Let SDN Be Your Eyes: Secure Forensics in Data Center Networks

Adam Bates Kevin Butler
University of Oregon
{amb,butler}@cs.uoregon.edu

Andreas Haeberlen
University of Pennsylvania
ahae@cis.upenn.edu

Micah Sherr Wenchao Zhou
Georgetown University
{msherr,wzhou}@cs.georgetown.edu

***Abstract*—Discovering the causes of incorrect behavior in large networks is often difficult. This difficulty is compounded when some machines in the network are compromised, since these compromised machines may use deception or tamper with data to frustrate forensic analysis. Recently proposed forensic tools enable administrators to learn the causes of some system states in a partially compromised network, but these tools are inherently unable to (1) observe covert communication between compromised nodes or (2) detect attempts to exfiltrate sensitive data.**

In this paper, we observe that the emergence of Software-Defined Networking (SDN) offers interesting new opportunities for network forensics. We sketch the design of an SDN-based forensic system that can be used to investigate a wide variety of faults in data center networks, including previously unobservable attacks such as data exfiltration and collusion between compromised nodes. Our key insight is that the network itself can be used as a point of observation, which gives us a holistic view of network activity. We show that a collection of lightweight middleboxes would be sufficient to support this functionality, and we discuss several additional challenges and opportunities for SDN-based forensic tools.

I. Introduction

Investigating a possible security breach is often a tedious, manual task. When the administrator observes a suspicious event – perhaps an unusual log entry, or suspicious connection attempts between unrelated machines – she must decide whether there is an actual attack, or whether there is a benign explanation. In a complex network, finding such an explanation can be highly nontrivial even when the real cause is benign (say, a malfunction or a misconfiguration); if the cause is an actual intrusion, the attacker will probably try to cover his tracks, or even make it appear as if the cause is on a different machine, thus sending the administrator on a wild goose chase.

To aid system administrators, a variety of forensic tools have been proposed to monitor and debug networks. Recently, there has been work that formulates network accountability and forensic analysis as a problem of *data provenance* [3]. Conceptually, provenance is a detailed history of events that allows a user to understand how the present or past state of an object was derived. Within the context of the data center, *network provenance* can be used to trace back traffic and discover the cause of an event [28]. For example, an administrator can use a network provenance system to discover if a suspicious routing table entry is due to a simple misconfiguration, the result of a routing attack, or is actually benign.

Network provenance systems have been applied in distributed systems to detect faults and attacks while incurring only modest overheads. These systems often rely on *correct* (i.e., uncompromised, non-faulty) nodes to observe and record the actions of other nodes for possible forensic uses. A limitation of this approach is that, if an attacker carefully avoids interacting with nodes that are not yet compromised, the attack may remain invisible to the forensic system. This is particularly problematic in data center networks that store sensitive data: if these data are exfiltrated to a remote adversary, the provenance system cannot identify the node that leaked the data unless the leakage was observed by at least one correct node.

This paper is inspired by the observation that, through the advent of software-defined networking (SDN), we can now use *the network itself* as the observation point for a forensic system. In this paper, we examine the issues related to incorporating SDN into a network diagnostic tool. We show that, by letting SDN be our eyes, system administrators are able to ascertain the correctness of every machine when issuing a forensic query, making it possible to detect the presence of previously unobservable attacks. Our security properties are achieved by leveraging the *complete observability* of network events that is possible in data center networks with SDN. Perhaps surprisingly, we show that such powerful guarantees can be attained without introducing significantly greater overhead than existing forensic systems. Moreover, we argue that such SDN-based forensic systems can operate effectively in the presence of compromised network components.

II. Overview

This paper considers the scenario that is illustrated in Figure 1. An administrator (Alice) is running a data center and has observed some kind of unexpected behavior within the network (e.g., multiple connections to an unknown remote address, high network traffic on a switch, etc.). Now, she must investigate whether the behavior is legitimate or is indicative of a fault or attack. Unfortunately, it is not possible for Alice to simply ask the nodes about their activities. An attacker may have gained complete control of a node and could return false information in response to Alice's request. Especially in the case of a clandestine attack, failure to detect the cause of the observed anomaly could have severe consequences. The attacker could

SENT '14, 23 February 2014, San Diego, CA, USA
Copyright 2014 Internet Society, ISBN 1-891562-36-3
http://dx.doi.org/10.14722/sent.2014.23002

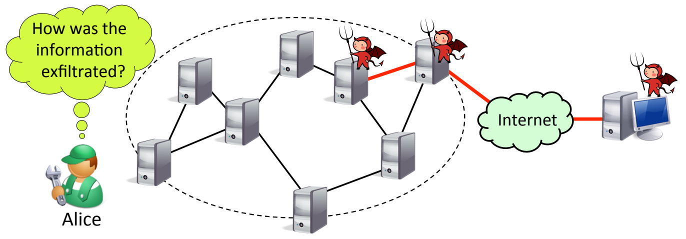


Fig. 1: Motivating example for network forensics. Alice runs a data center and suspects that sensitive data has been leaked to the Internet by compromised nodes.

attempt to exfiltrate sensitive or classified information from the node, as was the case with Operation Aurora [15].

A. Prior Work

To obtain reliable information about the network, Alice can turn to a forensic system that answers queries about past and present network state (e.g. *"Why was this node sending so much traffic at time t?"*). Fearful of an active attacker, Alice cannot rely on a tool designed for non-adversarial settings (cf. [27], [30]) since adversaries may subvert the forensic investigation by contributing false information. Alice might also be wary of introducing trusted host components such as a virtual machine monitor [1], [13], host-level monitor [19], or operating system [20], as OSes and VMMs might be susceptible to exploitable bugs and require significant management overhead. Alice may consider turning to a trusted hardware solution [4], but these require specialized devices and can also be subject to compromise [12].

Secure Network Provenance (SNP) [29] has been proposed as an alternate forensic system that can operate in untrusted environments. SNP constructs a *provenance graph* based on the execution of a network system, where each vertex in the graph represents a system state or event, and an edge represents a direct causal dependency between the corresponding system states or events. The constructed provenance graph captures transitive causes/effects of any system state or events. SNP then answers forensic queries by extracting a recursive explanation from the provenance graph, effectively *replaying* network events for Alice to review.

SNP instruments each node in the network to manage its own tamper-evident log [8], recording the messages they send to other nodes, as well as those they receive. When node A sends a message to node B, A commits to the message with a cryptographic signature using a private key known only to A. In this manner, nodes are unable to forge messages about other nodes' activity. Later, when Alice issues a forensic query, SNP performs on-demand retrieval of the records from the tamper-evident logs. A consistency check is performed between node responses to detect omissions or equivocations. Nodes that have lied or omitted information are marked as faulty. SNP provides the provable guarantee that an *observable* symptom of a fault or an attack can be traced to a specific event on at least one faulty node. Imposing less than 4% additional runtime CPU load and requiring just 178 bytes per message authenticator, the cost of provenance maintenance with SNP is low enough to be practical [29].

B. Challenges

While SNP provides a strong guarantee even in a system that is under attack, it makes concessions that limit its usability in scenarios such as the one described in Figure 1. Since SNP detects omissions and equivocations by checking inconsistencies between node logs, multiple faulty nodes might coordinate their lies in order to avoid detection. As a consequence, SNP provides answers only to queries about behavior that is *observable by at least one correct node*; that is, SNP can answer questions about network activity when one or more of the communicating nodes are uncompromised and fully functioning. As the number of correct nodes in the network decreases, so too does the observable network state, reducing the network area the administrator can "see" when she issues a query.

Outside of the observable network, compromised nodes can exchange covert messages with impunity or even engage in clandestine activity such as data exfiltration, without fear of being detected. In addition, in the presence of faulty nodes, SNP might return an incomplete result. Though eventual detection is guaranteed, faulty nodes may coordinate to minimize the "loss" caused by a detection. For instance, a botnet master may sacrifice one compromised bot to free itself from detection.

These limitations arise due to the reliance on end hosts for information about network activity. While nodes are not inherently trusted, a critical mass of correct nodes are required, or else system performance begins to degrade. Even when correct nodes are available, they are often poorly positioned to observe the network interior. Even collectively, they cannot achieve *complete observability* of the network, which is the optimal goal of a forensic system.

C. SDN as a Global Observer

We envision that, with the advent of SDN, the *network itself* can now be used as a point of observation in forensic systems. Rather than rely on reports from end nodes, we can transform every network link into a reporting tool by programming a distributed set of SDN switches and middleboxes. Accomplishing such a feat in a traditionally network system would have been costly and extraordinarily complicated, requiring a proxy box between every node in the network. Based on a set of programmable flow table rules, SDN allows us to perform complex sets of operations on a packet as it enters a switch. By pattern matching in the packet headers, these operations permit dropping unauthorized communications, modifying headers, and forwarding packets to potentially multiple other points in the network. When an operation is not possible within the switch itself (e.g., deep packet inspection), packets can be forwarded to middleboxes or the network controller for further processing [2], [10]. As we show in the rest of the paper, by leveraging these SDN functionalities, a set of lightweight middleboxes is sufficient to enable a holistic view of network activities even in untrusted environments.

III. Design Considerations

In this section, we develop the design of an SDN-based forensic system for data center networks.

Road map. In Section III-A, we discuss the core functionality that we need from software-defined networking. Noting that the capabilities within an OpenFlow switch are insufficient to accomplish our goals, we introduce *Provenance Verification Points*, middlebox components for forensic packet processing. Section III-B considers where to place these devices in the network: either as *in-line* devices that actively interpose on traffic, or as passive monitors of network activity. Acknowledging that efficiency will be critical to our architecture's practicality, we develop a verification protocol in Section III-C that minimizes the computational requirements of our forensic components. In Section III-D, we discuss the security properties of our design. Finally, Section III-E considers the implications of including our SDN components within the network's trusted computing base.

Threat model. This work considers the task of forensic analysis in a data center that is comprised entirely of SDN/OpenFlow switches. Many recent reports have indicated growing interest in the deployment of SDN in real-world data centers [6], [11].

At any time, one to many of the data center's nodes may suffer a benign failure or be compromised by a malicious attacker. Nodes may not only crash, but also silently change behavior and continue to operate. It then becomes the administrator's task to determine the cause of the problem, bearing in mind that the affected nodes may take countermeasures to evade detection. Therefore, we conservatively assume that Byzantine faults [16] will occur – adversarial nodes do not need to obey our protocols and may behave arbitrarily – and we say that the affected nodes are *faulty*, regardless of whether they are compromised or simply malfunctioning. (We also say that a uncompromised, non-faulty nodes are *correct*.) In particular, faulty nodes may attempt to hide from the administrator and they may collude, although any messaging that occurs between them must flow through SDN switches (as with all other data center traffic). We assume the standard computational bounds—our adversary cannot invert cryptographic hashes or forge digital signatures.

We assume that all inter-node communication in the data center takes place through the monitored physical network; out-of-band communication via wireless technologies or sneaker nets is not possible. Similarly, communication entering or leaving the data center must pass through SDN switches. We leave the matter of covert channels (e.g. timing) to future work, although we discuss a possible detection strategy in Section IV. For simplicity, we assume that the network is lossless, although we also discuss the implications of packet loss in Section IV.

Finally, we make the following assumptions about security of our network infrastructure. We assume that our switches and network controller cannot be compromised by an adversary. Mechanisms for SDN security is an emerging field that is out of scope to this work [14], [22], [23], [26]. Finally, we introduce a forensic middlebox in Section III-A; *we do not assume that these middleboxes are secure*. We address the matter of middlebox failure in Section III-E.

A. OpenFlow Requirements

While SDN allows us to extend network functionality, there are limits to the operations that can be performed on an OpenFlow switch. These actions include packet forwarding, dropping, flooding, and header modification [21]. Action sets are associated with flow table rules that are enabled through performing pattern matching over the packet header fields. OpenFlow switches also maintain minimal state per flow. More complex features (e.g., inspecting packet payloads) can be achieved by forwarding packets to the network controller or to some other middlebox; traffic replication for enhanced network features is an application of SDN that is available in enterprise products [2] and has been explored in the literature [7], [10]. Our architecture relies on two SDN primitives: forwarding traffic to middleboxes, and dropping traffic that does not adhere to a specified behavior.

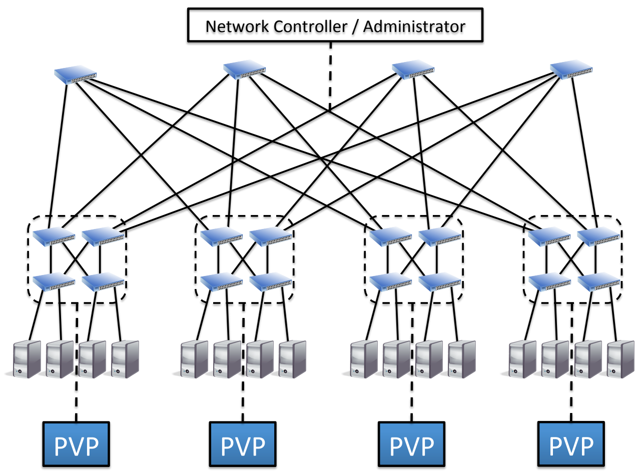


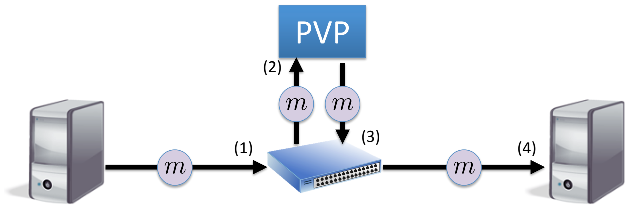
Fig. 2: *Provenance Verification Points* (PVPs) are SDN middleboxes that collect forensic information. Dotted lines represent logical groupings and links, and solid lines represent physical links.

The SDN functionality available on a switch is not sufficient to build a forensics system, so our design must feature a middlebox component. We call these middleboxes *Provenance Verification Points*, or *PVPs*. Each PVP is responsible for monitoring activity for some subset of system nodes; determining the exact number of middleboxes required for deployment is a task that is explored elsewhere in the literature [5], [7], [9], [24], [25]. As a demonstrative example, Figure 2 shows a FAT tree topography in which one PVP is allocated to monitor traffic in each pod [17].

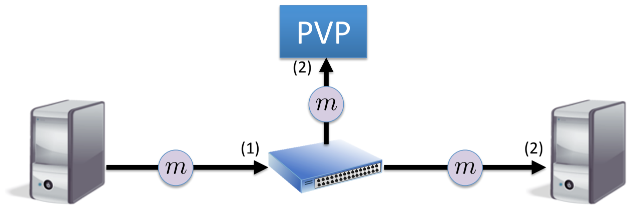
For each permissible flow type in our data center, we install OpenFlow forwarding rules that direct traffic through a PVP. These rules act as hooks into our forensic system. In short, our SDN policies ensure that all traffic is either blocked or monitored. The adversary is able to communicate across the network through the permitted flows; however, the administrator will later be able to uncover the adversary's actions by issuing forensic queries. In the remainder of this section, we explore how to best integrate the PVP component into our forensic tool.

B. Middlebox Routing

We first consider how to route network traffic to the PVPs. The two basic forms of middlebox routing are visualized in



(a) Traffic Interposition



(b) Traffic Mirroring

Fig. 3: Two methods of PVP interaction. The time of each message transmission is marked in parentheses.

Figure 3: *traffic interposition* (3a) and *traffic mirroring* (3b). For our purposes, traffic interposition would allow PVPs to actively enforce message commitment protocols, dropping messages that were not digitally signed by the senders. In contrast, traffic mirroring does not allow instantaneous policy enforcement, but minimizes packet latency.

Due to the additional latency imposed by traffic interposition, we opt for the passive observation model of traffic mirroring. A negative consequence of our choice is that our forensic system cannot actively detect (or prevent) exfiltration attempts since PVPs may receive copies of outgoing packets after the corresponding copies have exited the data center. However, our design still significantly aids in the *forensic process* since the PVPs capture a copy of all traffic that occurred during the exfiltration. Hence, PVPs may still provide verification over network events (explained in the next section) and investigators can identify the source of the exfiltration.

More generally, when a PVP detects a message that has not been properly signed, it can alert the system administrator and request that the network controller isolate the offending node.[1] Insertion of flow table rules has been shown to add processing delays on the order of tens of milliseconds [23]; taking into account processing delay at the PVP, we estimate that the faulty node could be sandboxed within hundreds of milliseconds of its first unauthorized transmission.

C. Lightweight Monitoring

While SDN makes it easy to mirror packet transmissions to arbitrary points in the network, it is much more difficult to process those packets in a manner that allows the forensic system to keep up with network traffic. If the PVPs are slower than the network's real time communications, their buffers would become exhausted, leading to message loss, which in turn would break the *complete observation* property that we are attempting to achieve in our forensic system. We thus identify the minimum functionality necessary for PVPs to provide while still facilitating complete and correct forensic querying. This means that PVPs must possess all of the required information to help Alice explore complex series of states, such as inspecting the causal chain that led to a packet's transmission.

To accomplish this, we propose that the PVPs operate as a verification layer in an inter-node commitment protocol, much like the accountability system described in PeerReview [8], [29]. PeerReview allows correct nodes to defend themselves against false accusations, but it cannot attest to the actions of multiple incorrect nodes that communicate. Extending the PeerReview protocol with PVPs makes these attestations possible. Each cross-node communication, even between two faulty nodes, is monitored by the corresponding PVP, and an evidence for the communication is retained. Such capability allows PVPs to achieve full observability and catch a wider class of misbehavior than SNP.

Informally, the PVP participates in the commitment as follows: Node A wishes to send a message m to node B. To do so, A first records an entry for m in a local append-only tamper-evident log. As A appends to the log, a hash chain of its actions is incrementally built. Next, A sends m to B along with a signed copy of the new hash, as well as a short hash chain segment that connects this hash to the previous signed hash. Within the network, this message is mirrored and arrives at both Node B and a PVP. Both B and the PVP verify A's signature and the hash chain. The PVP stores the signed authenticator, and then discards the message. Meanwhile, Node B continues the protocol by sending an acknowledgement (ACK) to A, following the same procedure as A did in its original message. The network also mirrors the ACK such that it arrives at Node A and the PVP. Both A and the PVP verify B's signature and hash chain, and the PVP retains this signed authenticator. The PVP now possesses proof of both message m and its acknowledged receipt.

Later, Alice suspects a potential attack on her network around the time m was sent. She queries the nodes in the network about m, its derivation, and its consequences. That is, she asks *"Why did m exist at time t, and what other events were a result of m?"*. Consider the case in which A and B are both faulty and wish to hide m's presence. In previous accountability systems, they would have been able to deny knowledge of message m, as no correct system component observed m's existence. However, the PVP now possesses proof of m's transmission by A and receipt by B. Alice can demand a transcript of network activity from the relevant parts of A's and B's respective local logs, then compare the transcripts to the PVP's list of authenticators. She will be able to detect the absence of the message and its acknowledgment, and declare both A and B to be faulty.

[1]We separate PVPs and network controllers in our design, but the PVP component could also run atop the controller.

The above protocol can be optimized such that the PVP can maintain all state within memory, avoiding the need to write to disk. Periodically, each node can be required to report on the contents of their local log. The PVP can use this report to confirm that its own records are complete, then chain the hashes together and discard the individual authenticators. The end of the hash chain serves as a proof of a node's activity over the entire timespan. A node that fails to report on its activity within a required time interval is flagged as faulty.

D. Security Properties

We have sketched the design for a distributed set of SDN components. The message commitment system employed represents the previous state of the art in forensic analysis [29]. We now summarize the additional guarantees that are provided by our SDN-based tool:

Detect covert communication. In traditional network architectures, capturing messages exchanged between compromised nodes is difficult: if no honest node is on the path between the two compromised nodes, then the communication will go unnoticed. Our global observer model (see Section II-C) eliminates unmonitored communication paths for explicit messages by instantiating SDN policies that forward a copy of all communication to a PVP. When PVPs are queried during a forensic investigation, the network operator will discover the attempted covert communication.

Detect equivocation. In previous systems, nodes were able to equivocate about their actions when responding to multiple queries; a faulty node could make inconsistent claims about the local inputs through which a particular message was derived. For example, in a BGP application, a faulty node could send conflicting route updates to two of its neighbors. In the past, both of the neighbors would need to be correct at query time in order to detect this lie. Using PVPs, though, we already have a full snapshot of network events. As a result, the inconsistent claims of the faulty node can be detected.

Response availability. As specified, our system cannot guarantee that forensic records will still exist at the time that a query is issued. This is because message content is recorded locally by the nodes, rather than globally at the PVPs. As a result, when a node becomes faulty through compromise or drive failure, we must assume its forensic records will be lost. However, by maintaining the authenticators associated with each message, PVPs possess a proof of each node's correspondence. Because the PVPs possess the authenticators associated with the lost forensics, the absence of the records is detectable by the administrator. As a result, the administrator will know that the node is faulty, even though she will be unable to observe some of its actions.

One of the primary objectives of a forensic system is to identify nodes that are incorrect due to compromise, hardware failure, etc.; through storing message authenticators, the PVPs offer the ability to detect all incorrect nodes in the network at any time. This is already a powerful guarantee. However, in environments where all network activity must be guaranteed to be *replayable* at a later date, we note that the PVP architecture would also be able to facilitate this ability. To do so, the PVPs would need to keep a complete log of all messages, rather than solely storing the authenticators. This additional forensic power would require significantly more computational provisioning for the PVP architecture.

E. Are PVPs Trustworthy?

Up to this point, we have treated the PVPs as a trusted component. We consider the PVP to be a highly specialized middlebox that is only responsible for its role in the verification protocol. The PVP must receive packets, verify their signatures, perform a hashing operation, and then respond to queries from the network operator. We estimate that the PVP could be implemented in a sufficiently small codebase to be a candidate for formal verification. It may even be possible to implement the PVP in hardware as an ASIC device [4].

However, it is not strictly necessary to trust the PVPs. Since nodes commit their messages to one another using cryptographic signatures, nodes have the required evidence to defend themselves if the PVP presents a false history of events. The PVP does not possess the private keys of the nodes, and so cannot forge messages.

More generally, if a PVP is faulty, then the global observation property no longer holds. Importantly, the portion of the network that is monitored by correct PVPs maintains the security properties described in the previous section (since all messages in that segment of the network are observed by honest observers). Notably, in the presence of faulty PVPs, the security guarantees of our system gracefully degrade to those provided by SNP [29].

Additionally, a dishonest PVP can falsely claim that a node has sent an unauthenticated message. Because a node cannot present evidence that it did not send a particular message, this creates the potential for deniability and inconsistency between messages from the accused node and the dishonest PVP. In this case, it will be necessary for the system administrator to intervene and resolve the conflict and determine whether the node or the PVP is faulty (or both).

IV. Discussion

Building the Provenance Verification Points presents a variety of technical challenges and opportunities. Having described a prototype PVP design that allows a holistic view of the network, we identify some additional challenges that we hope to address in future work:

Message loss. Message loss sometimes is inevitable in networks due to traffic bursts, policy errors, or malicious attacks. However, we believe that some features of SDN can be used by the PVPs to detect and recover from message loss. We consider the following two scenarios:

If message loss occurs between a sender node A and the corresponding receiver node B (i.e., B did not receive the message sent by A), then B will not send an ACK and A can retransmit the message after a timeout expires. The PVP will observe the retransmission and (eventually) B's acknowledgment of the message.

If message loss occurs between either A or B and the PVP, then the PVP will be able to detect the drop by polling the switch for flow statistics [21]; the switch's *bytes sent* will not match the PVP's *bytes received*.

Timing side channels. In networked systems and even individual hosts, covert channels are myriad, and can be difficult to detect and remove. While our system captures all explicit message exchanges, the monitored network may be vulnerable to a *timing* side channel attack in which faulty notes can send implicit messages to one another through measurement delays between innocuous explicit messages. We remark that, by recording timing information in our message commitment protocol [29], the administrator will have the necessary information to later test for the presence of the timing channel. Timing based channels have been shown to be difficult to hide from a suspicious observer [18], and so the administrator may catch the faulty nodes through replaying the portion of the log in which the covert message was embedded.

Forensic automation. A common obstacle to forensic system deployment is the requirement for application instrumentation. For example, SNP required the insertion of several hundreds of lines of code in the Hadoop MapReduce implementation [29]. We envision that SDN can also be used to overcome this obstacle by extending the concept of *external specifications* of application behavior. External specifications were proposed in SNP as a method of overcoming closed-source applications; the specification was placed in a proxy box, which would use the high-level description of application behavior in order to extract network provenance. We believe that external specifications could also be expressed as a set of flow table rules that handle the different forms of application traffic. We intend to write a utility that parses a specification and outputs OpenFlow rules, thus automating the process of programming the network. These rules would be responsible for directing network traffic to additional forensic middleboxes, which would remove the need for instrumentation within the nodes.

Performance considerations. Our system is based on a message commitment architecture that has been shown to impose acceptable overheads [29]. We anticipate the discovery of further interesting design tradeoffs in the deployment of our architecture. For example, network overhead will decrease as the placement of PVPs becomes more localized within the network; however, this will require provisioning additional PVPs. Additionally, PVPs may periodically fail. Compensating for PVP failure requires the introduction of PVP redundancy and an efficient form of recovery. We intend to better characterize the tradeoffs between reliability and performance in future work.

V. Conclusion

This paper explores software-defined networking's ability to aid in the forensic analysis of distributed systems. While SDN grants novel capabilities, we have shown that the design of SDN-based forensic components must be carefully considered prior to deployment in data center environments. Through introducing the *Provenance Verification Point* component, we have demonstrated the ability to detect the presence of attacks that were previously unobservable by forensic systems. In doing so, we have shown that SDN is an important missing piece in existing forensic systems that allows system administrators to achieve total observability of network activity.

Acknowledgments

We thank the anonymous reviewers for their insightful comments and suggestions. This work is partially supported by the National Science Foundation through grants CNS-1054229, CNS-1064986, CNS-1065130, CNS-1118046, CNS-1149832, CNS-1204347, and CNS-1254198, and by DARPA contract FA8650-11-C-7189.

References

[1] M. Basrai and P. M. Chen. Cooperative Revirt: Adapting Message Logging for Intrusion Analysis. Technical Report CSE-TR-504-04, University of Michigan, 2004.

[2] Big Switch Networks, Inc. Big Tap: Monitor Traffic Everywhere, Deliver Traffic Anywhere, 2012. Available at http://www.bigswitch.com/products/big-tap-network-monitoring.

[3] P. Buneman, S. Khanna, and T. Wang-Chiew. Why and Where: A Characterization of Data Provenance. In *International Conference on Database Theory (ICDT)*, Jan 2001.

[4] B.-G. Chun, P. Maniatis, S. Shenker, and J. Kubiatowicz. Attested Append-Only Memory: Making Adversaries Stick to Their Word. In *Proceedings of 21st ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*, Oct 2007.

[5] C. Dixon, H. Uppal, V. Brajkovic, D. Brandon, T. Anderson, and A. Krishnamurthy. ETTM: A Scalable Fault Tolerant Network Manager. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation (NSDI)*, Mar 2011.

[6] S. Elby. Software Defined Networks: A Carrier Perspective. In *Open Networking Summit*, Oct 2011.

[7] A. Gember, P. Prabhu, Z. Ghadiyali, and A. Akella. Toward Software-Defined Middlebox Networking. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks (HotNets)*, Aug 2012.

[8] A. Haeberlen, P. Kouznetsov, and P. Druschel. PeerReview: Practical Accountability for Distributed Systems. In *ACM Symposium on Operating Systems Principles (SOSP)*, Oct 2007.

[9] B. Heller, R. Sherwood, and N. McKeown. The Controller Placement Problem. In *Proceedings of the 1st Workshop on Hot topics in Software Defined Networks (HotSDN)*, Aug 2012.

[10] V. Heorhiadi, M. K. Reiter, and V. Sekar. New Opportunities for Load Balancing in Network-wide Intrusion Detection Systems. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, 2012.

[11] U. Hoelzle. OpenFlow at Google. Available at http://youtu.be/VLHJUfgxEO4.

[12] B. Kauer. OSLO: Improving the Security of Trusted Computing. In *USENIX Security Symposium*, Aug 2007.

[13] S. T. King and P. M. Chen. Backtracking intrusions. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, Oct 2003.

[14] D. Kreutz, F. M. Ramos, and P. Verissimo. Towards Secure and Dependable Software-Defined Networks. In *Workshop on Hot Topics in Software Defined Networks (HotSDN)*, August 2013.

[15] G. Kurtz. Operation Aurora Hit Google, Others, Jan 2010. Available at http://securityinnovator.com/index.php?articleID=42948&sectionID=25.

[16] L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[17] C. E. Leiserson. Fat-trees: Universal Networks for Hardware-efficient Supercomputing. *IEEE Trans. Comput.*, 34(10):892–901, Oct. 1985.

[18] X. Luo, P. Zhou, J. Zhang, R. Perdisci, W. Lee, and R. K. C. Chang. Exposing Invisible Timing-Based Traffic Watermarks with BACKLIT. In *Proceedings of the 27th Annual Computer Security Applications Conference (ACSAC)*, Dec 2011.

[19] P. Mcdaniel, K. Butler, S. Mclaughlin, R. Sion, E. Zadok, and M. Winslett. Towards a Secure and Efficient System for End-to-End Provenance. In *USENIX Workshop on Theory and Practice of Provenance (TaPP)*, Feb 2010.

[20] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. Seltzer. Provenance-aware Storage Systems. In *USENIX Annual Technical Conference (ATC)*, May 2006.

[21] OpenFlow Switch Consortium. OpenFlow Switch Specification Version 1.1.0, February 2011. Available at http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf.

[22] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu. A Security Enforcement Kernel for OpenFlow Networks. In *Workshop on Hot Topics in Software Defined Networks (HotSDN)*, August 2012.

[23] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore. OFLOPS: An Open Framework for Openflow Switch Evaluation. In *International Conference on Passive and Active Measurement (PAM)*, Mar 2012.

[24] S. Schmid and J. Suomela. Exploiting Locality in Distributed SDN Control. In *Proceedings of the 2nd Workshop on Hot Topics in Software Defined Networks (HotSDN)*, Aug 2013.

[25] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi. Design and Implementation of a Consolidated Middlebox Architecture. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation (NSDI)*, Apr 2012.

[26] S. Shin and G. Gu. Attacking Software-Defined Networks: A First Feasibility Study (short paper). In *Workshop on Hot Topics in Software Defined Networking (HotSDN)*, Aug 2013.

[27] A. Singh, P. Maniatis, T. Roscoe, and P. Druschel. Using Queries for Distributed Monitoring and Forensics. In *European Conference on Computer Systems (EuroSys)*, Apr 2006.

[28] W. Zhou, E. Cronin, and B. T. Loo. Provenance-aware Secure Networks. In *International Conference on Data Engineering Workshop*, 2008.

[29] W. Zhou, Q. Fei, A. Narayan, A. Haeberlen, B. T. Loo, and M. Sherr. Secure Network Provenance. In *ACM Symposium on Operating Systems Principles (SOSP)*, Oct 2011.

[30] W. Zhou, M. Sherr, T. Tao, X. Li, B. T. Loo, and Y. Mao. Efficient Querying and Maintenance of Network Provenance at Internet-scale. In *ACM International Conference on Management of Data (SIGMOD)*, June 2010.