A Secure Computation Framework for SDNs

Nachikethas A. Jagadeesan
USC
nanantha@usc.edu

Ranjan Pal
USC
rpal@usc.edu

Kaushik Nadikuditi
USC
nadikudi@usc.edu

Yan Huang
UMD, College Park
yhuang@cs.umd.edu

Elaine Shi
UMD, College Park
elaine@cs.umd.edu

Minlan Yu
USC
minlanyu@umd.edu

ABSTRACT

SDN enabled networks of interest consist of multiple controllers forming a logically centralized control plane. A multiple controller architecture scales well, provides a platform for distributed computing amongst the controllers, while maintaining the benefits of centralized control. However, the resilience of such multi-controller SDN systems against malicious attacks has not received much research attention. In this paper, we propose a novel approach for provably secure computation in SDNs. The necessity of secure computation arises from a fault-tolerance requirement - controllers can be compromised by malicious entities with the intention to negatively influence the results and inferences made from the computation tasks; We borrow techniques from Secure Multi-Party Computation (SMPC) in cryptography to address security and fault-tolerance concerns of SDN applications. In particular, we propose a framework to allow the controllers to support a library of SDN applications whose secure and effective operation depends on distributed computation in multiple controllers. We report an initial exploration of our idea by developing and implementing a secure randomized algorithm with low overhead, for identifying heavy hitters in a network.

Keywords

SDN, security, SMPC, heavy hitters

1. INTRODUCTION

Software Defined Networking (SDN) introduces a logically centralized control plane to run diverse management applications. In practice, a logically centralized control plane is realized using multiple controllers for scalability, reliability, and availability reasons. In fact, for various current and future networks of interest, it is practically infeasible to attempt a physically centralized SDN system. As SDN gains popularity, it is important to secure the SDN infrastructure to be resilient to potential attacks.

In SDN, controllers can become high-value and attractive targets for an adversary for the following reasons. First, controllers are sinks of information collected from different switches. This includes network topology and flow-counter values. Such information can be privacy sensitive. For example, an organization may wish to protect its internal network topology or hide what type of traffic is being routed through its network. In addition, privacy policies may prohibit information from flowing between one part of the organizational network to another. Second, controllers run full-fledged software stacks including an operating system and management applications. Therefore, they may expose a much larger attack surface than switches. Moreover, threats may arise from multiple sources. In addition to software vulnerabilities that may exist in the controller software stack, malicious insiders who have privileged access to the controllers may leak sensitive information or sabotage network operations. For example, the network operator would want to make sure that traffic flow counters in the controllers stay untouched by an adversary. Manipulation of these counters could allow DDoS attacks on hosts go undetected. As another example, the network operator would want to protect network topology information from adversaries in order to protect DDoS attacks on network edges rather than on the hosts. An adversary knowing the network topology might want to launch these special types of DDoS attacks where there is small attack traffic on many paths but there is a single critical link common to all these paths which gets taken down [6].

Historically, practical and efficient realizations of SMPC have been a challenge. Given current developments regarding fast softwares to run SMPC algorithms [4], we find it suitable to propose using multi-party computation to achieve the following security guarantees in SDNs: (i) When a subset of the controllers are compromised, no sensitive information such as network topologies are leaked (ii) The network's resilience to controller failure is improved. Finally, an operator should support guarantees on fault tolerant computation whereby for a threshold k, even if $k-1$ controllers become completely non-functional due to attacker compromise or otherwise, the remaining controllers can execute the computation correctly.

Under the SMPC paradigm, multiple controllers collectively compute a given function, say the calculation of the top k flows in the network, without revealing their network data with each other. In fact, using secret sharing, we can ensure that the data share held by each controller is worthless on its own. Only when a group of controllers collude together does the result of distributed computation becomes meaningful to each member of the group. More formally, using threshold cryptography, we provide a guarantee that the computation will proceed unhindered as long as there are *any* k controllers functioning correctly. Thus an adversary would necessarily have to take down k or more controllers in order to hinder the functioning of our network. In this way, each network controller is relieved of individual duties of ensuring the privacy and security of the data that it handles. These features are inherently baked into the multi-controller architecture.

The rest of the paper is organized as follows. In Section 2, we describe the SMPC framework and how it fits SDNs in the context of design and applications. We also propose an efficient SMPC algorithm to implement heavy hitters in an SDN-enabled network. In Section 3, we explain our experimental setup to implement the heavy hitter application and state our performance evaluation results. We describe related work in Section 4, and conclude our paper in Section 5.

2. SMPC FRAMEWORK

In this section we propose our model for provably secure computation in SDNs. This section is divided into four parts. First, we briefly explain the basics of Secure Multi-Party Computation (SMPC). Second, we explore the design space of SMPC in the context of SDN security. Third, we describe a potential SDN application space for which SMPC algorithms can be designed. Finally, we design an SMPC algorithm to identify heavy hitters in a network.

2.1 SMPC Background

Imagine a situation in which a group of mutually distrustful individuals are required to cooperate in order to achieve a common objective. Specifically, consider the case in which each individual possesses some private data and the group collectively attempts to achieve the following:

1. Compute some function that may depend on the private inputs of all the participating parties.

2. Not to reveal anything more than what may be deduced from the outcome of the computation.

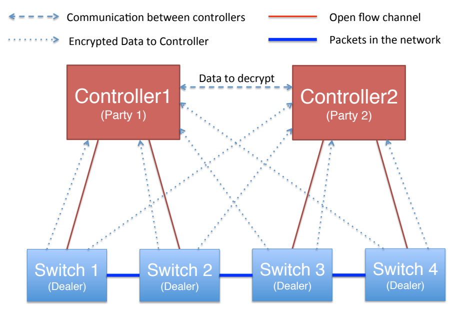

Figure 1: Secret Sharing across two controllers

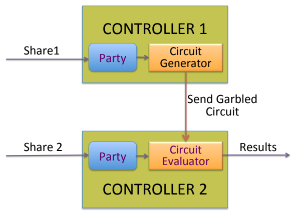

Figure 2: Two party secure computation

As an illustrative example, consider the millionaires' problem [9]. Alice has i million and Bob has j million. How do we design a protocol that allows both Alice and Bob to ascertain whether $i < j$, such that this is the only thing they know in the end (apart from their own values)? SMPC is the subfield of cryptography that provides answers to such problems where we are required to design systems that achieve a certain utility while providing provable security guarantees for our inputs. Let us make this notion more precise. We have n parties, $C_1, \ldots, C_n$, each of whom possesses a private input. Let the input of C_i be denoted by x_i. Our goal is to compute $y = f(x_1, \ldots, x_n)$ such that C_i learns only y and is ignorant about $\{x_j \mid j \neq i\}$. It is known that there exists protocols that allow us to compute any arbitrary function f in the manner mentioned above, provided we assume that not more than a certain fraction of the parties collude. Note that require each party has an equivalent part to play in computing y. In other words, we discard the solution where the input set $\{x_i \mid i = 1, \ldots, n\}$ is sent to a trusted third party that computes y and sends it to $\{C_1, \ldots, C_n\}$. Secret Sharing [8] is a concept closely related to SMPC and often used in conjunction with it. In secret sharing, we aim to divide an input among individuals in such a way so as to ensure that only an appropriately sized subset of these individuals can re-create the input. The individual shares (or the shares of any coalition that is

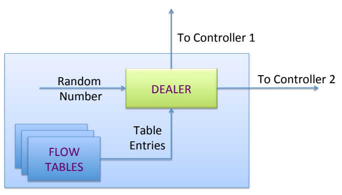

Figure 3: Secret sharing at the switch

sufficiently small) convey no information on their own. The architecture of SDNs lends itself naturally to secure computation. The data plane switches assume the role of input providers to the controllers, where the secure computation is carried out. In our model, each input is divided into different parts (or *shares*) using an appropriate secret-sharing scheme. This division of inputs to different shares is done at the switches. Each share is given to a different controller. The controllers collectively run an SMPC protocol to compute the desired function over these inputs. For example, consider the simple case where we have two controllers. One way of performing secure computation over them is by using Yao's garbled circuits [9, 10]. In this model, a controller (say, A) creates a boolean circuit that encodes his input and sends the same controller B. Controller B evaluates this boolean circuit using his input, thereby generating the result of our chosen computation. The efficiency of this approach depends on our data and the function that we are trying to compute. We base the implementation of our algorithms in an efficient framework for running garbled circuits known as *fastGC* [4].

2.2 SMPC Design in SDNs

Any generally applicable SMPC framework for SDNs must address the issue of deciding whether an application is admissible for computation. SMPC only guarantees the *execution* of any given application in a secure manner and is conspicuously non-committal about what may be released by the results. As an illustration of the potential pitfalls, consider the trivial example of an identity function which simply publishes its inputs. The input, say topology information, to any application that realizes this design is inherently insecure. While the above may be an extreme example, it highlights the need for analysis that aids in deciding the admissibility of a function.

Other design decisions pertain to the handling of network state. Controllers are often heterogeneous in nature. We may have a cluster of servers that we trust more. Some of them may have more computational power than the others. Consequently, we may desire to pre-assign the division of state amongst controllers so that we extract the maximum efficiency. Details on how one might go about doing this are a subject for future discussion. For the purposes of this paper, we make the simplifying assumption that all state is handled equally by the controllers.

SMPC algorithms that can handle a large class of functions tend to trade-off performance to achieve their generality. It follows that applications that are of sufficient importance to our network or those that have strict latency requirements may warrant the development of custom algorithms that exploit the structure inherent in the problem to achieve the desired performance. Keeping that in mind, we posit that SMPC is generally better suited for applications that allow for offline processing of data as opposed to those that demand results in real time.

As shown in figure 2.2, each switch has a component named *dealer* that collects the required flow counter values and prepares them to be split across multiple controllers. Similarly, each controller has a component named *party* that collects such data from the switches and engages in SMPC from with the other controller. Figure 2.1 indicates conceptually how two controllers would run a circuit implementation of SMPC. Controller 1 has a generator module that generates a garbled circuit for heavy hitters using the data of stream 1. Controller 2 has an evaluation module that evaluates this circuit using stream 2 data. The output of this evaluation is published, which is our required output.

2.3 Applications in SDN

Our proposed SMPC framework can support many popular and practically relevant applications in a multi-controller SDN system. These applications may be broadly classified into two classes, namely routing and traffic analysis. Routing applications typically involve graph computations such as finding paths that posses some property or properties of interest. Traffic analysis applications typically analyse counter data to extract interesting patterns or information. Below, we list out some interesting applications and outline the challenges involved in computing them in a secure manner.

- **Streaming**: In a streaming applications, we are given a list (*stream*) representing items collected across a moving window of size T. Our objective is to release the most frequently used item within this moving window. A straw man proposal for implementing the above algorithm using two party computation is as follows. The stream of items is secret shared across two controllers. An oblivious linear scan of this list follows, where we maintain a count for the most frequent item we have encountered so far. For every new item in our list, we increment the counter if it's the same with the current item. We emphasize that the challenge in realizing the above lies in making each step data oblivious and secure while hitting the performance requirements of streaming.

- **Heavy Hitter**: In a heavy hitter application we are concerned with identifying the top k sources that send most traffic into our network. Flow counters in each switch are secret shared across multiple controllers and who then engage in SMPC to identify the top k sources. In the next subsection, We present our algorithm that implements heavy hitters.
- **DDoS**: A DDoS application shares similar properties with heavy hitters. While a heavy hitter application would identify the top k sources that sent traffic into the *network* our goal in the DDoS application is to identify the top k sources that sent traffic to a particular destination in the network. Indeed, as may be seen in next subsection, heavy hitters may be used a subroutine in DDoS.
- **Cuts**: Identifying cuts in a graph is important for many network applications. For example, they are often used to aid in the deployment of VLANs so as to minimize the communications between any two VLANs. They also help in answering questions regarding the placement of access control between connected VLANs. For this application, each switch secret shares its view of the network topology across multiple controllers who run SMPC. Then challenge here lies in the development of scalable and efficient algorithms.

2.4 Algorithm to Find Heavy Hitters

We propose the following algorithm for identifying the top k flows in a network. Before diving in to the algorithm, let us make our problem statement more precise. We have two switches, each of which has a stream of tuples of the form (`seq`, `IP`, `nPackets`). `IP` represents the 32-bit IP address associated with the source of the flow. `nPackets` represents the number of packets sent from that IP address in this flow. Our goal is to identify the k IP addresses that sent the maximum number of packets, for each window of certain size (say t).

The streams containing flow counter values from the switches are secret shared between the two controllers. Identifying the top k flows from both streams involves two steps. Firstly, we merge the streams obtained from different switches into a single stream, taking care to appropriately handle different flows emerging from the same source. This step is implemented in a circuit. Secondly, we sort the merged stream using an oblivious sorting algorithm. A data oblivious sorting algorithm performs no compare-exchange operations that depend on the relative order of the elements in the input array. Thus an adversary cannot infer anything about the stream by inspecting the data access patterns of the sorting algorithm. Our algorithm uses randomized shell sort [2] for data oblivious sorting. The complete pseudo code for the algorithm is given below.

```
Data: Stream1, Stream2
Result: The top $k$ flows
Stream = (Stream1,Stream2);
Stream = ObliviousSort(Stream);
for $i \leftarrow 1$ to length(Stream) do
    if Stream[i].IP == Stream[i+1].IP then
        Stream[i+1].nPackets +=
        Stream[i].nPackets;
        Stream[i].nPackets = 0 ;
    end
end
Stream = ObliviousSort(Stream);
Print(Top $k$ records in Stream);
```

Algorithm 1: Secure heavy hitter detection

The algorithm for DDoS attack detection is similar. Indeed, the above method for heavy hitter detection may be included as a subroutine in our algorithm. This is easily seen once the problem definition is made explicit. In DDoS detection detection, we are interested in identifying the k sources that sent the maximum number of flows for a given destination. Switches 1 and 2 each has a stream of tuples of the form (`seq`, `srcIP`, `destIP`, `nPackets`). Note that tuple remains unchanged from that of heavy hitters, except for the addition of `srcIP`. These steams are secret shared between the controllers as before. Now, the controllers filter the streams to retain only those entries that correspond to the `srcIP` that they are interested in. The filtered stream is then fed into the heavy hitter method above, after removing the `srcIP` field. The complete pseudo code is omitted for the sake of brevity.

It bears mentioning that the above two examples represent fairly sophisticated instantiations of SMPC in SDNs. Simpler applications, involving counting and comparison of packets or flows based on header field values, are supported. Indeed, support for the above algorithms is predicated on the existence of such simpler primitives.

3. PERFORMANCE EVALUATION

In this section, we describe our proposed experimental setup and analyse the experimental results on a heavy hitter application. The first part of the section describes the experimental setup, which is then followed by explanations of our experimental results. Historically, practical and efficient realizations of SMPC has been a challenge with progress being made in recent years. Figure 2 shows the topology we used to test the performance

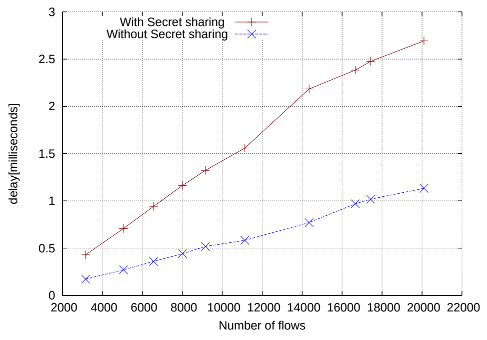

Figure 4: Delay for bit-by-bit secret sharing

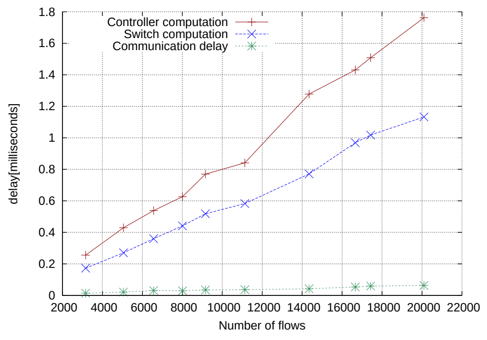

Figure 5: Delay at switches and controllers

of our secret sharing mechanism. We used Mininet to set up the given topology. Switches 1-4 are L2 learning switches, while controllers 1 and 2 are POX instances. POX does not natively support communication between multiple controllers. We implemented the required communication channels between controllers using sockets. The implementation overheads at the switches and controllers for engaging in bit-by-bit secret sharing is shown in Figure 3. Note the increasing gap between the delay comparing the secret sharing and non-secret sharing mechanism for increasing number of flows. This is due to the increased computation overheads. Figure 3 shows the overhead, in terms of the time required for computation at switches and controllers, for running bit-by-bit secret-sharing, in addition to the communication delay between switches and controllers. It is seen that the overhead is low with respect to absolute time guarantees for the different types of computation delays, as well as the communication delay. We also implemented the heavy hitter algorithm on the fastGC framework using traces from the CAIDA data set. The results of our implementation are shown in figure 5. to evaluate the performance of our heavy hitters implementation. The results are given in figure 5.

4. RELATED WORK

Security in SDNs is a relatively new topic. In this section we review related work in this area. We structure this section in two parts. The first part compares our work with work related to preventing attacks in SDNs, and the second part compares our work to existing techniques that can potentially achieve the same purpose as our proposed techniques.

Prevention of Network Attacks: [5] attempts to prevent attacks by assigning virtual IPs to hosts. These IP addresses are varied sufficiently often so as to make it difficult for an adversary to co-ordinate an attack on the host. Our work protects the SDN controllers instead of the hosts and is thus complimentary to the above, even though our proposed technique could be extended to prevent an attacker to successfully coordinate attacks on hosts in data center and enterprise networks. FortNOX [7] focuses on the detection and prevention of flow rule conflicts. They protect against a malicious application that attempts to insert flow rules that circumvent that of a SDN security application. This again is orthogonal to our approach of ensuring that each application does not have any access to raw network data. More specifically, the above mentioned works focus on the prevention of select network attacks while through our work we severely limit the negative effects due to an attacker even when controller attacks do succeed. To achieve our goal, we use SMPC as a radical way of securing software defined networks.To the best of our knowledge this is the first application of SMPC in SDNs. A proposal for using SMPC in an inter-domain setting can be found in [3]. The authors study the use of SMPC for securing BGP routing information but identify the running time for their algorithm as a matter of concern. In contrast to [3], our proposed SMPC algorithm is applicable to a multi-controller SDN setting, and could also potentially be applied to an inter-domain setting. Moreover, we emphasise the development of efficient SMPC algorithms that run within an acceptable time frame as suited to the applications at hand. We identify the development of efficient SMPC algorithms for multi-domain networks as a line of future research.

Similar Techniques: One could argue that techniques like differential privacy and homomorphic encryption would help to secure network data. In differential privacy [1], a data distortion method, all computation is carried out on noisy network data. The amount of noise to be added depends on the specific function we are trying to compute. However, this method often suffers from lack of utility. There is an inherent conflict between the amount of distortion we need to apply to secure our data versus the utility that we expect to gain from the system. In SMPC however, the computation is

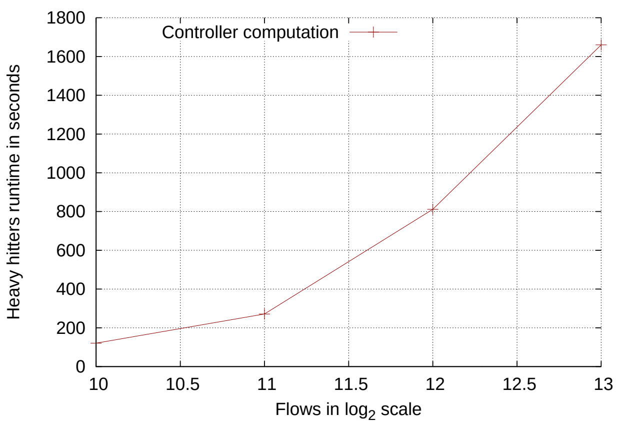

Figure 6: Heavy hitter runtime

exact. Our output is precise and accurate while simultaneously protecting the network data. Homomorphic encryption is sometimes used in place of SMPC with secret sharing. In a homomorphic system, computation is done on ciphertext such that the resulting ciphertext generates the correct outcome when decrypted. However, fully homomorphic systems are typically resource-wise expensive. SMPC, along with secret sharing, allows us to ensure the security and privacy of our network data while allowing us to compute our network wide functions efficiently using modest resources.

5. CONCLUSION

In this paper we developed a novel framework for provably secure computation in SDNs via distributed computation amongst multiple controllers. We borrowed techniques from secure multiparty computation (SMPC) in cryptography and addressed security and fault tolerant concerns of SDN applications. In particular, we investigated the secure computation of heavy hitter and DDos applications, and showed that such computations can be performed with low overheads using efficient SMPC algorithms. Our proposed contribution is generic enough to support a library of SDN applications.

As part of future work, we plan to achieve the following goals: (i) implement a wider set of securely computable SDN applications, (ii) design efficient low overhead SMPC algorithms for certain classes of applications, e.g., graph computations relating to paths an cuts, (iii) extend our proposed framework to protect against the case where a malicious controller can modify the flow counter values, and (iv) design, develop, and implement a *generic* framework running on multiple controllers that implements secure computation algorithms for an existing SDN application library, and also imparts flexibility to accommodate new applications. In this paper, we only designed a *specific* prototype of the aforementioned framework that can potentially run on multiple controllers.

6. REFERENCES

[1] C. Dwork. Differential privacy: A survey of results. In *Proceedings of the 5th International Conference on Theory and Applications of Models of Computation*, TAMC'08, page 119, Berlin, Heidelberg, 2008. Springer-Verlag.

[2] Michael T. Goodrich. Randomized shellsort: A simple data-oblivious sorting algorithm. *J. ACM*, 58(6):27:1–27:26, December 2011.

[3] D. Gupta, A. Segal, A. Panda, G. Segev, M. Schapira, J. Feigenbaum, J. Rexford, and S. Shenker. A new approach to interdomain routing based on secure multi-party computation. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, HotNets-XI, pages 37–42, New York, NY, USA, 2012. ACM.

[4] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, pages 35–35, Berkeley, CA, USA, 2011. USENIX Association.

[5] J. Jafarian, E. Al-Shaer, and Q. Duan. Openflow random host mutation: Transparent moving target defense using software defined networking. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 127–132, New York, NY, USA, 2012. ACM.

[6] M. Kang, S. Lee, and V. Gligor. The crossfire attack. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, pages 127–141, Washington, DC, USA, 2013. IEEE Computer Society.

[7] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu. A security enforcement kernel for openflow networks. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 121–126, New York, NY, USA, 2012. ACM.

[8] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.

[9] Andrew Chi-Chih Yao. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science, 1982. SFCS '08*, pages 160–164, November 1982.

[10] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science, 1986*, pages 162–167, October 1986.