesired path to be used and because it is difficult to detect network routing policy violations, this approach allows for the possibility of data exfiltration. In particular, a malicious administrator may cause potentially sensitive traffic from one host in the network to be redirected to a machine that he controls, allowing data to be extracted in spite of a multi-configuration approach. This paper focuses on finding solutions that maintain network availability, but we hope to address this problem in future work.

5. ANALYSIS AND EVALUATION

We first present a brief analysis of the security of both our single-configuration and multi-configuration approaches. We then describe simulations we conducted on an implementation of the single-configuration approach and discuss the results of those simulations.

5.1 Security Analysis

In the single-configuration approach, a malicious administrator must be able to forge a Schnorr signature on an incorrect routing configuration in order to cause a malicious route to be used. Forging such a signature is equivalent to solving the discrete logarithm problem, for which no polynomial-time algorithm (in the number of bits) exists. Furthermore, an administrator who proposes a malicious configuration can get at most k signature shares, and because the signature threshold is $k+1$, the adversary gains no information about the value of the split secret key or signature.

In the multi-configuration approach, a malicious administrator simply pushes a routing configuration to the switch intelligence layer. The switch intelligence then probabilistically selects a routing plane to use for each flow, altering these probabilities as it gains information about a plane's performance. However, the probability of compromise is $1/n$ if the switch intelligence has no information. Since n is small, a malicious route will likely be selected, even though a loss of unavailability may be short-lived as the switch intelligence stops using the malicious plane.

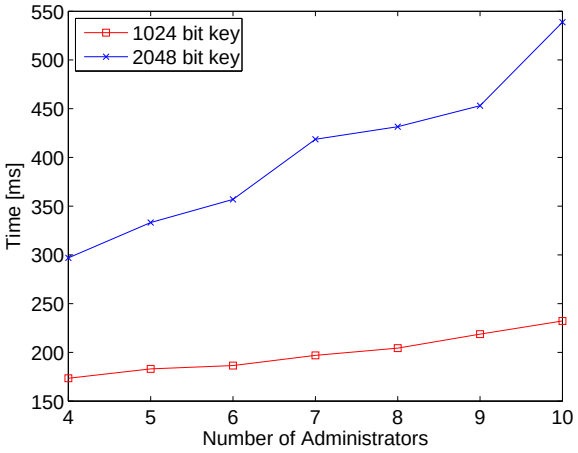

Figure 2: Recovery time from link failure detection to agreement on a fix based on the total number of administrators, using a 1024-bit or 2048-bit shared secret key.

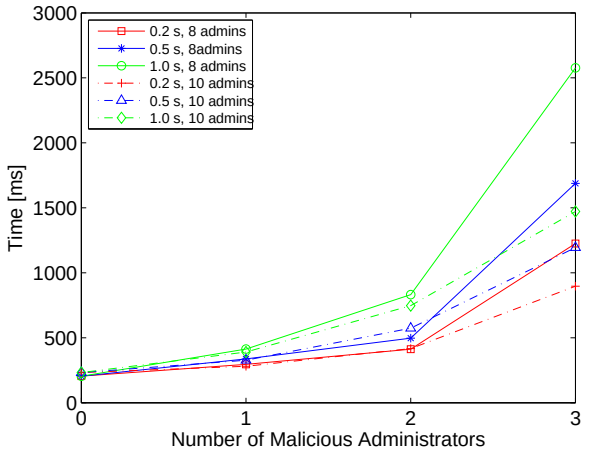

Figure 3: Duration from link failure detection to agreement on a fix based on the number of malicious administrators and voting epoch length.

5.2 Simulation and Evaluation

Our multi-configuration approach can tolerate more malicious administrators over a long term, but we did not implement it due to the challenges of defining an appropriate metric with which to evaluate route performance and of evaluating these metrics to dynamically select among different network configurations. While this approach provides several interesting topics for further exploration (§6), we chose to defer implementation until some of the existing challenges have been resolved.

We implemented our single-configuration approach in a Mininet virtual machine running Ubuntu Linux 13.10 on a host with an Intel Core i5-3470S 2.9 GHz CPU. The VM instance was allocated a single CPU core and 1 GB of RAM. Both layers of Fleet were implemented in Python using the POX[4] framework, and our code for threshold Schnorr signatures was implemented using the PyCrypto[5] and gmpy2[6] libraries, which are written in C and Python. The keys used in the threshold signature protocol were generated using OpenSSL. We simulated Fleet on randomly generated topologies with 20 switches and 50 hosts, tearing down randomly-selected links and measuring both the protocol overhead in time and the recovery time. The switches periodically exchange LLDP packets in order to update the topology and detect link failures.

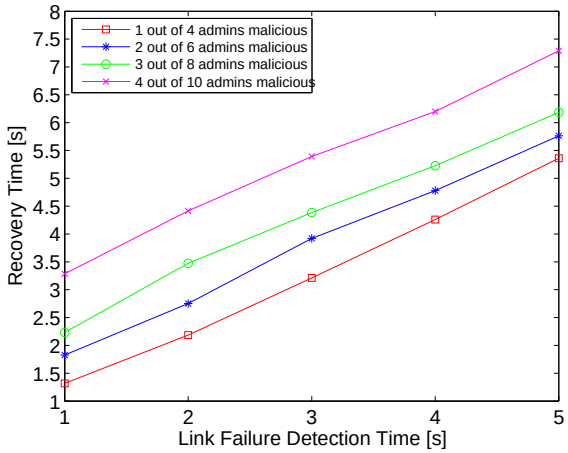

Figure 4: Total recovery time for a broken link based on how long it takes for the link failure to be detected.

Figure 2 shows the effect of two different Schnorr public key sizes on the time required by the voting protocol with a varying number of administrators n. Even with no malicious administrators, all of the signature shares will be verified when they are broadcast. For 2048-bit keys, the number of administrators has a much larger impact than in the 1024-bit case. All of our subsequent measurements therefore use 1024-bit keys.

Figure 3 illustrates the effect of the number of malicious administrators and voting epoch durations on the time overhead of the voting protocol, averaged over 100 simulated runs. As the number of malicious administrators increases we can see how the overhead increases. Longer voting epochs also increase the overhead, since an epoch with a malicious proposer results in a lack of consensus and requires a new round of voting. Because all non-malicious administrators agree, any non-malicious proposal will result in agreement on a correct configuration. Therefore a higher ratio of malicious administrators lead to more routing rounds on average. Shorter voting epochs lead to faster consensus in case a proposal gets rejected. The duration of the voting epoch can be made smaller if the inter-administrator latency is low, and thus the communication channels between administrators should be well-engineered.

Finally, we measured the total recovery time for a link from the moment of failure to the installation of a new configuration in the switch intelligence, which includes the link failure detection time as well as the protocol overhead. Figure 4 shows the recovery time in a worst-case scenario (maximum number of malicious administrators) depending on the link failure detection delay. While the number of malicious administrators affects the recovery time, the link failure detection time dominates. We thus argue that the most important factors to Fleet's performance in practice are the link failure detection time and the inter-administrator latency.

[4]http://www.noxrepo.org/pox/about-pox/

[5]https://www.dlitz.net/software/pycrypto/

[6]https://gmpy2.readthedocs.org/en/latest/intro.html

6. FUTURE WORK

In future work, we plan to extend the implementation of our single-configuration approach, deploying Fleet on real hardware. Additionally, we plan to test Fleet in several SDN deployments at ISPs and universities. Because our simulations did not scale to test large networks, we plan to use these deployments to further study how latency among administrators and switches affect our protocol in real deployments. Given that the inter-administrator latency dominates the time overhead of the voting protocol, we do not anticipate that the size of the network itself will have a significant effect on the time needed to install a new configuration in the network.

We also plan to develop a method and series of metrics for evaluating routing planes in our multi-configuration 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 an application. With such a method, we can implement and test the performance of this approach against the single-configuration approach. Since the multi-configuration approach only requires one non-malicious administrator but allows for the possibility of a malicious route being used, it would be interesting to explore the tradeoffs between maintaining availability and tolerating a greater number of malicious administrators.

Finally, we plan to extend our approaches to handle malicious administrators with respect to secrecy in order to prevent data exfiltration. A malicious administrator who successfully attacks routing may be able to exfiltrate 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 exfiltration 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.

7. CONCLUSION

In this paper we present the malicious administrator problem and propose approaches to address this problem. While similar problems have been documented before, we focus on maintaining availability in a network where one or more controllers are misconfigured. Our results show that Fleet is able to achieve this goal with little time overhead, provided that inter-administrator communication and failure detection is fast. Additionally, our work motivates other important questions in this underexplored area of SDN security, particularly in how to maintain secrecy of data in the network if there are malicious administrators.

We believe that our first steps in this area can be used to address both accidental and malicious network configurations. Corporate networks can leverage this property for such uses as expanding geographically and hiring network administrators locally, rather than displacing existing administrators. Our approaches also allow companies to train network administrators through real everyday network operations without risk of unavailability in the event of a misconfiguration. Through further work in this space, we hope to spur future improvements to securing SDN operation and management.

8. REFERENCES

[1] Ivan Damgård and Maciej Koprowski. *Practical threshold RSA signatures without a trusted dealer*. Springer, 2001.

[2] Advait Dixit, Fang Hao, Sarit Mukherjee, TV Lakshman, and Ramana Kompella. Towards an elastic distributed SDN controller. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, 2013.

[3] Open Networking Foundation. OpenFlow switch specification version 1.4.0, 2013.

[4] Ahmed Khurshid, Wenxuan Zhou, Matthew Caesar, and P Brighten Godfrey. VeriFlow: Verifying network-wide invariants in real time. 2012.

[5] Diego Kreutz, Fernando Ramos, and Paulo Verissimo. Towards secure and dependable software-defined networks. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, 2013.

[6] Juniper Networks. What's behind network downtime?, 2008.

[7] Philip Porras, Seungwon Shin, Vinod Yegneswaran, Martin Fong, Mabry Tyson, and Guofei Gu. A security enforcement kernel for OpenFlow networks. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, 2012.

[8] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *Proceedings of Advances in Cryptology (Crypto)*. Springer, 1989.

[9] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11), 1979.

[10] Seungwon Shin, Vinod Yegneswaran, Phillip Porras, and Guofei Gu. AVANT-GUARD: scalable and vigilant switch flow management in software-defined networks. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & communications security*, 2013.

[11] Douglas R Stinson and Reto Strobl. Provably secure distributed Schnorr signatures and a (t,n) threshold scheme for implicit certificates. In *Information Security and Privacy*. Springer, 2001.

[12] Amin Tootoonchian and Yashar Ganjali. HyperFlow: a distributed control plane for OpenFlow. In *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*. USENIX Association, 2010.