Anomaly Traceback using Software Defined Networking

Jérôme François, Olivier Festor
INRIA Nancy Grand Est, France, firstname.lastname@inria.fr

***Abstract*—While the threats in Internet are still increasing and evolving (like intra multi-tenant data center attacks), protection and detection mechanisms are not fully accurate. Therefore, forensics is vital for recovering from an attack but also to identify the responsible entities. Therefore, this paper focuses on tracing back to the sources of an anomaly in the network. In this paper, we propose a method leveraging the Software Defined Networking (SDN) paradigm to passively identify switches composing the network path of an anomaly. As SDN technologies tend to be deployed in the next generation of networks including in data centers, they provide a helpful framework to implement our proposal without developing dedicated routers like usual IP traceback techniques. We evaluated our scheme with different network topologies (Internet and data centers) by considering distributed attacks with numerous hosts.**

I. Introduction

Nowadays, Internet is an amazing playground for the attackers because there are many users and devices which can be targeted. Therefore, designing and developing new protection mechanisms, like Intrusion Detection System [1], is of paramount importance. While these techniques are helpful for identifying when an attack occurs, they do not aim at identifying root causes and origins (responsible entities) of the attacks which are an objective of forensics. A naive approach would consider source IP addresses as the origins of an attack but an attacker can easily impersonate it (IP address spoofing) [2]. Such spoofing techniques are especially useful for (D)DoS (Distributed Denial of Serivce) attacks, which are still a major threat in Internet [3]. In particular, an attacker can send data directly by forging his own IP address or by impersonating the IP address of his target when contacting multiple servers (DNS, web, etc.) usually from multiple zombie machines of a botnet. This causes flooding replies from these servers to the victim (reflector attack) [4].

In this paper, we propose a new technique to trace back to the source of an anomaly[1]. Our method takes advantage of Software Defined Networking (SDN). SDN is a recent networking paradigm which relies on centralizing forwarding decisions in a controller. It gained a lot of interests and is currently mainly implemented through OpenFlow [5] which is the *de facto* standard. It supports incremental deployment and is useful for various network management tasks like monitoring [6]. However, using SDN for forensic purposes has been very few investigated. Based on OpenFlow, our method is able to find all flows which are related to an attack specified in a general way without necessarily mentioning IP addresses. For example, assuming a web server having been attacked, our objective is to find all locations, and in fact all paths, which may have connected to this web server on port 80. Hence, it is helpful for forensic analysis without assuming any source IP addresses (logged by the server) which are subject to spoofing.

The main contribution of this papers is a traceback approach relying on SDN, and so which does not require to deploy any specific tool in SDN enabled network. Our methods has been evaluated through various scenarios.

The paper is structured as follows. Section II mentions related work. Necessary background is described in Section III and our approach is detailed in Section IV. Section V focuses on the evaluation prior concluding remarks in Section VI.

II. Related work

As introduced previously, tracing back anomalies is strongly related to IP spoofing detection and DDoS attacks. Detecting DDoS attacks has been widely covered [7] especially by focusing on the victim (who is under attack?) without identifying real sources. For example, it is possible to detect incoherent Time-to-Live value of IP packets supposedly incoming from the same host [8]. Even if spoofed IP addresses are distinguished, it is not possible to retrieve the real addresses behind. IP traceback techniques (packet marking or logging) have been proposed to find the origins of an IP packet. Packet marking consists for a router to mark forwarded packets. Numerous works have been done to select the packets to mark and to define the marks in order to limit the overhead (Probabilistic packet Marking). In [9], the path of an attack is reconstructed from multiple marked packets with partial information. Song *et al.* focused on authenticating marked packets [10]. A checksum-based method is given in [11]. A hop-by-hop reconstruction is proposed in [12]. In a packet logging scheme [13], the routers have to log the traffic and are requested when an attack occurs. A major challenge is to limit the router overhead by sampling packets [14]. All these approaches have never been deployed at large scales as switches were not enough flexible to implement new functionalities for less. However, the programmability introduced by SDN enables it. Actually, SDN technologies like OpenFlow [5], which consider switches configured remotely by a controller, are highly supported by equipment vendors and operators. This eases its deployment in real networks, in particular in data centers[2]. For instance, Google's backbone networks are already OpenFlow-enabled [15]. Another incentive to deploy such technologies is that they help to realize various tasks (routing, load-balancing, quality of service, etc.) unlike previously proposed traceback

[1] An anomaly might be an attack, a misbehavior or a misconfiguration. In this paper, both terms, attacks and anomalies are interchangeable.

[2] China Telecom and Huawei deploy commercial SDN, http://www.datacenterdynamics.com/focus/archive/2014/04/china-telecom-and-huawei-deploy-commercial-sdn, accessed on 06/13/2014

methods. SDN is thus a natural choice for addressing our problem. In addition, it does not solely focus on the IP level (layer 3) and can so traceback to the origins of an attack at layer 2 (Ethernet). Regarding network monitoring, OpenFlow has been widely used to collect network flow statistics in a passive way [6] or to do active measurements about latency by injecting packets [16]. In [17], the authors leverage SDN in order to redirect traffic to middleboxes which are in charge of doing forensic analysis.

III. Background

A. Software Defined Networking

SDN promotes the programmability of the network by decoupling the data plane from the forwarding plane. The key idea is to let the forwarding devices (like routers or switches) only being affected to the forwarding task while the decisions about where to forward data are decentralized to a controller. Hence, routers are not anymore responsible for running routing algorithms to compute their forwarding tables. In fact, such a decoupling enables powerful functionalities by having a controller which, thanks to more resources and more knowledge, can take better decisions about forwarding. For instance, at layer 2, the spanning tree protocol [18] creates a spanning tree for a local network. However, it can be considered as suboptimal because some links are not used to avoid loops. Actually, two nodes can be connected by a physical link whereas the logical path is going through other intermediate nodes. With SDN, fine-grained routing decisions can be done. In addition, QoS (Quality-of-Service) is possible as well as isolation which is very helpful in a cloud context.

SDN might be envisioned as complementary to cloud resources management by enabling networking through virtual instances [19]. This illustrates the concept of network programmability which has been proposed in 90's but which, at that time, didn't get support from many industrials. Nowadays, this concept leads to a strong joint community of academia and industrial which has notably led to the OpenFlow [5].

B. OpenFlow

OpenFlow is a major representative to configure and monitor SDN capable devices. The switches are communicating through the OpenFlow protocol with a controller like Floodlight[3]. While the full protocol specification is given at https://www.opennetworking.org, this section describes the main components and functionalities, which are particularly in the scope of the addressed problem (we rely on v.1.3).

The core functionality is to install forwarding rules on switches. Each forwarding rule is defined by different parameters where the most important are the *match fields* and the *instructions*. The *match fields* represent a filter to apply on packet headers in order to determine if a packet belongs to the flow or not. For instance, the *match fields* can specify what is the source or destination IP address or subnetwork, source or destination ports, Ethernet source or destination addresses, physical port on the switch where the packet arrives, etc.

The instructions is a set of *actions*. The main one is *output* to send the packet to a specific switch port. Packets can be also dropped or pushed into queues which are handled by specific schedulers for performing QoS. Other possibles actions are modifications to the packets headers like Ethernet addresses or the TTL (time-to-live). Moreover, each flow table entry can be associated with various counters for monitoring purposes like number of bytes, number of packets, duration, etc.

Actually, while a controller can install rules pro-actively, the most common case is to install rules on demand. When, a packet is not matching any rule (*table-miss*), a usual default action is to send a *PacketIn* message to the controller for requesting what actions have to be taken regarding this packet. For example, the controller uses the *FlowMod message* to install a new flow table entry which matches this packet and all subsequent ones of the flow. Simplified examples of flow table entries are given in table I where the major used fields are represented. To ease the understanding, a textual description of the purpose of the rule is mentioned in the first column. The rules can also be removed, either on demand from the controller, or after a timeout occurs. This table also highlights that differentiating a switch (Ethernet layer) from a router (IP layer) depends on the installed rules and the same device can even act as both. Hence, we use the generic term of switch in this paper independently of the installed rule.

Hence, the processing pipeline of a packet is the following. First the switch looks for an entry where the *match fields* match the packet headers. In such a case, the actions are applied on the packet (modifications and/or forwarding). There may exist several flow tables and an action can be to pass the packet processing to another flow table. Finally, if there is no match, the switch requests the controller, which then install a new rule to match the incoming packet. The OpenFlow controller can also request a switch about its current state, like flow statistics, through the *read-state* message.

IV. Traceback method

A. Inputs and ouputs

The main input of our method is an anomaly specified as:

$$a =< t_{start}, t_{end}, location, fields >$$

where:

- t_{start}: the time when the anomaly starts,
- t_{end}: the time when the anomaly ends,
- $location$: the location of the attack, *i.e.* the end host where the anomaly is detected,
- $fields$: a set of matching fields in the sense of the OpenFlow flow tables.

The output is the list of origins (hosts or switches in the network) of this anomaly.

B. Graph based modeling

We model our network as a directed graph $G = (V, E)$ whit V vertices and E edges. In such a network, each vertex represents an intermediate node, *i.e.* a switch, and each edge represents a network link. Our goal is to detect what are the entry points (first switches) on the path of an anomalous traffic.

[3]http://www.projectfloodlight.org/,accessed on 2014/05/13

Textual description	Ingress port	Mac Src Addr	Mac Dst Addr	Ip Src Address	Ip Dst Address	Protocol	Src port	Dst port	Instructions
Switching	*	*	AB:CD:EF:00:11:22	*	*	*	*	*	Forward to port 3
Routing	*	*	*	*	1.2.3.*	*	*	*	Set Mac src addr=AB:CD:EF:00:11:33, Mac dst addr = AB:CD:EF:00:11:44, forward to port 5
Firewall (deny incoming web connections)	1	*	*	*	1.2.3.*	TCP	*	22	Drop
Proxy	*	*	*	*	2.3.4.5	TCP	*	80	Set IP addr=10.11.12.13, forward to port 5
Load balancing	1	*	*	*	2.3.4.5	TCP	*	80	set dst addr = 2.3.4.6, Forward to port 4
	2	*	*	*	2.3.4.5	TCP	*	80	set dst addr = 2.3.4.7, Forward to port 6

TABLE I: Examples of OpenFLow flow table entries (* represents a wildcard to match any value)

Each node is associated to a set of tables similar to the flow tables of OpenFlow: $[T_0, \ldots, T_M]$.

Each entry in such a table represents an OpenFlow rule. For sake of clarity, we limit the definition of the flow table as follows: $T_i = \{f_o, \ldots, f_N\}$ where a flow entry f_i is defined as a tuple $f_i =< fields_i, modifications_i, output_i >$:

- $fields_i$ represents the fields to match and so, for each packet p, $match(p, fields_i)$ returns true if the current packet headers match the fields.
- $modifications_i$ represents the modification that will be applied to the different header fields of a packet. Hence, assuming $modifications_i(p, p')$, p' is the results of applying modifications on p, where p and o' are packets.
- $output$ is the output of the packet after the rule has been executed. To consider most of usual cases that are usable with OpenFlow, it can represents either a physical port and so an edge from E, another flow table or a drop action.

OpenFlow specification allows more functionalities. For instance, the flow tables can be merged by groups but, while it can be handled in more complex data structures in our model, this will impact the understanding and the readability of our model. Hence, we prefer to focus on the core functionalities of OpenFlow. Moreover, queuing based QoS is also possible by forwarding packets to specific queues. In our model, this corresponds to have an intermediate step for the output of a flow entry. Since, we are targeting forensics purpose, our model is sufficient but needs some processing to compress rules which are successively applied to a single packet in order to have only one rule with associated matched fields, outputs and modifications.

We assume that each forwarding rule is expressed as OpenFlow rules even if some forwarding decisions are given by usual routing algorithms (NORMAL port). It is not a strong assumption as such decisions might be also defined using OpenFlow.

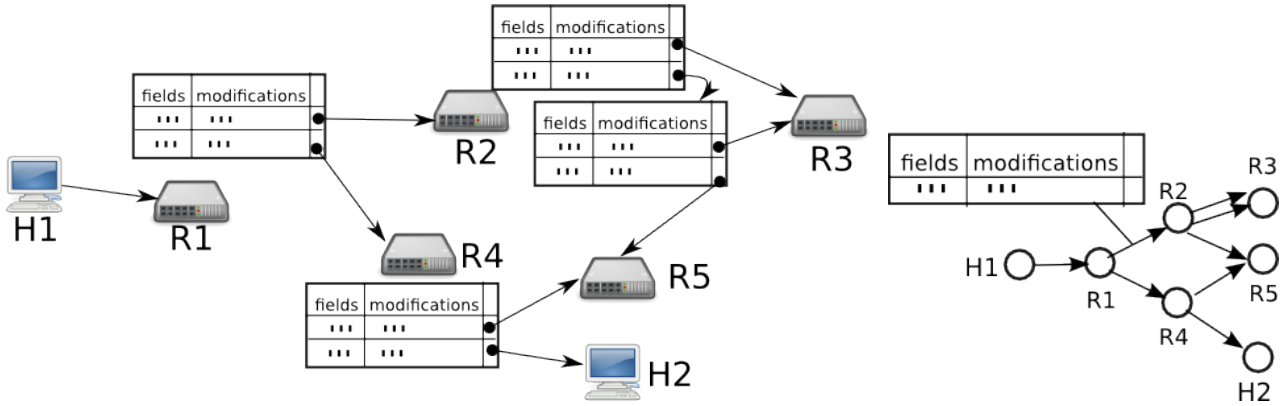
An example of our model is illustrated in Figure 1. Figure 1(a) represents a small topology with few hosts and switches associated with one or more flow tables. In fact, each output of a flow table entry is represented by an arrow either pointing to another switch (physical port forwarding) or to another flow table, which is the case for R2. If there is a drop action, this results in the lack of an arrow. In figure 1(b), the model is represented as a directed graph where there might have multiple arrows between two nodes, for instance between R2 and R3. This is due to multiple flow table entries with the same action but usually with different matching fields. However, for our analysis, information regarding the matching fields and modifications are necessary. Thus, such a graph requires to maintain an additional association between each arrow and an OpenFlow rule as highlighted between R1 and R2. From a terminology point of view, our graph contains attributed directed edges.

C. Model analysis

To find the origins of an anomaly, our graph-based model is analyzed through a depth-first search traversal. Assuming the anomaly $a =< t_{start}, t_{end}, location, fields >$, we will retrieve all flow tables entries which have been effective between t_{start} and t_{end}. This is feasible through periodic polling from the controller.

The main process is presented in algorithm 1 and based on the following functions:

- $ingoing\_edges(g, l)$ returns the ingoing directed edges of the node l.
- $rule(e)$ returns the associated rule to the edge e as explained in the previous section.
- $inv\_mod(r, f)$ applies the modifications of the rule r in an inverse way to the fields f. If no modification is defined, it is equivalent to the identity function. This function is necessary as the matching fields, *i.e.* the specific protocol headers, of an anomaly might have been modified along the network path. However, a one-to-one mapping cannot be expected since the rules are usually not specific to the exact anomaly. For example, a rule could just modify the Ethernet address for forwarding purposes while the anomaly is related to a specific TCP port. Also, it is possible to have an anomaly related to a source IP address while modifications are applied to an entire subnetwork. In such a case, the matching fields of the anomaly expressing an IP address would be then expressed as



(a) Topology and flow tables (b) Graph representation

Fig. 1: OpenFlow swicthes network model

a subnet. These approximations in finding the root causes of an anomaly will be evaluated in section V.

- $matching(r, f)$ returns true if the fields f match the rule r, false otherwise.
- $tail(e)$ returns the tail of the directed edge e.

Algorithm 1 Finding origins of an anomaly: $origins(l, f, g)$

Require: l: location of anomaly
Require: f: header fields expressing the anomaly
Require: g: graph based representation of OpenFlow switches
1: $in\_edges \leftarrow ingoing\_edges(g, l)$
2: **for** $e \in in\_edges$ **do**
3: $r \leftarrow rule(e)$
4: $new\_fields \leftarrow inv\_mod(r, f)$
5: **if** $matching(r, new\_fields)$ **then**
6: $next \leftarrow tail(e)$
7: $origins \leftarrow origins(next, new\_fields, g)$
8: $origins \leftarrow l \bigcup origins$
9: **end if**
10: **end for**
return $origins$

Algorithm 1 is recursive and starts by exploring the edges which may have forwarded the traffic corresponding to the anomaly. To achieve that, all edges pointing to the location where the anomaly has been detected are retrieved in line 1. For each of them, the algorithm applies inverse modifications based on the associated rules in line 4. In fact, the goal is to check if the anomaly traffic characterized by the matching fields f might have been processed by the rule r while such a rule may have applied modifications afterwards. Hence, applying inverse modifications is necessary as a first step before verifying the matching between r and f in line 5. If a match exists, then the tail of the edge associated in the rule is considered for further analysis by the recursive process. The current node being explored is added to the origins of the anomaly (line 8) discovered by the recursive execution of the algorithm. In fact, the origins retrieved by the our algorithm is all switches on the anomaly paths.

For sake of clarity and due to limited space, this algorithm does not take in account multiple flow tables on a switch but this is realized by considering each rule in intermediate flow table as another edge.

V. Evaluation

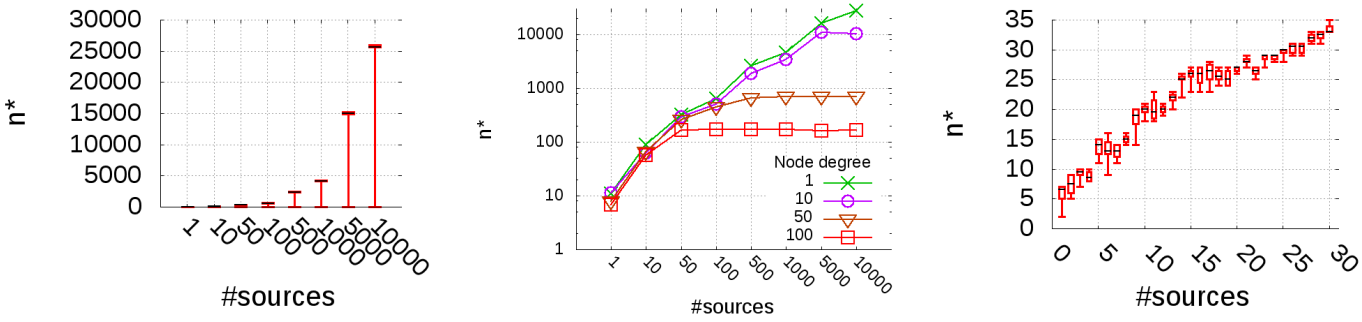
A. Methodology

Our approach to trace back anomalies is based on a deterministic process and is thus able, by design, to find all the potential paths, and so sources, of an anomaly in the network. However, the granularity of the defined rules with OpenFlow will impact on the accuracy of the localized paths. For example, with highly generic rules like those dedicated to forward packets to a specific destination address, tracing back an anomaly targeting a TCP specific port like 80 would consider all incoming paths towards this host without port distinction, which thus represents an overhead for future forensic analysis. For example, it might lead to retrieve useless systems or traffic logs related to all incoming paths.

Therefore, the goal is not to evaluate how much an anomaly traffic is covered by the rules, as there is always a rule otherwise such a traffic would have not been forwarded. The evaluation focuses on assessing how many nodes are considered by our approach as having forwarded the traffic related to the anomaly. The best case would be when there is exactly one rule per switch on the anomaly traffic path solely, which exactly and exclusively matches the anomaly traffic. From a practical point of view, it is not feasible because it would require to create a rule for all potential anomalies a priori, *i.e.* for all combination of unique values (IP and Ethernet addresses, ports, etc.) in the matching fields in a flow table entry. However, such an ideal case is considered as a baseline during our evaluation. Therefore, the main metric calculated is the overhead ratio:

$$r = \frac{n_{tb}}{n*} \tag{1}$$

where $n*$ represents the number of switches that have effectively forwarded the anomaly traffic while n_{tb} is the number of switches which have been retrieved by our method. If anomaly detection is done in a timely manner, it is possible to install highly specific rules before the anomaly is ended.



(a) Skitter, sources and destinations randomly selected (b) Skitter, attack node with a degree in $\{1, 10, 50, 100\}$ (c) Fat-tree, sources and destinations randomly selected

Fig. 2: Exact number of switches which have forwarded the anomaly ($n*$)

B. Datasets

Two distinct topologies are used:

- a real network topology extracted from a 100 millions links subgraph of a *Skitter*[4] dataset which has been constructed by active measurements on Internet.
- a fat-tree topology which is largely adopted for designing data center networking as they are much scalable than hierarchical topologies [20]. In addition, attacks within a data center become popular especially with multi tenant cloud services [21].

A fat-tree topology [22] is tuned through a simple parameter k and is composed of 3 layers. At the higher level, there are $(k/2)^2$ core switches where each of them is connected to k pods. A pod ($(k/2)^2$ end-hosts and $k/2$ switches with k ports) consists of the aggregation and edge layer where each aggregation switch is connected to $k/2$ edge switches and $k/2$ core switches. Each edge switches connected to $k/2$ end hosts and $k/2$ aggregation switches. In our experiment, $k = 4$ leads to a topology with 30 switches.

C. Ideal OpenFlow Configuration

In this experiment, the ideal configuration is considered and so the goal is to compute $n*$ in equation (1) assuming different configurations. In particular, a major parameter impacting on the number of switches involved in capturing the traffic is related to the number of sources of the anomaly. Actually, most of major threats today are distributed. In Figure 2(a), assuming the *Skitter* topology, the number of sources vary between 1 and 10000 and $n*$ in the same way. It is important to note that this experiment has been done by considering 100 distinct targets before computing the quartile values. Due to the lack of routing information in our datasets, we consider the anomaly traffic paths as the shortest paths between the sources and the destination. As shown in the figure, the variations between each individual target are small and most of of the values are concentrated around the mean in the box plot. Although the scale is not linear in the figure for sake of clarity, the increase in number of switches is almost linear. For instance, around 500 switches would forward the anomaly traffic when the latter originates from 1000 distinct sources. For the fat-tree topology, the same increase can be observed in Figure 2(c) (the maximal number of sources is 30 as it is the total number of nodes). Such an increase is due to a larger number of paths when the packet is coming from several sources even if some switches are common to multiple paths.

In Figure 2(b), nodes are randomly selected based on their degrees (1, 10, 50 or 100) because attacks can even target switches, routers or middleboxes in the network. Therefore, the nodes with high degrees are located in the core of the topology whereas low degrees are representing the nodes at the edge. Naturally, being located at the edge implies that path from the sources of the anomaly are longer and so $n*$ is higher. There is a notable difference when there are at least 50 sources.

D. Overhead evaluation

The ratio r considers the real number n_{tb} of switches which have rules that match the anomaly even if the anomaly traffic has not been forwarded by all of them. To achieve that, a proportion p of the ingoing edges ($ingoing\_edges(g, l)$) is considered at each reverse hop from the destination to the sources in addition of the ones composing the shortest path. This allows to take in account the approximations due to the granularity of installed rules explained in Section IV-C.

In Figure 3(a), this proportion varies between 0.1 and 0.9 while the number of attack sources is 100. The overhead remains limited when the proportion is below 0.3. Even in such a case ($r < 10$), it corresponds to traverse almost 8000 vertices in the constructed graph. However, this considers many paths including long ones. Figure 3(b) shows the number of additional vertices at each hop. Assuming a maximum number of 5 hops (which is reasonable for a university network), less than 50 switches are considered to traceback to all the sources of an anomaly (with $p = 0.4$) which can be internal or external. Assuming external sources, limiting the number of hops is equivalent to have only OpenFlow switches within the internal network. Thus, our method is able to locate the entry points in the network (not the exact origins of the attack) in such a case. Finally, Figure 3(c) highlights the negligible impact when case of the fat-tree topology of the data centers, which are the most promising candidates for deploying SDN technologies.

[4]The CAIDA UCSD Macroscopic Skitter Topology Dataset, http://www.caida.org/tools/measurements/skitter, dataset LNK0304

(a) Skitter, overhead ratio r

(b) Skitter, additional swicthes ($n_{tb} - n*$) at each hop

(c) Fat-tree, overhead ratio r

Fig. 3: Impact of the granularity of rules which may entail to consider additional routes as traversed by the anomaly, number of sources = 100, average over 100 destinations

VI. Conclusion

In this paper, a novel approach for anomaly tracebacking has been proposed. The main motivation is to rely on OpenFlow compatible switches which tend to be deployed at large scales in local, backbone or data center networks. We defined a graph-based model to represent an OpenFlow enabled network which is then analyzed to identify the potential paths of an anomaly. This is helpful then to limit the selection of data (like traffic or system logs) for forensic analysis. However the granularity of the rules has a great impact on unhelpful selected paths. Because there is no publicly available dataset of deployed OpenFlow networks (and so rule configurations), our evaluation assesses multiple generic configurations (when p varies) which could then be mapped to real deployment scenarios. In future work, we plan to extend our approach by considering flow statistics to weight the anomaly and so apply a stochastic analysis to retrieve the anomaly paths.

Acknowledgements: This work is partially funded by Flamingo, a Network of Excellence project (ICT-318488) supported by the European Commission under FP7 and by ANR DOCTOR project.

References

[1] Y. Yu, "A survey of anomaly intrusion detection techniques," *J. Comput. Sci. Coll.*, vol. 28, no. 1, Oct. 2012.

[2] Y. Gilad and A. Herzberg, "Lot: A defense against ip spoofing and flooding attacks," *Trans. Inf. Syst. Secur.*, vol. 15, no. 2, Jul. 2012.

[3] A. networks, "Worldwide infrastructure security report (2013 report)," Tech. Rep., 2014.

[4] V. Paxson, "An analysis of using reflectors for distributed denial-of-service attacks," *SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 3, pp. 38–47, Jul. 2001.

[5] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[6] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, "PayLess: A Low Cost Netowrk Monitoring Framework for Software Defined Networks," in *IEEE/IFIP Network Operations and Management Symposium (NOMS 2014)*.

[7] S. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks," *Communications Surveys Tutorials*, vol. 15, no. 4, pp. 2046–2069, 2013.

[8] H. Wang, C. Jin, and K. G. Shin, "Defense against spoofed ip traffic using hop-count filtering," *Transactions on Networking*, vol. 15, no. 1, pp. 40–53, 2007.

[9] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Practical network support for ip traceback," in *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication - SIGCOMM*. ACM, 2000.

[10] D. X. Song and A. Perrig, "Advanced and authenticated marking schemes for ip traceback," in *Annual Joint Conference of the IEEE Computer and Communications Societies - INFOCOM*. IEEE, 2001.

[11] M. T. Goodrich, "Efficient packet marking for large-scale ip traceback," in *Conference on Computer and Communications Security - CCS*. ACM, 2002.

[12] L. Lu, M. C. Chan, and E.-C. Chang, "A general model of probabilistic packet marking for ip traceback," in *Symposium on Information, Computer and Communications Security - ASIACCS*. ACM, 2008.

[13] A. Snoeren, C. Partridge, L. Sanchez, C. Jones, F. Tchakountio, B. Schwartz, S. Kent, and W. Strayer, *Transactions on Networking*.

[14] M. Sung, J. Xu, J. Li, and L. Li, "Large-scale ip traceback in high-speed internet: Practical techniques and information-theoretic foundation," *Transactions on Networking*.

[15] U. Hoelzle, "Openflow @ google," in *OpenNetSummit*, 2012. [Online]. Available: http://www.opennetsummit.org/archives/apr12/hoelzle-tue-openflow.pdf

[16] N. L. M. Van Adrichem, D. Doerr, and F. A. Kuipers, "OpenNetMon: Network Monitoring in OpenFlow Software-Defined Networks," in *Network Operations and Management Symposium (NOMS)*. IEEE/IFIP, 2014.

[17] A. Bates, K. Butler, A. Haeberlen, M. Sherr, and W. Zhou, "Let SDN be your eyes: Secure forensics in data center networks," in *Proceedings of the NDSS Workshop on Security of Emerging Network Technologies (SENT'14)*, Feb. 2014.

[18] R. Perlman, "An algorithm for distributed computation of a spanningtree in an extended lan," in *Proceedings of the Ninth Symposium on Data Communications, SIGCOMM*. ACM, 1985.

[19] T. lin, J.-M. Kang, H. Bannazadeh, and A. Leon-Garcia, "Enabling SDN Applications on Software-Defined Infrastructure," in *Network Operations and Management Symposium (NOMS)*. IEEE/IFIP, 2014.

[20] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *SIGCOMM Conference on Data Communication*. New York, NY, USA: ACM, 2008.

[21] A. Shieh, S. Kandula, A. Greenberg, and C. Kim, "Seawall: Performance isolation for cloud datacenter networks," in *Conference on Hot Topics in Cloud Computing - HotCloud*. USENIX, 2010.

[22] D. Abts and J. Kim, *High Performance Datacenter Networks: Architectures, Algorithms, and Opportunities*. Morgan & Claypool Publishers, 2011.