Routing-Verification-as-a-Service (RVaaS):

Trustworthy Routing Despite Insecure Providers

Liron Schiff[1] Kashyap Thimmaraju[2] Stefan Schmid[3]

[1] Tel Aviv University, Israel [2] TU Berlin & T-Labs, Berlin, Germany [3] Aalborg University, Denmark & TU Berlin, Germany

***Abstract*—Computer networks today typically do not provide any mechanisms to the users to learn, in a reliable manner, which paths have (and have not!) been taken by their packets. Rather, it seems inevitable that as soon as a packet leaves the network card, the user is forced to trust the network provider to forward the packets as expected or agreed upon. This can be undesirable, especially in the light of today's trend toward more programmable networks: after a successful cyber attack on the network management system or Software-Defined Network (SDN) control plane, an adversary in principle has complete control over the network.**

This paper presents a low-cost and efficient solution to detect misbehaviors and ensure trustworthy routing over untrusted or insecure providers, in particular providers whose management system or control plane has been compromised (e.g., using a cyber attack). We propose Routing-Verification-as-a-Service (RVaaS): RVaaS offers clients a flexible interface to query information relevant to their traffic, while respecting the autonomy of the network provider. RVaaS leverages key features of OpenFlow-based SDNs to combine (passive and active) configuration monitoring, logical data plane verification and actual in-band tests, in a novel manner.

I. Introduction

While improving the security of the Internet routing system has been a prime concern for many years already, the interface between Internet users (including companies) and the network provider (e.g., the carrier or datacenter operator) has received little attention. Today, the user typically does not even have any means to *specify* desired and undesired routing paths (e.g., using white or black lists), and even less is supported in terms of *verification*. Rather, it is often implicitly assumed that the user needs to trust its network provider, including its network management system software, unconditionally.

While traceroute and trajectory sampling tools may be sufficient to verify routes in regular networks [6], [8], and may still perform well in the context of faulty and heterogeneous networks [7], [38], they are insufficient in non-cooperative and adversarial environments: an unreliable network operator may simply not reply with the correct information, also breaking any scheme based on packet labeling or tagging [30], [46]. Even more challenging than verifying the used paths, is to *test avoidance*, i.e., verifying that certain paths have not been taken and certain destinations have not been reached [22].

The threats introduced by untrusted providers are manifold. In particular, routing can be compromised even in scenarios where the provider itself is in principle benign. For example, over the last years, numerous flaws have been found in network management systems [27]. The problem is exarcerbated in Software-Defined Networks (SDNs) [14]: A new SDN control plane may be vulnerable to cyber attacks, which, given the important role the SDN controller plays compared to more distributed legacy network protocols, is particularly worrying: an adversary with access to the control plane can in principle arbitrarily change the network forwarding behavior, and violate security policies (e.g., breaking logical isolation domains between health care providers [17]) or exfiltrate confidential traffic. Today, clients do not have a means to reliably verify the data plane configuration.

At first sight, the problem seems to be an inherent one: as soon as the packet enters the provider network, its fate is inevitably decided by the provider and its network management system and software control plane. While a (possibly signed) acknowledgment from the receiver may eventually confirm to the sender that the packet has successfully arrived, this is insufficient as it does not provide any information about which paths have been taken and which (possibly *additional*) destinations have been reached. The problem is particularly cumbersome in the context of high-performance networks where cryptographic per-packet operations (like encryption, signatures, etc.) are out of question.

This paper is motivated by the question whether it is possible to reduce the seemingly inevitable trust assumptions in the network provider, and to empower the user to verify the routes taken and destinations reached by its packets. Ideally, the resulting solution should also not introduce significant computational overheads, and also respect the autonomy of the network operator: security and business critical details of the underlying topology should not be revealed.

A. Our Contributions

This paper presents *Routing-Verification-as-a-Service (RVaaS)*, a novel network service which allows users (or more generally: *clients*) to query and verify relevant properties of the network routes installed on their behalf. RVaaS removes the need for users to unconditionally trust the network providers to forward their packets according to the agreed upon routing policies, and also accounts for the possibility that operators or control software is compromised, e.g., due to a cyber attack.

RVaaS is based on passively and actively monitoring network configurations, and on the in-band interception of user request messages (e.g., using OpenFlow *Packet-ins*). Upon a query request, RVaaS performes a static packet trajectory analysis (identifying relevant endpoints), and actively issues verfication

packets and client authentication tests (e.g., the verify that endpoints are legitimate).

RVaaS features the following properties:

1) **Verifiable routing properties:** Users can learn about and verify, through a flexible interface, relevant information related to the routes taken by their packets, such as *the set of destinations*, or whether *fairness conditions* are fulfilled (e.g., regarding bandwidth allocations). For example, users can verify that their traffic is not routed in a way which violates privacy, e.g., is not exfiltrated or routed through certain geographic regions.
2) **Confidentiality:** The autonomy of the provider is preserved, and security or business critical topological details can be kept confidential.

One attractive feature of our approach is that it allows users to issue very general queries, which are not limited to connectivity alone, but may also include geographic, performance and fairness aspects. Another feature is the provided *modularity*: queries may not be limited to a single provider but may recursively span consecutive networks along a route.

To provide the RVaaS service, it is sufficient to deploy a single secure server, somewhere in the network; additional (independent) servers can increase the security further. These servers do not have to inspect live traffic, and have low resource requirements; they also does not come with strict latency requirements.

B. Paper Scope and Novelty

We emphasize that the goal of our approach is to empower the users to *detect* misbehavior, as opposed to *prevent* misbehavior. In other words, alone, our approach is unable to ensure a user's packets will not traverse certain network regions or reach certain destinations. However, we believe that the possibility to detect misbehavior can often be a strong disincentive to deviate from the correct behavior. Moreover, we in this paper do not consider the orthogonal question of how a user should specify its desired and undesired routes to the network provider.

Generally, we believe that our work assumes an interesting position in the secure routing space. While there has been much interest in securing the inter-domain routing protocol or in dealing with unreliable data plane components, we study how to reduce trust assumptions in the entity installing the rules on the routers in a single administrative domain. Moreover, we make the case for marrying verification mechanisms in the "logical space" (e.g., *which routes exist?*), with physical verification mechanisms in the data plane (e.g., *which host destinations are actually reached?*).

We also note that while for $RVaaS$, any secure server is in principle sufficient, our architecture can also benefit from the advent of novel hardware developed in the context of *Intel SGX* [9], [16], [23], [36]. In this respect, we see our work also as an interesting case study demonstrating a new application of this technology in the context of secure routing.

C. Organzation

The remainder of this paper is organized as follows. Section II provides necessary background on SDN/OpenFlow. Section III introduces our model together with some terminology. RVaaS is described in detail in Section IV. After reviewing related work in Section V, we conclude our contribution in Section VI.

II. Background

The solution proposed in this paper is tailored for Software-Defined Networks (SDNs), and we will provide the necessary background accordingly in this section.

In a nutshell, Software-Defined Networks (SDNs) outsource and consolidate the control over the data plane devices (the switches or routers) to a logically centralized software controller. This decoupling introduces flexibilities and innovation opportunities, as the control plane can now evolve independently from the constraints of the data plane [11]. OpenFlow is the de facto SDN protocol standard today. OpenFlow is based on a match-action concept: OpenFlow switches store rules a.k.a. flow entries (installed by the controller) consisting of a match and an action part. A packet matched by a certain rule will be subject to the associated action. For example, an action can define a port to which the matched packet should be forwarded, or add or change a tag (a certain part in the packet header). In OpenFlow networks, the distinction between switches and routers disappears: an OpenFlow switch can match (and apply actions to) not only layer-2 but also layer-3 and layer-4 header fields.

An OpenFlow switch can (and should) be connected to one or multiple controllers via an authenticated and secure communication channel (e.g., SSL/TLS).[1] Thus, only legitimate controllers can send rule updates to the switch.

In order for the controller to learn about newly arriving flows, OpenFlow switches can forward packets to the controller by sending them within so-called *Packet-In* messages. Similarly other events (link failures, switch errors, etc.) are reported as well to the controller using dedicated OpenFlow messages.

As a reaction to such events, a controller may want to change the installed flow on the switch (using *Flow-Mod* commands) or explicitly send packets out from the switch (using *Packet-Out*). Moreover, to stay informed about the current configuration of a switch (the existing flow entries), the controller should use the OpenFlow *add flow monitor command*.

III. Model and Threat

We consider a software-defined network servicing multiple clients which are geo-spatially distributed. The client and provider roles are defined as follows:

1) **The Clients (or Users):** We will refer to the users or communication endpoints of the network as the *clients*. Each client may be connected to the network infrastructure at multiple access points (switch ports),

[1] We note however that according to a 2013 study, only 2 out of 8 OpenFlow switches and 1 out of 8 (popular) OpenFlow controllers fully support. [2]

and request connectivity and routing services (regarding his access points) from the provider.

2) **The Provider:** The provider running the software-defined network consists of two parts:
   a) *Network management system and control plane:* A software in charge of defining and installing the device configuration (e.g., routing policies), within the constraints defined by the clients.
   b) *Infrastructure:* Routers (resp. OpenFlow switches) and links.

We consider a threat model motivated by *cyber attacks*: an external attacker which compromised the network management or control plane (e.g., using a Trojan or a remote cyber attack) aims to change the data plane configuration, e.g., to divert client traffic to unsupervised access points or through undesired jurisdiction, thereby putting the security of the network and the traffic privacy at risk. However, while the network management system and control plane may be hacked, we assume the infrastructure to be secure: The question of what security properties can be guaranteed in scenarios where control planes can be compromised and malicious while the data plane is correct, is scientifically interesting on its own right. However, we argue that the question is also a practically relevant one, in three respects:

- While a cyber attacker (not an insider!) may be able to hack the management and control plane, it is impossible to change physical configurations from remote locations.
- In the context of network virtualization, the physical infrastructure provider and the virtual network operator are often considered two different roles. In this respect, our model can be understood as a case study of how to deal with a malicious virtual network operator.
- Our model can also be motivated by the current trend toward more trusted hardware (see, e.g., Intel SGX).

The clients can also be untrusted in our model, and may for example not inform the sender about having received packets, or may try to infer confidential details about the network topology.

Our objective is to enable a trustworthy routing, by empowering a client to find out about and verify relevant properties of the routing applied to its packets (e.g., the set of reached destinations). Moreover, the autonomy of the provider should be preserved. In particular, clients should not be able to infer the topology or critical features (like bottlenecks) of the network itself. Finally, details of the client should not leak: the provider should not learn about their queries (whose content is somewhat confidential).

In general, we assume a high-speed network (e.g., Internet backbone or datacenter), where per-packet encryptions or public key operations are hardly used due to the high costs of deploying and maintaining them. Concretely, we rule out signed logs in every packet, per-flow state in forwarders (which stymies fail-over), and ideally not even per-flow public key operations.

In summary:

- Switches are trusted (e.g., bought from a trusted vendor), and are initially configured correctly.
- Internal network ports are known, and follow a well-defined wiring plan.
- Links are trusted: no physical taps are installed.
- Switch to RVaaS controller sessions are secured, using encrypted OpenFlow sessions and apriori configured switch certificates for authentication.

IV. Trustworthy Routing

We will first discuss the main ideas and concepts behind Routing-Verification-as-a-Service (RVaaS). Subsequently, we give an example and discuss extensions and limitations.

A. Main Concepts

At the heart of RVaaS lies a flexible interface which allows the clients to query relevant information related to how their packets are being forwarded in the network. The interface allows clients to ask questions such as:

- Which destinations (resp. other clients and hosts) can be reached by the traffic leaving my network card? This question may also be made more specific, e.g., constrained to traffic within a certain header space.
- For which sources (e.g., other clients, hosts) currently exist routing paths which can reach my network card? Again, the question can be made more specific.
- Is my traffic forwarded fairly, e.g., according to network neutrality principles?

Generally, queries related to connectivity, path lengths optimality, traversed geographic regions, traffic shaping, quality-of-service (e.g., dedicated bandwith), etc. are supported. A client may also request a compact representation of the transfer function of its offered routing service.

Through attestation, the client can verify that $RVaaS$ is the one that securely responds to its queries. Moreover, the provider makes sure that the correct $RVaaS$ application is operating on the server, and not a fake one that may leak sensitive information regarding the infrastructure or clients.

RVaaS is based on a passive-active approach: events in the data plane are monitored, and analyzed in the control plane; upon a client request, endpoints are actively tested and authenticated. RVaaS can be realized using a stand-alone OpenFlow controller, henceforth called *RVaaS controller*, which monitors the configurations of all the switches, and which may send and receive (resp. intercept and inject) specific messages in-band, in order to communicate with the clients. This controller is different from possibly additional controllers used by the network provider to manage the network, and should be trusted. Also, while a single one is in principle enough, different entities (e.g., a certification authority) may provide different independent controllers, reducing the attack surface further.

In order to provide its service, the RVaaS controller performs three different functions: passive or active configuration monitoring, logical data plane verification, and actual in-band testing using client interaction.

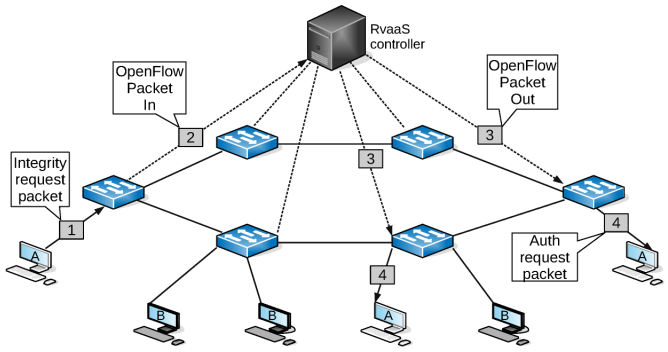

Fig. 1. A client makes an integrity request to the RVaaS controller. The RVaaS controller analyzes the request and then dispatches Auth(entication) request packets to relevant clients.

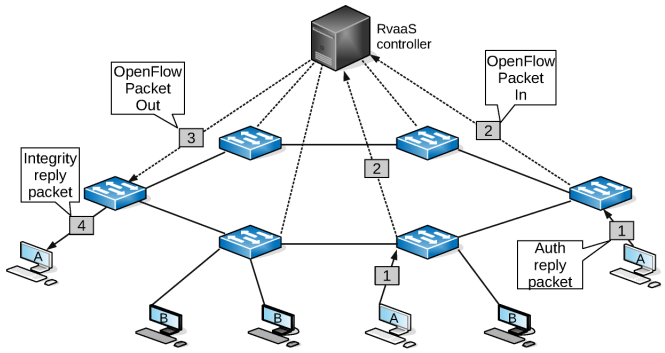

Fig. 2. Relevant clients send Auth(entication) reply packets back to the RVaaS controller which collects the replies and sends them back to the requesting client.

1) Configuration Monitoring: Our stand-alone RVaaS controller is secured (i.e., installed correctly, and cannot be influenced by the network provider) and connected to all switches, via authenticated and encrypted OpenFlow sessions. Through these sessions, the controller maintains an up-to-date snapshot of the network configuration, either passively (monitoring events) or actively (query the switch state or issue and later intercept LLDP like packets through all internal ports). This information is acquired in a manner that cannot be exploited by an untrusted operator.

2) Logical Verification: In order to answer client queries, relevant routes are computed in the logical space, given the current network snapshot collected by the RVaaS controller. For example, the RVaaS controller may perform Header Space Analysis [17], or simply emulate the network based on the current configuration.

3) In-band Test & Client Interaction: The RVaaS controller has active access to the network traffic in order to collect user requests/queries. Such client messages have distinct properties (e.g., destination address, VLAN tag, etc.) that allow them to be matched at the (ingress) switches and reported to the controller. In addition each client knows the public key of the RVaaS controller, allowing it to encrypt messages and verify authenticity of the results.

In some cases, answering user requests, involves sending further requests (e.g., using OpenFlow packet-out commands). For example, these packets trigger destination clients to respond to the querying clients, in an authenticated manner (*authentication requests*). Towards this end, clients run a software which responds to our authentication requests, in user space, publishing themselves by sending a UDP packet (with a specific magic header field value which can be intercepted and traced back to the origin, due to the logically centralized view). Concretely, in order to not expose $RVaaS$ to cyber attacks itself, our solution is based on in-band interception (namely OpenFlow packet-ins) of user request messages (e.g., using a magic header value); responses are sent via packet-outs. That is, $RVaaS$ is only reachable via a very simple OpenFlow interface and indirectly; no special protocols and servers are needed.

In general, the security of our architecture relies on the integrity of the secure hardware, as well as on the frequency of the network snapshot it takes. In particular, $RVaaS$ needs to ensure that it receives all the relevant updates from the switches. This is guaranteed in our setting where OpenFlow switches are reliable. However, additionally, it is also possible for $RVaaS$ to proactively query the switches for their current configuration. The latter however needs to happen at random times, which are hard to guess for the adversary. This is important as otherwise, the adversary may simply set the correct rules for the short time periods in which the box checks the configuration. Short term reconfiguration attacks can also be prevented by maintaining some history. Regarding the confidentiality of the network topology and respecting the provider's autonomy, our architecture offers many flexibilities. In particular, queries can be limited to learn only about endpoints, but nothing about the actual routing paths inside the network.

B. Case Studies

Let us consider some relevant case studies. We refer the reader to Figures 1 and 2 for some illustrations.

1) Isolation Checks: One of the most fundamental security queries supported by RVaaS regards whether the sub-networks where different clients are located, are isolated from each other: no client can gain access to another client's network except through some access points used by the client. Failing to guarantee such a requirement makes the client vulnerable to join attacks in which an attacker first manipulates the network operation, and secretly adds access points which can then be used to access and/or damage client assets (such as private data or hardware) managed by the network.

Our system can detect violations of isolation as follows: a client request, sent through an access point (the request point), is intercepted at an ingress switch and reported through a *Packet-In* to the system server. Based on its current network view, the server computes all the possible access points that can communicate with the request point, e.g., using reachability tests based on Header Space Analysis. Given these

access points, the server issues a *Packet-Out* message at the corresponding outgress ports of the network. The hosts behind these ports respond, with authenticated messages, which are intercepted and reported to the server. The collection of these authenticated messages are then forwarded to the querying client, which can verify the correctness and authenticity of these destinations.

Note that the server also forwards to the client the total number of authentication requests that were made, such that it can detect cases where some access points did not respond.

2) Geo-Location Checks: As a second case study, we present a query made by a client to discover the locations where its traffic passes through. This is relevant, e.g., in scenarios where different jurisdictions exercise different privacy policies regarding user data. First we require that the locations of all the switches (and preferably also the links) are known to the RVaaS controller. These locations can be revealed/estimated in each of the following ways: (1) either disclosed by the infrastructure provider; (2) collected from the clients themselves in a crowd-sourcing manner: clients can e.g., report their geographical locations which allows $RVaaS$ to guess the location of nearby switches; (3) or passively inferred from clients traffic, e.g., using geo-IP mappings, domain name records information, time zone estimations, etc.

Given a client geo-location request, the RVaaS controller uses header space analysis to find out all the intermediate and end point switch (and link) of any possible route for the client. Then using the locations of the network provider components, the set of locations exposed to the client traffic is computed and sent to the client.

C. Extensions and Discussion

This section provides a discussion of our approach, and identifies limitations and extensions.

a) Multi-Provider Settings: While we have described our architecture for a single-provider setting, in principle, our approach can also be used across multiple providers. In this case, queries need to be propagated between the $RVaaS$ servers of the respective providers. Clearly, the trust assumptions then need to be extended accordingly, to those servers as well.

b) Supported Queries: In principle, a wide range of queries can be supported within our framework, beyond simply identifying reachable clients. For example, given the up-to-date network view, performance and fairness related queries by clients may be answered. Moreover, $RVaaS$ could be used to check whether allocated routes and meter tables meet network neutrality requirements. Moreover, a slightly more complex service may also maintain some history of the recent past, allowing $RVaaS$ for example to traceback the ingress port of an attack.

V. Related Work

Our paper assumes an interesting new position in the secure routing space. Arguably the most intensively-studied problem in the secure routing literature regards how to ensure authenticity and correctness of topology propagation and route computation across multiple untrusted and insecure domains, e.g., by extending [18], [31] or redesigning [20], [45] the Border Gateway Protocol (BGP). Moreover, the problem of how to design secure routing protocols which allow to deal with untrusted and insecure switches and routers currently experiences a rennaissance [28], [29], [34]. In contrast, we in this paper investigate mechanisms which empower *the user* to deal with untrusted or insecure operators, subject to a cyber attack (from an external adversary without physical access). Our problem is also different from the recently introduced and interesting malicious administrator problem [25], [37]: in that problem, it is assumed that a network is redundantly managed by multiple administrators or controllers, out of which only a minority can be malicious. This allows for simple (yet crypto intensive) secure solutions based on threshold objects and majority decisions. In the context of operator networks, such a redundancy is not available, and to the best of our knowledge, the threats introduced by a malicious network operator have not been studied before.

We are not the first to identify new security-related opportunities and challenges introduced by the software-defined networking paradigm [14], [21], [32], also regarding traffic monitoring [13], [41], [43]. While the static logic of $RVaaS$ can be implemented using Header Space Analysis (HSA) [17], over the last years many alternative tools have emerged [19], [24], [39]. These tools in turn rely on early works on reachability by Xie et al. [42], and are not limited to switches and routers but can also be employed, e.g., in the context of firewalls [1], [10], [26], [44]. Our work is orthogonal in the sense that RVaaS can benefit from such systems in order to implement its query interface, while performing the required authentication requests in the data plane. That is, RVaaS in some sense combines data plane [3], [5], [12], [33] and control plane [4], [15], [40] query systems, and issues a minimal amount of requests in the data plane (e.g., to collect information about attached clients). In this sense, our work is also orthogonal to recent literature aiming to improve the latency at which network monitoring information can be retrieved [35].

VI. Discussion and Conclusion

This paper initiated the study of trustworthy routing architectures in the context of hacked and untrusted network management systems and control planes, as well as malicious virtual network operators. We have identified different requirements and different roles in such a setting, and provided a first solution which, based on a secure but simple hardware, allows to decouple the roles and empower clients to verify routing properties while preserving the autonomy of the operator, e.g., by respecting the confidentiality of topological details.

While the underlying concepts may be more general, $RVaaS$ is particularly well-suited for SDNs based on OpenFlow: OpenFlow's match-action interface provides an ideal technological basis for our approach. In particular, OpenFlow enables a simplified monitoring of different equipment, the interception

and injection of packets in order to communicate with the clients, without affecting existing services, and the centralized view and analysis of the collected configuration.

Clearly, at least initially, $RVaaS$ targets power users, but in the longer run, may also be incorporated into security/privacy products directly, and made available to end users. Our work also raises the question why an operator would be willing to install $RVaaS$. Besides the possibility to consolidate logical network view and physical configuration (e.g., in scenarios where the operators does not necessarily trust the SDN controller software to be perfectly correct), we also see economic incentives: a telco hosting one or multiple (independent) $RVaaS$ servers may appear to be more trustworthy to their customers, which can constitute a business advantage. For instance, customers relying on security-critical networks, such as governmental networks, are likely to prefer certified telcos, which offer an independent means of verification.

In general, we believe that our work assumes an interesting new perspective on the classic topic of secure routing, in several respects. For example, we believe that our distinction of network operator from physical infrastructure provider is an interesting and timely one, beyond the considered cyber attack threat model: in the context of network virtualization and with the ongoing infrastructure liberalization trend, network operators are more and more seen as a business role which may be independent from the infrastructure owner. Moreover, while our assumption of trusted infrastructure is a strong one, we believe that it constitutes more than an academic exercise: given today's trend toward trusted hardware, our work is timely and provides an interesting new look on this trend from a networking perspective.

We understand our work as a first step. In particular, while we show the potential for a more trusted routing in less trusted environments, much more research is required to understand the minimal assumptions required to implement such an architecture, as well as to understand the fundamental tradeoffs in terms of security and performance. It is also clear that there are inherent limitations to such a solution. For example, it seems impossible to deal with untrusted network operators who also have physical access to the network, at least in the classical, non-quantum physics world.

ACKNOWLEDGMENTS

Research supported by the German Federal Office for Information Security (BSI). In particular, the authors would like to thank Jens Sieberg.

REFERENCES

[1] Y. Bartal, A. Mayer, K. Nissim, and A. Wool. Firmato: A novel firewall management toolkit. *ACM Trans. Comput. Syst.*, 22(4):381–420, Nov. 2004.

[2] K. Benton, L. J. Camp, and C. Small. Openflow vulnerability assessment. In *Proc. ACM SIGCOMM HotSDN*, 2013.

[3] K. Borders, J. Springer, and M. Burnside. Chimera: A declarative language for streaming network traffic analysis. In *Proc. USENIX Conference on Security Symposium*, 2012.

[4] X. Chen, Y. Mao, Z. M. Mao, and J. Van der Merwe. Decor: Declarative network management and operation. *SIGCOMM Comput. Commun. Rev.*, 40(1), 2010.

[5] C. Cranor, T. Johnson, O. Spataschek, and V. Shkapenyuk. Gigascope: A stream database for network applications. In *Proc. ACM SIGMOD*, pages 647–651, 2003.

[6] N. Duffield, A. Gerber, and M. Grossglauser. Trajectory engine: a backend for trajectory sampling. In *Network Operations and Management Symposium, 2002. NOMS 2002. 2002 IEEE/IFIP*, pages 437–450, 2002.

[7] N. Duffield and M. Grossglauser. Trajectory sampling with unreliable reporting. *Networking, IEEE/ACM Transactions on*, 16(1):37–50, 2008.

[8] N. G. Duffield and M. Grossglauser. Trajectory sampling for direct traffic observation. *IEEE/ACM Trans. Netw.*, 9(3):280–292, June 2001.

[9] F. McKeen. Innovative instructions and software model for isolated execution. In *Proc. HASP*, 2013.

[10] N. Feamster and H. Balakrishnan. Detecting bgp configuration faults with static analysis. In *Proc. 2nd Conference on Symposium on Networked Systems Design & Implementation (NSDI)*, pages 43–56, 2005.

[11] N. Feamster, J. Rexford, and E. Zegura. The road to sdn. *Queue*, 11(12):20:20–20:40, 2013.

[12] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker. Frenetic: A Network Programming Language. In *ACM ICFP*, 2011.

[13] N. Handigol, B. Heller, V. Jeyakumar, D. Mazières, and N. McKeown. I know what your packet did last hop: Using packet histories to troubleshoot networks. In *Proc. 11th USENIX Conference on Networked Systems Design and Implementation (NSDI)*, pages 71–85, 2014.

[14] J. Hizver. Taxonomic modeling of security threats in software defined networking. In *BlackHat Conference*, 2015.

[15] R. Huebsch, J. M. Hellerstein, N. Lanham, B. T. Loo, S. Shenker, and I. Stoica. Querying the internet with pier. In *Proc. VLDB*, pages 321–332, 2003.

[16] I. Anati et al. Innovative technology for cpu based attestation and sealing. In *Proc. HASP*, 2013.

[17] P. Kazemian, G. Varghese, and N. McKeown. Header space analysis: Static checking for networks. In *Proc. 9th USENIX NSDI*, 2012.

[18] S. Kent, C. Lynn, and K. Seo. Secure border gateway protocol (s-bgp). *IEEE J.Sel. A. Commun.*, 18(4):582–592, Sept. 2006.

[19] A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey. Veriflow: Verifying network-wide invariants in real time. In *Proc. 10th USENIX NSDI*, 2013.

[20] T. H.-J. Kim, C. Basescu, L. Jia, S. B. Lee, Y.-C. Hu, and A. Perrig. Lightweight source authentication and path validation. In *Proc. ACM SIGCOMM*, pages 271–282, 2014.

[21] D. Kreutz, F. M. Ramos, and P. Verissimo. Towards secure and dependable software-defined networks. In *Proc. ACM HotSDN*, 2013.

[22] D. Levin, Y. Lee, L. Valenta, Z. Li, V. Lai, C. Lumezanu, N. Spring, and B. Bhattacharjee. Alibi routing. In *Proc. ACM SIGCOMM*, pages 611–624, 2015.

[23] M. Hoekstra et al. Using innovative instructions to create trustworthy software solutions. In *Proc. HASP*, 2013.

[24] H. Mai, A. Khurshid, R. Agarwal, M. Caesar, P. B. Godfrey, and S. T. King. Debugging the data plane with anteater. In *Proc. ACM SIGCOMM*, pages 290–301, 2011.

[25] S. Matsumoto, S. Hitz, and A. Perrig. Fleet: Defending sdns from malicious administrators. In *Proc. ACM HotSDN*, pages 103–108. ACM, 2014.

[26] A. Mayer, A. Wool, and E. Ziskind. Fang: A firewall analysis engine. In *Proc. IEEE Symposium on Security and Privacy (SP)*, 2000.

[27] M. Mimoso. Critical flaws found in network management systems. *https://threatpost.com/critical-flaws-found-in-network-management-systems/115649/*, 2015.

[28] A. T. Mizrak, Y.-C. Cheng, K. Marzullo, and S. Savage. Detecting and isolating malicious routers. *IEEE Trans. Dependable Secur. Comput.*, 3(3):230–244, July 2006.

[29] J. Naous, M. Walfish, A. Nicolosi, D. Mazières, M. Miller, and A. Seehra. Verifying and enforcing network paths with icing. In *Proc. ACM CoNEXT*, pages 30:1–30:12, 2011.

[30] S. Narayana, J. Rexford, and D. Walker. Compiling path queries in software-defined networks. In *Proc. ACM HotSDN*, pages 181–186, 2014.

[31] P. v. Oorschot, T. Wan, and E. Kranakis. On interdomain routing security and pretty secure bgp (psbgp). *ACM Trans. Inf. Syst. Secur.*, 10(3), July 2007.

[32] Open Networking Foundation. Sdn security considerations in the data center. In *https://www.opennetworking.org/images/stories/downloads/sdnresources/solutionbriefs/*, 2013.

[33] P. Peresini, M. Kuzniar, and D. Kostic. Monocle: Dynamic, fine-grained data plane monitoring. In *Proc. 11th International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, 2015.

[34] R. Perlman. Network layer protocols with byzantine robustness. In *PhD thesis, MIT LCS TR-429*, 1988.

[35] J. Rasley, B. Stephens, C. Dixon, E. Rozner, W. Felter, K. Agarwal, J. Carter, and R. Fonseca. Planck: Millisecond-scale monitoring and control for commodity networks. In *Proc. ACM SIGCOMM*, pages 407–418, 2014.

[36] SANS Website. https://www.sans.org/reading-room/whitepapers/authentication/ssl-tls-whats-hood-34297. 2016.

[37] L. Schiff and S. Schmid. Study the past if you would define the future: Implementing secure multi-party sdn updates. In *Proc. IEEE SwSTE*, 2016.

[38] V. Sekar, M. K. Reiter, W. Willinger, H. Zhang, R. R. Kompella, and D. G. Andersen. Csamp: A system for network-wide flow monitoring. In *Proc. USENIX NSDI*, pages 233–246, 2008.

[39] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. Mckeown, and G. Parulkar. Can the Production Network Be the Testbed? In *OSDI*, 2010.

[40] A. Shieh, E. G. Sirer, and F. B. Schneider. Netquery: A knowledge plane for reasoning about network properties. In *Proc. ACM SIGCOMM*, pages 278–289, 2011.

[41] S. Shirali-Shahreza and Y. Ganjali. Flexam: Flexible sampling extension for monitoring and security applications in openflow. In *Proc. ACM HotSDN*, pages 167–168, 2013.

[42] G. G. Xie, J. Zhan, D. A. Maltz, H. Zhang, A. Greenberg, G. Hjalmtysson, and J. Rexford. On static reachability analysis of ip networks. In *in Proc. IEEE INFOCOM*, 2005.

[43] Y. Yu, C. Qian, and X. Li. Distributed and collaborative traffic monitoring in software defined networks. In *Proc. ACM HotSDN*, pages 85–90, 2014.

[44] L. Yuan, J. Mai, Z. Su, H. Chen, C.-N. Chuah, and P. Mohapatra. Fireman: A toolkit for firewall modeling and analysis. In *Proc. IEEE Symposium on Security and Privacy (SP)*, pages 199–213, 2006.

[45] X. Zhang, H.-C. Hsiao, G. Hasker, H. Chan, A. Perrig, and D. G. Andersen. Scion: Scalability, control, and isolation on next-generation networks. In *Proc. IEEE Symposium on Security and Privacy (SP)*, pages 212–227, 2011.

[46] Y. Zhang, M. Roughan, N. Duffield, and A. Greenberg. Fast accurate computation of large-scale ip traffic matrices from link loads. In *Proc. ACM SIGMETRICS*, pages 206–217, 2003.