Assessing the Quality of Flow Measurements from OpenFlow Devices

Luuk Hendriks[1], Ricardo de O. Schmidt[1], Ramin Sadre[2], Jeronimo A. Bezerra[3] and Aiko Pras[1]

[1]University of Twente, the Netherlands
[2]Université Catholique de Louvain, Belgium
[3]Florida International University, USA

***Abstract*—Since its initial proposal in 2008, OpenFlow has evolved to become today's main enabler of Software-Defined Networking. OpenFlow specifies operations for network forwarding devices and a communication protocol between data and control planes. Although not primarily designed as a traffic measurement tool, many works have proposed to use measured data from OpenFlow to support, *e.g.,* traffic engineering or security in OpenFlow-enabled networks. These works, however, generally do not question or address the quality of actual measured data obtained from OpenFlow devices. Therefore, in this paper we assess the quality of measurements in real OpenFlow devices from multiple vendors. We demonstrate that inconsistencies and measurement artifacts can be found due to particularities of different OpenFlow implementations, making it impractical to deploy an OpenFlow measurement-based approach in a network consisting of devices from multiple vendors. In addition, we show that the accuracy of measured packet and byte counts and duration for flows vary among the tested devices, and in some cases counters are not even implemented for the sake of forwarding performance.**

***Index Terms*—OpenFlow, traffic measurements, counters**

I. Introduction

Although not a new concept, the paradigms of Software-Defined Networking (SDN) and programmable networking have recently gained lots of attention from academia and industry. We can now find many works in the literature that propose solutions for traffic engineering problems by implementing SDN concepts and, as its main enabler, OpenFlow [1]. These solutions take advantage of management flexibility brought by SDN and OpenFlow on decoupling the data and control planes.

The evolution of OpenFlow is somewhat similar to that of NetFlow, which started as a side-product from a traffic forwarding technology to become one of today's major measurement tools. Right now, it is impossible to foresee whether OpenFlow will ever evolve towards a measurement-specific technology. The fact is, however, that many works have been taking advantage of the potential to combine control, forwarding and measurement features all embedded in a single technology. For example, works have been proposing to use measured data from OpenFlow for a variety of network management problems, such as capacity planning and provisioning [2], [3], [4], [5], [6], energy-efficiency [7], [8] and security [9], [10], [11]. The survey in [12] also shows how comprehensive the use of traffic measurements in SDN is. Most of the works in the literature have validated their proposals by means of simulation, using tools such as Mininet [13]. Moreover, many surveys such as [14], [15], [16], [17], [18], [19], [20] provide an extensive overview of potential future applications of SDN, mostly relying on some sort of measurement data.

Due to the rapidly increasing interest and adoption of OpenFlow, all the major vendors of networking devices have already incorporated OpenFlow support into their routers and switches. Moreover, new vendors have brought the first "pure OpenFlow" devices to the market, *i.e.,* OpenFlow-only switches. Since it is still a new technology, we can expect that OpenFlow implementations are not yet homogeneous amongst devices from different vendors. Independent and immature implementations might give rise to certain artifacts that compromise the quality of the measured data. This would ultimately invalidate the practicality of many proposed solutions in the literature, especially those aiming at large deployments on heterogeneous infrastructure.

Contributions. In this paper we assess the quality of measurement operations and measured data from various OpenFlow devices deployed in controlled testbeds. We focus on flow-level measurements. Our analysis is divided in two parts, namely a qualitative and a quantitative assessment. In the qualitative assessment (§ IV) we compare the measurement capabilities of each tested device, also bringing up peculiarities not documented by the respective vendors and comparing our findings to what is defined in the pertinent OpenFlow specifications regarding traffic measurements. In the quantitative assessment (§ V), we study the accuracy of measured data, *i.e.,* per-flow packet and byte counts and flow duration. We demonstrate that implementations of OpenFlow in devices from different vendors have specific limitations and artifacts, limiting the use of their measurements, especially in a multi-vendor deployment. Alongside to our contributions, we make all the developed tools used in our experiments publicly available in online repositories.

It should be emphasized that the goal of this paper is to assess *available OpenFlow implementations*, and not whether a different measurement approach or technology, such as described in [21], [22], would provide better results.

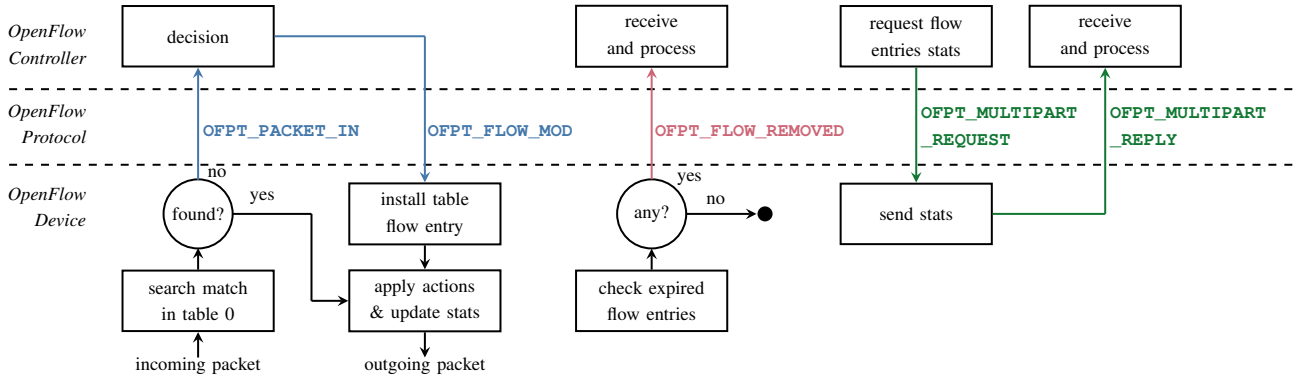


Fig. 1. Summarized messages exchange between OpenFlow controller and device.

II. OpenFlow Background

OpenFlow has become today's main enabler of SDN, and many networking vendors have it implemented in their devices, which can be either *full OpenFlow* or *OpenFlow-capable* switches/routers. While the former are specifically designed to operate OpenFlow, the latter support OpenFlow in parallel to traditional packet forwarding.

Most switches support OpenFlow 1.0.0 [23] published in 2009, and this version is the mostly used in related work. OpenFlow 1.5.0 [24] is currently the latest published standard by the ONF[1]. However, few vendors have implemented higher versions than OpenFlow 1.3.1 [25]. There are many crucial differences between OpenFlow versions that directly influence what kind of measured data we can obtain from the device. Specifics of these differences are out of this paper's scope. Nonetheless, one example is the support for IPv6 in the *match fields* from OpenFlow 1.2 onwards.

A. Flow Tables

OpenFlow devices must have at least one *flow table*. The table contains flow entries that consist of, among others: 1) the *match fields* specifying the set of properties that identify flows; 2) an *action set* with instructions on what to do with the matching packets; 3) a *priority* to resolve conflicting situations of multiple matches for a single packet; 4) the *duration* telling for how long the entry has been active in the table; 5) *packets* and *bytes counters*; 6) *timeouts* to determine the entry's lifetime; and 7) the *cookie* which is a unique identifier for the flow entry and set by the controller.

Forwarding instructions can propagate a packet through a pipeline of flow tables before a terminal action is taken (drop or forward the packet). Table pipelining is, however, out of this paper's scope.

The match fields of a flow entry define what a flow is. Flows can be defined by the traditional 5-tuple of NetFlow (source and destination IP addresses, source and destination ports, and transport protocol), or by a single field, such as the VLAN ID. If an incoming packet matches more than one entry, the entries' priorities define which one to be considered.

[1]https://www.opennetworking.org/

B. Protocol Messages

OpenFlow messages are used for communication between controllers and devices, and some of them are directly related to traffic measurements. Figure 1 shows a summarized scheme of messages used for traffic measurements.

For every incoming packet the device checks for a matching flow entry. If found, the entry's counters are updated and actions are taken as defined in the action set. Otherwise, the packet hits the wildcard entry ("matching all" rule). Typically, the action set for the wildcard rule is to send an `OFPT_PACKET_IN` message to the controller. The controller understands that no matching was found and tells the switch what to do with the incoming packet. The controller replies to the switch with an `OFPT_FLOW_MOD` message of type `OFPFC_ADD` that contains instructions for the new entry that will match the next packets for the same flow. Existing entries can also be modified by the controller with the same message but with type `OFPFC_MODIFY`.

When an entry's timeout expires, the OpenFlow device removes the entry from the flow table and sends an `OFPT_FLOW_REMOVED` to the controller, containing the entry's packet and byte counts. This message is only sent if explicitly requested; when adding or modifying an entry, the controller must set the `OFPFF_SEND_FLOW_REM` flag in the `OFPT_FLOW_MOD` message.

With an `OFPT_MULTIPART_REQUEST` message the controller can at any time request for statistics of individual flows, aggregates, tables, ports, among others. The OpenFlow device replies with an `OFPT_MULTIPART_REPLY` message to the controller containing the requested statistics.

C. Flow Timeouts

The controller can specify *hard* and a *idle timeouts* for individual flow entries using the `OFPT_FLOW_MOD` message. The hard timeout defines the maximum lifetime of a flow entry and when expired the entry, regardless of being active or not, is removed from the flow table. The idle timeout defines the maximum interval between the last matching packet and the removal of the entry from the table due to flow inactivity. Timeouts of "infinity" are typically set to the wildcard rule.

TABLE I
TESTED OPENFLOW DEVICES

Device	Firmware	Used OF version
Pica8 P3295	PicOS 2.6	1.3
HP 2920-24G	WB.15.17.0007	1.3
Brocade CES	5.8.0bT183	1.3
Brocade MLXe	5.7.0cT163	1.0
Juniper MX240	13.3R6.5	1.0

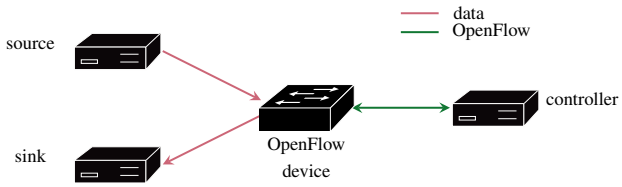


Fig. 2. Measurement setup topology.

TABLE II
COMPOSITION OF TESTED MATCH FIELDS

ID	fields
5-t	nw_src, tp_src, nw_dst, tp_dst, nw_proto
5-tv6	ipv6_src, tp_src, ipv6_dst, tp_dst, ipv6_proto
icmp	nw_src, nw_dst, nw_proto, icmp_type, icmp_code
icmp6	ipv6_src, ivp6_dst, nw_proto, icmp_type, icmp_code
ipv6	ipv6_src, ipv6_dst, ipv6_label

Abbreviations:
nw_src, nw_dst, nw_proto: IPv4 source, destination and protocol
ipv6_src, ipv6_dst, ipv6_proto: IPv6 source, destination and protocol
ipv6_label: IPv6 flow label specification
tp_src, tp_dst: source and destination ports
icmp_type, icmp_code: ICMP type and code

III. EXPERIMENTAL SETUP

A. Tested OpenFlow Devices

We carried out experiments on multiple testbeds consisting of OpenFlow devices from different vendors (Table I). We included two devices from Brocade in our experiments; the Brocade CES is an Ethernet switch, and the MLXe is a router and, hence, one could expect better support for traffic accounting and measurements from the latter. We used OpenFlow 1.3 in the Pica8, HP and Brocade CES devices. By the time of our experiments Pica8 firmware with support for OpenFlow 1.4 was still on beta. For Brocade MLXe and Juniper devices we used OpenFlow 1.0. The MLXe support to OpenFlow 1.3 would have required a hardware update, and Juniper only supported this version of OpenFlow. The HP can be configured to operate either in Hardware (HW) or Software (SW) mode, which are not complementary.

B. Network Setup

The devices in Table I were all assessed using the same topology shown in Figure 2. The *source* and the *sink* (virtual) machines are used to, respectively, send and receive traffic through the OpenFlow device, and the *controller* (virtual) machine is connected to the management port of the OpenFlow device. This setup gives us full control over the traffic transfer: at the source machine we used `tcpreplay` to replay traffic traces to the sink, where we used `tcpdump` to check whether all traffic sent was correctly received. All the links are 1 Gbps and no concurrent traffic was present in the testbeds during the experiments.

C. Measurement Approach

All traces replayed in our experiments are `pcap` files consisting of synthetic traffic generated using `Scapy` (Python library). Synthetic traces allowed for variation of traffic characteristics to assess the specific OpenFlow features of interest. To validate the flow data obtained from OpenFlow, we developed a script that uses the *flow entry cookie* to identify unique flows within the replayed traces. Note that we had full control over the flow table during the measurements on the devices, *i.e.,* no rules other than ours were installed. Our scripts and example traces are publicly available at https://github.com/ut-dacs/openflow-accuracy-measurement.

We used *proactive* and *reactive measurements* to assess the quality of measurement operations and data from OpenFlow devices. For proactive measurements we used `ovs_ofctl`[2], a command-line tool from Open vSwitch to monitor and manage OpenFlow devices. Using `ovs_ofctl` on the controller we requested flow statistics from the OpenFlow device. For the reactive approach we implemented a controller on top of Ryu[3] that receives and processes `OFPT_FLOW_REMOVED` messages sent by the device when a flow entry's timeout expires. Our controller implementation is available at https://github.com/ut-dacs/openflow-passive.

To avoid an overwhelming number of `OFPT_PACKET_IN` and `OFPT_FLOW_MOD` exchanges between controller and OpenFlow devices, at the start of a trace replay we preemptively preloaded the flow table with entries matching all flows within the trace. This ensured that the communication between controller and device to not be a source of inaccuracy for the traffic measurements.

IV. QUALITATIVE ANALYSIS

Our goal of our qualitative analysis is to understand to what extend relevant OpenFlow features to measurements are supported and correctly implemented by the various devices. In particular, we looked into key matching (§ IV-A), flow entry's timeouts (§ IV-B), flow entry's overlap checking (§ IV-C), and counters (§ IV-D).

A. Flow Key Matching

OpenFlow gives flexibility on the flow key definition (match fields). We assessed the support of OpenFlow devices for

[2]http://openvswitch.org
[3]http://osrg.github.io/ryu/

TABLE III
SUMMARY OF THE QUALITATIVE ANALYSIS

Device	Match fields composition					Timeouts		Entry overlap checking	Statistics counters		
	5-t	*5-tv6*	*icmp*	*icmp6*	*ipv6*	*Idle*	*Hard*		*Packets*	*Bytes*	*Duration*
Pica8 P3295	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓
HP 2920-24G	✓	✓	✓	✓	✓	✓	✓	✓			
HP 2920-24G HW									✓		✓
HP 2920-24G SW									✓	✓	✓
Brocade CES	✓	✓	✓	✓	✓			✓		✓	✓
Brocade MLXe	✓		✓					✓	✓	✓	✓
Juniper MX240	✓		✓			✓	✓	✓	✓	✓	✓

various keys, as shown in Table II. The combinations *5-t* and *5-tv6* are typical 5-tuple key for, respectively, IPv4 and IPv6. Similarly, we define the ICMP keys *icmp* and *icmp6*. In addition, we define the *ipv6* key using IPv6 labels. The IPv6-specific combinations were tested only for devices running OpenFlow 1.3 (Table I).

To perform this measurement we loaded flow tables with entries containing specific keys using `ovs_ofctl`. We then sent matching packets through the device and checked the flow statistics to verify if the number of matched packets for each entry corresponded to the number of packets sent. Table III shows the results of this experiment. Pica8, HP and Brocade CES accepted the insertion of all key compositions and correctly matched packets to these entries. Brocade MLXe and Juniper devices accepted the IPv4-only keys *5-t* and *icmp*, also correctly matching their respective packets.

B. Flow Timeouts Support

Flow timeouts is one of the main concepts in NetFlow/IPFIX, and they can be used to define the temporal granularity of measurement data. Shorter timeouts can provide a better view on traffic dynamics at smaller timescales [26]. We used the reactive measurement approach to assess if OpenFlow devices support and properly implement timeouts. We loaded the flow table with entries, setting values for both hard and idle timeouts. The flow entries were added with the flag `OFPT_SEND_FLOW_REM`, which explicitly requests the device to send an `OFPT_FLOW_REMOVED` to the controller once a flow entry is removed due to timeout expiration. We then simply waited for the entry's timeout expiration.

Table III summarizes the support for timeouts on the tested devices. Pica8, HP and Juniper support both hard and idle timeouts and also correctly implement them. Brocade devices, however, do not implement any of the timeouts specified by OpenFlow. Timeouts are acknowledged in the Brocade devices specifications, and `OFPT_FLOW_MOD` messages containing timeout information are successfully processed by the devices. However, the provided timeout values are simply ignored.

By not implementing timeouts, a device transfers the responsibility for removing expired entries to the controller (or an application running on top of the controller). The controller has to somehow keep track of the time/duration of each installed flow entry in the switch and actively remove entries using an `OFPT_FLOW_MOD` message of type `OFPFC_DELETE` (or `OFPFC_DELETE_STRICT`). This adds complexity to the controller and increases the number of messages exchanged between controller and device, since the controller must request flow statistics from the device and send delete messages.

Obviously, the lack of timeouts also has an impact on potential measurement applications. The `OFPT_FLOW_REMOVED` message contains important information on the expired flow, such as counters and duration. Without this message, a measurement application has to periodically contact the OpenFlow device, which results in additional overhead.

Finally, there could also exist devices that actually support timeouts but do not support flow removed messages. In such cases, the controller might lose measurement data if the request for statistics reaches the OpenFlow device after the expiration of flow entry's timeouts. Nonetheless, none of the tested OpenFlow devices showed this behavior.

C. Flow Entry Overlap Checking

OpenFlow specifies ways to avoid duplicate flow entries, what helps ensuring the correctness of the flow measurements. This is achieved by setting the flag `OFPFF_CHECK_OVERLAP` in `OFPT_FLOW_MOD` messages, requesting the device to check for duplicates before adding a new entry.

To test the overlap checking support on OpenFlow devices, we used `ovs_ofctl` to send two add-request messages (with the same match fields and priority) to the device with the overlap check flag set. For the first request, the device is expected to simply add the entry. For the second request, however, the device was expected to react on the duplicate entry.

Table III shows that all OpenFlow devices implement the overlap checking. However, some of them do not fully adhere to the OpenFlow specification. If the add request contains the check overlap flag and an overlapping entry exists, the new entry should not be added to the flow table and an `OFPT_ERROR_MSG` message of type `OFPET_FLOW_MOD_FAILED` and code `OFPFMFC_OVERLAP` should be sent to the controller. All

the tested devices successfully generate this error message for overlapping entries.

Pica8 makes a distinction between overlapping and identical entries (entries with completely identical fields), which is not in accordance with the OpenFlow specification. In case of an identical entry with overlap flag, Pica8 ignores the flag and resets the flow duration of the existing entry, but keeps the packet and byte counters unchanged.

Most devices correctly implement the OpenFlow specification in case the add-request message does *not* contain the check overlap flag. Overlapping entries are simply added, while identical entries are handled differently depending on the OpenFlow version. Pica8, HP and Brocade CES correctly reset flow duration and keep counters untouched (v1.3). The only exception is the HP device in software mode: it resets both duration and counters. Brocade MLXe behaves as expected in OpenFlow 1.0; in the presence of the check overlap flag the existing identical entry should be removed from the table and the new entry added (resetting both duration and counters).

D. Flow Counters Support

The OpenFlow specification defines that for each entry in a flow table, packet and byte counters should be maintained. The implementation of these counters is, however, vendor-specific. In previous experience [27] we showed that although a value was returned for packet counter upon request, the counter was effectively not implemented. The device would actually count the number of bytes and then roughly estimate the packet number; dividing the number of bytes by 100 (one hundred).

To assess if counters are properly implemented in the OpenFlow devices, we added flow entries using `ovs_ofctl` and sent traffic through the device. The counters values were retrieved by proactive and reactive measurements. Table III summarizes our findings.

Since our first experience with Pica8 (with PicOS 2.3 firmware), the vendor has changed the support for measurement operations of their OpenFlow devices. The packet counter has been removed and the byte counter is now providing more accurate counts than previously observed [27]. For HP, in the software mode only byte counters are available and in the hardware mode both packets and bytes are counted. Brocade MLXe counts packets and bytes, but Brocade CES only bytes. Finally, Juniper implements both counters. The presence of a counter does not guarantee correctness of counts, because hardware limitations and implementation decisions by the vendors affect accuracy (§ V-A).

OpenFlow also specifies flow entry duration, which is of major importance for applications that use measured data to estimate, for example, packet (pps) or bytes rates (bps). All the tested devices keep track of the flow entry duration. However, the correctness of the duration value depends on how it is obtained (§ V-B).

V. Quantitative Analysis

In this section, we present and discuss our findings resulting from the quantitative assessments of measurements from the OpenFlow devices. The accuracy of the packet and byte counters are tested (§ V-A), as well as the granularity of the timeout mechanisms (§ V-B).

TABLE IV
Extra bytes per packet

Device	Extra bytes
Pica8 P3295	4
HP 2920-24G	0
Brocade CES	24
Brocade MLXe	4
Juniper MX240	4

A. Accuracy of Flow Entry Counters

As shown in § IV-D, all tested devices support per-flow packet counters, byte counters, or both. To assess the accuracy of the counters, we sent traffic from the source to the sink machines and retrieved statistics afterwards at the controller by the proactive and reactive methods described in § III-C. The following steps were performed:

1) Create a traffic trace in *pcap* format, a file in *ovs* format containing the flow definitions, and a "ground truth" file containing the packets and bytes counts for each flow.
2) Preload the flow definitions (flow entries) to the flow table of the device and replay the traffic trace. (Note that the maximum number of preloaded flow entries was empirically defined for each device.)
3) After the trace has been replayed, retrieve the statistics from the device and compare with the ground truth.

In addition, the traffic arriving at the sink machine was collected to verify that the packets were forwarded correctly.

We first noticed that several devices systematically count more bytes per packet than the actual Ethernet frame size, independently of the characteristics of the incoming traffic (see Table IV). For example, the Brocade CES counts 24 bytes per packet extra, which can result in a significant error of the byte counter if the traffic consists of a large number of small packets. The reason for this behavior is not known to us and an inspection of the traffic collected at the sink machine showed no difference to the traffic sent from the source machine.

Another source of error is the way counters are updated. Some devices internally update their counters by a periodically executed process. This results in inaccuracies if a device is queried for statistics between two updates. All but the Pica8 and the HP device in software mode showed this behavior, with varying update intervals. The HP switch in hardware mode is, by default, configured to an update interval of 20 s. The minimum configurable value is 1 s. Such as value can drastically decrease the negative impact on the accuracy.

For the other devices, no update interval is documented. Using the proactive approach, we queried the devices every second for a minute upon completion of replaying a trace in order to determine the time where the resulting statistics become static, *i.e.,* counters are not updated anymore. For the

Brocade devices, up to 30 s of delay was observed, and for Juniper a maximum of approximately 10 s. In software mode, the HP switch bases its forwarding rate upon a configuration option, limiting the rate at 100 *pps* by default. Replaying traces at higher rates causes incomplete forwarding and incomplete statistics. We measured the ratio of forwarded packets at the default 100 *pps* and the maximum configurable 2000 *pps* and observed that the switch is not capable of forwarding at the maximum rate (see Figure 3). This might be considered rather a performance problem than a measurement accuracy problem, but as the software mode provides more statistics (*i.e.,* packet counts), one should be aware of this limitation.

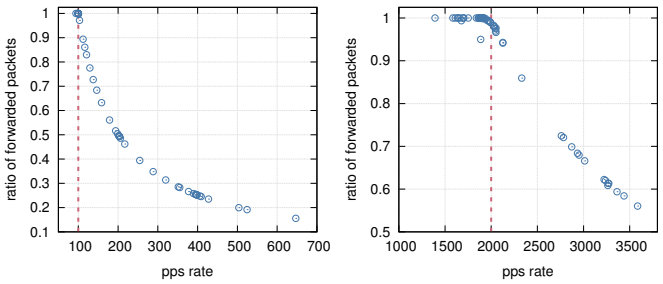
Furthermore, two devices showed incorrect statistics caused by other problems. The Brocade CES does not count traffic sent immediately after adding the flow entries to the flow table. We investigated this by adding a single flow entry, assuring it has been installed by querying for statistics, and replaying a trace with matching packets and with constant packet size and packet rate. After receiving all packets at the sink machine, the statistics were retrieved, showing inaccurate counters. Using a flow of 10,000 packets sent at 1000 *pps*, we observed a mean of 1231 missing packets with a standard deviation of 409 over 16 runs. In terms of time, this equals to approximately 1.2 seconds of lost counting.

Another artifact on the Brocade CES are flows without any statistics set (packet and byte counters are zero). Even when loading less than the maximum number of flow entries in the table, this behavior was observed, but not in a deterministic way: multiple runs of the same trace resulted in either complete statistics or in a constant number of entries with missing statistics. In the latter case, not always the same flow entries were affected. However, it always concerned consecutively inserted flows, that means, after ordering the obtained flows by their respective cookie value, all flows missing statistics formed a single block. We believe this might be caused by a bug, or some sort of heuristic to trade-off forwarding performance against statistics accounting.

Lastly, the Juniper showed erratic behavior related to the maximum size of the flow table. We measured the number of installable *5-t* flow entries, which resulted in a mean of 6325 entries, but with a standard deviation of 442 over 15 runs. It seems that flows are added in a non-deterministic way, with no constant maximum of entries. Replaying traces with 6000 flows resulted in missing statistics for all flow entries. However, counters showed deviations from the ground truth even for smaller flow table sizes. Attempts with 1500 flows resulted in a mean of 3% of packets that were forwarded but not accounted for, when the trace was replayed at 1000 *pps*. At 5000 *pps* that ratio increased to 8.7%. In addition to inaccuracies of counters, the device had problems matching packets to installed flow entries, resulting in sending unnecessary `PACKET_IN` messages to the controller.

B. Accuracy of Flow Durations

Besides the packet and byte counters, OpenFlow statistics also report flow duration. As this duration can be used to calculate rates for a certain flow, *e.g., bps* or *pps*, any inaccuracy will directly affect those calculations. Unlike NetFlow/IPFIX, flow duration in OpenFlow is not based on the packet observation times. Instead, the duration specifies how long the flow entry has been in the flow table.



(a) 100 pps (b) 2000 pps

Fig. 3. Ratio of forwarded packets on the HP switch in software mode. Observe the different ranges of the y-axes. The vertical line at 100 *pps* and 2000 *pps*, respectively, shows the configured packet forwarding rate.

One way to emulate the NetFlow/IPFIX style of flow duration is to use OpenFlow's idle timeout mechanism. However, since the timeout is included in the flow duration, any anomalies in the behavior of the timeout mechanism would directly influence the accuracy of the duration estimation. To assess this, we installed flow entries with an idle timeout of 60 s and collected the `OFPT_FLOW_REMOVED` messages, without sending any traffic through the OpenFlow device. The obtained durations should, therefore, be equal to the idle timeout, as no packet was matched for the flow entry. For the devices that support flow expiration, *i.e.,* all but the Brocades, the mean and standard deviation of the inaccuracy were calculated over 10 runs. For Pica8, the reported duration was 271 ms longer than the actual timeout (standard deviation 24 ms). For the HP switch, the reported duration was 604 ms longer in hardware mode, and 609 ms in software mode. The standard deviation in hardware mode however was higher, with 296 ms compared to 186 ms for software mode. The Juniper showed 486 ms extra, with a standard deviation of 385 ms. We also tried other timeout values and obtained similar means and deviations.

A possible explanation could be that the timeout is implemented as a periodic process scanning the flow table for expired entries. In that case, the Pica8 switch would scan the table with a period of around 500 ms, twice as frequently as the HP and Juniper devices. Of course, another explanation could be that the used timers simply do not have millisecond precision. Similar problems have been observed on NetFlow/IPFIX devices by us and other researchers (see § VII).

Interestingly, the stability of the timeout mechanism seems to depend on the workload of the device. When tested with traffic, especially the HP and Juniper devices showed deviations of several seconds. This indicates (as one could expect) that the table-scanning process runs with a lower priority than the forwarding engine.

VI. Discussion

A major challenges for operators today is to deploy a measurement-based application on top of an OpenFlow-based network consisting of devices from multiple vendors. Differences between OpenFlow implementations are so vast that they might compromise operations relying on measured data. In the following, we discuss our findings in this context.

On the devices tested by us, the **counter updates** are a potential source of inaccuracy for packet and byte counters. As explained in § V-A, the time interval between counter updates might cause flow entries to be reported with outdated statistics, *i.e.,* statistics proactively queried by the controller do not include the packets forwarded since the last counter update. The same is true for the statistics sent to the controller when a flow entry expires due to a timeout. In order to avoid this problem, the controller would have to synchronize its statistics requests with the counter updates on the OpenFlow device — a hardly feasable solution in multi-vendor scenarios where the update interval is vendor specific.

In addition, the inconsistent number of **extra bytes** requires model-specific corrections when processing counter values from different OpenFlow devices, even for devices from the same vendor (see Table IV).

The **expiration check** is another periodical operation that might influence the quality of measured data. As explained in § V-B, the idle timeout can be used to obtain an estimation of the flow duration similarly to NetFlow/IPFIX. This is, however, only possible if the OpenFlow device supports timeouts (which is not the case for the Brocade devices we tested). Even if the timeouts work correctly, the estimation of the flow duration might suffer from variations caused by the expiration check process. As we have shown, these variations are not only vendor-specific but also depend on the workload, making a possible correction very difficult. A very important remark about the flow entry duration is that it can only be used to estimate the actual flow duration if the entry has been inserted to the flow table when the first packet was seen for the flow (*i.e.,* using an add request in response to a packet-in message). Otherwise, the time between the entry installation and the first packet of the flow is added to the duration.

Finally, we also observed **erratic behavior** for some of the tested devices that can impact the quality of measurements. For example, the Brocade CES often reported byte counters set to zero. Another erratic behavior was observed for the Juniper router, where `OFPT_PACKET_IN` messages were sent to the controller even though matching flow entries were already preloaded into the flow table. We believe that these issues are bugs and will likely be fixed in next firmware versions.

VII. Related Work

Although OpenFlow was not proposed as a measurement solution, there are numerous publications and foreseen SDN applications where the statistics provided by an OpenFlow-enabled device are used for network measurements and monitoring (see the surveys listed in § I). The applications themselves are out of the scope of this paper. However, it is crucial to note that, to the best of our knowledge, the quality of the statistics provided by OpenFlow devices is not studied in existing work. For example, [28] proposes a tool named *OpenNetMon* to monitor networks by polling edge switches at an adaptive rate. The (relatively small) differences between results for bandwidth measurements reported by the tool and a packet-based ground truth are explained by binning effects due to the interaction between the counter update frequency of the switch and the polling frequency of the tool. A systematic assessment of the accuracy of the counters is not made.

In the context of network measurement, it is quite natural to compare OpenFlow to NetFlow/IPFIX. Indeed, most of the papers discussed in [12] explicitly refer to NetFlow or IPFIX when discussing the advantages of their OpenFlow-based solutions. Since OpenFlow and NetFlow/IPFIX flow monitoring are implemented in similar ways (using flow tables, active and inactive timeouts, etc.), it is interesting to see how NetFlow/IPFIX-enabled devices behave in terms of accuracy. Due to the extreme popularity of NetFlow, the performance of such devices has been extensively studied by researchers. Problems caused by an insufficient timestamp resolution in the flow metering and exporting process have been analyzed in [29] and [30], and methods have been proposed to mitigate their impact on the measurement accuracy. Artifacts found in flow data from Juniper devices have been studied in [31]. In [32], flow-enabled devices have been compared and various artifacts identified. There are various reasons for such artifacts. For example, some switches have a hardware-switching engine as well as a software-switching engine and, depending on which engine is used to switch a particular flow, different information is available to the flow metering process. Other artifacts are caused by resource constraints: in order to make the export process more efficient, the flow table is only scanned for terminated flows in intervals of several seconds, resulting in deviations from the configured timeout values.

VIII. Conclusions

Many works in the recent literature propose to join traffic measurements with the flexibility of network management both enabled by OpenFlow. These works do not, however, address the quality of measured data one can obtain from real OpenFlow devices, limiting the validation of their respective proposals to simulated and controlled environments.

In this paper we have systematically assessed the quality of flow-level traffic measurements from real OpenFlow devices. Note that instead of finding the best device, our goal was to raise awareness of the difficulties imposed by early and immature implementations of OpenFlow by vendors. These implementations do not fully adhere to OpenFlow specifications and are not consistent among themselves. We have identified many pitfalls with the tested devices that directly or indirectly impact the quality of measured data. One of our major observations is that the inaccuracies and artifacts found are not consistent among devices. That is, different sets of problems are found on different OpenFlow devices, and some even between devices from the same vendor. Network operators should be aware

of the measurement limitations of their OpenFlow devices. Furthermore, the differences between vendor implementations restrict the deployment of a measurement-based application in a multi-vendor OpenFlow-based network.

Nonetheless, SDN and OpenFlow are still relatively new concepts and technologies and, hence, the situation is expected to improve in the foreseeable future. A strong indicator that OpenFlow implementations by vendors are persistently evolving is the large number of new firmware releases in the last few months, containing bug fixes and new features. Furthermore, in addition to ONF, there are lots of efforts on the standardization of SDN and OpenFlow, such as IRTF SDNRG[4] and several documents being published within the IETF community.

Acknowledgements

We thank Sebastian Seeber (UniBW), RNP, ANSP, and RedClara. Ricardo de O. Schmidt, Luuk Hendriks and Aiko Pras' work is partially supported by the EU FP7 Mobile Cloud Networking (#318109), EU FP7 FLAMINGO NoE (ICT-318488). Jeronimo A. Bezerra's work is supported by NSF IRNC-ProNet Americas Lightpaths (ACI-0963053).

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *ACM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[2] R. Wang, D. Butnariu, and J. Rexford, "OpenFlow-Based Server Load Balancing Gone Wild," in *11th USENIX conference on Hot topics in management of Internet, cloud, and enterprise networks and services*, ser. Hot-ICE 2011, 2011, pp. 1–6.

[3] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "OpenTM: Traffic Matrix Estimator for OpenFlow Networks," in *11th International Conference on Passive and Active Measurement*, ser. PAM 2010, 2010, pp. 201–210.

[4] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. V. Madhyastha, "FlowSense: Monitoring Network Utilization with Zero Measurement Cost," in *14th International Conference on Passive and Active Measurement*, ser. PAM 2013, 2013, pp. 31–41.

[5] P. Sun, L. Vanbever, and J. Rexford, "Scalable Programmable Inbound Traffic Engineering," in *ACM SIGCOMM Symposium on SDN Research*, ser. SORS 2015, 2015, pp. 1–7.

[6] P. Sun, M. Yu, M. J. Freedman, J. Rexford, and D. Walker, "HONE: Joint Host-Network Traffic Management in Software-Defined Networks," *Journal of Network and Systems Management*, vol. 23, no. 2, pp. 374–399, 2015.

[7] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. Mckeown, "ElasticTree: Saving Energy in Data Center Networks," in *7th USENIX conference on Networked systems design and implementation*, ser. NSDI 2010, 2010, pp. 1–16.

[8] C. Donato, P. Serrano, A. de la Oliva, A. Banchs, and C. J. Bernardos, "An OpenFlow Architecture for Energy-Aware Traffic Engineering in Mobile Networks," *IEEE Network*, vol. 29, no. 4, pp. 54–60, 2015.

[9] R. Braga, E. Mota, and A. Passito, "Lightweight DDoS Flooding Attack Detection Using NOX/OpenFlow," in *35th IEEE Conference on Local Computer Networks*, ser. LCN 2010, 2010, pp. 408–415.

[10] Y. Wang, Y. Zhang, V. Singh, C. Lumezanu, and G. Jiang, "NetFuse: Short-circuiting Traffic Surges in the Cloud," in *IEEE International Conference on Communications*, ser. ICC 2013, 2013, pp. 3514–3518.

[11] Y. Zhang, "An Adaptive Flow Counting Method for Anomaly Detection in SDN," in *9th ACM conference on Emerging networking experiments and technologies*, ser. CoNEXT 2013, 2013, pp. 25–30.

[12] A. Yassine, H. Rahimi, and S. Shirmohammadi, "Software Defined Network Traffic Measurement: Current Trends and Challenges," *IEEE Instrumentation & Measurement Magazine*, vol. 18, no. 2, pp. 42–50, 2015.

[4]https://irtf.org/sdnrg

[13] B. Lantz, B. Heller, and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-Defined Networks," in *9th ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets 2010, 2010, pp. 1–6.

[14] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "A roadmap for traffic engineering in SDN-OpenFlow networks," *Computer Networks*, vol. 71, pp. 1–30, 2014.

[15] F. Hu, Q. Hao, and K. Bao, "A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2181–2206, 2014.

[16] Y. Jarraya, T. Madi, and M. Debbabi, "A Survey and Layered Taxonomy of Software-Defined Networking," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 1955–1980, 2014.

[17] A. Lara, A. Kolasani, and B. Ramamurthy, "Network Innovation using OpenFlow: A Survey," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 493–512, 2014.

[18] B. A. A. Nunes, M. Mendonça, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[19] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[20] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A Survey on Software-Defined Networking," *IEEE Instrumentation & Measurement Magazine*, vol. 17, no. 1, pp. 27–51, 2015.

[21] M. Yu, L. Jose, and R. Miao, "Software Defined Traffic Measurement with OpenSketch," in *10th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI 2013, 2013, pp. 29–42.

[22] M. Moshref, M. Yu, R. Govindan, and A. Vahdat, "DREAM: Dynamic Resource Allocation for Software-defined Measurement," in *ACM SIGCOMM conference*, 2014, pp. 419–430.

[23] Open Networking Foundation, "OpenFlow Switch Specification – Version 1.0.0," https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf, 2009, online. Accessed Aug. 2015.

[24] ——, "OpenFlow Switch Specification – Version 1.5.0," https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf, 2014, online. Accessed Aug. 2015.

[25] ——, "OpenFlow Switch Specification – Version 1.3.1," https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.1.pdf, 2012, online. Accessed Aug. 2015.

[26] R. de O. Schmidt, A. Sperotto, R. Sadre, and A. Pras, "Towards Bandwidth Estimation using Flow-level Measurements," in *6th International Conference on Autonomous Infrastructure, Management and Security*, ser. AIMS 2012, 2012.

[27] R. de O. Schmidt, L. Hendriks, A. Pras, and R. van der Pol, "OpenFlow-based Link Dimensioning," in *Workshop on Innovating the Network for Data-Intensive Science (INDIS), International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC 2014, 2014.

[28] N. L. M. van Adrichen, C. Doerr, and F. A. Kuipers, "OpenNetMon: Network monitoring in OpenFlow Software-Defined Networks," in *IEEE/IFIP Network Operations and Management Symposium*, ser. NOMS 2014, 2014, pp. 1–8.

[29] B. Trammell, B. Tellenbach, D. Schatzmann, and M. Burkhart, "Peeling Away Timing Error in NetFlow Data," in *12th International Conference on Passive and Active Network Measurement*, ser. PAM 2011, 2011, pp. 194–203.

[30] J. Kögel, "One-way Delay Measurement based on Flow Data: Quantification and Compensation of Errors by Exporter Profiling," in *25th International Conference on Information Networking*, ser. ICOIN 2011, 2011, pp. 25–30.

[31] I. Cunha, F. Silveira, R. Oliveira, R. Teixeira, and C. Diot, "Uncovering Artifacts of Flow Measurement Tools," in *10th International Conference on Passive and Active Network Measurement*, ser. PAM 2009, 2009, pp. 187–196.

[32] R. Hofstede, I. Drago, A. Sperotto, R. Sadre, and A. Pras, "Measurement Artifacts in NetFlow Data," in *14th International Conference on Passive and Active Measurement*, ser. PAM 2013, 2013, pp. 1–10.