RFlow$^+$: An SDN-based WLAN Monitoring and Management Framework

*RhongHo Jang, †DongGyu Cho, ‡Youngtae Noh, and §DaeHun Nyang
The School of Computer and Information Engineering of Inha University, Korea
*jiyoo@seclab.inha.ac.kr, †12121564@inha.edu, ‡ytnoh@inha.ac.kr, §nyang@inha.ac.kr

***Abstract*—In this work, we propose an SDN-based WLAN monitoring and management framework called RFlow$^+$ to address WiFi service dissatisfaction caused by the limited view (lack of scalability) of network traffic monitoring and absence of intelligent and timely network treatments. Existing solutions (e.g., OpenFlow and sFlow) have limited view, no generic flow description, and poor trade-off between measurement accuracy and network overhead depending on the selection of the sampling rate. To resolve these issues, we devise a two-level counting mechanism, namely a distributed local counter (on-site and real-time) and central collector (a summation of local counters). With this, we proposed a highly scalable monitoring and management framework to handle immediate actions based on short-term (e.g., 50 ms) monitoring and eventual actions based on long-term (e.g., 1 month) monitoring. The former uses the local view of each access point (AP), and the latter uses the global view of the collector. Experimental results verify that RFlow$^+$ can achieve high accuracy (less than 5% standard error for short-term and less than 1% for long-term) and fast detection of flows of interest (within 23 ms) with manageable network overhead. We prove the practicality of RFlow$^+$ by showing the effectiveness of a MAC flooding attacker quarantine in a real-world testbed.**

I. Introduction

With the plethora of WLAN deployments in residential and enterprise settings, Internet accessibility has become easier than ever. This proliferation has become even more expedited because of increasing demands from a wide range of user devices, e.g., smartphones and tablet PCs. In order not to lag behind users' aggressive network bandwidth demands in their daily lives (e.g., for YouTube or Netflix), WLAN technologies have rapidly advanced: 802.11n (up to 600 Mbps), 802.11ac (up to 6.933 Gbps), and so on [25]. Interestingly but unfortunately, despite the advancements of WLAN technologies, people are easily dissatisfied with their WLAN infrastructures.

The reasons for this dissatisfaction are two-fold: (1) an absence of intelligent and timely network treatments followed by (2) the limited view of network traffic monitoring tools (e.g., NetFlow [12] and sFlow [24]) and vendor-oriented configurability. Instead of näive over-provisioning of access points (APs), we can provide users with more stable and thus more reliable network conditions (e.g., latency, jitter, and required minimum bandwidth) by accurate network monitoring and timely treatments such as rate-limiting, the access control list (ACL), or flow quarantines.

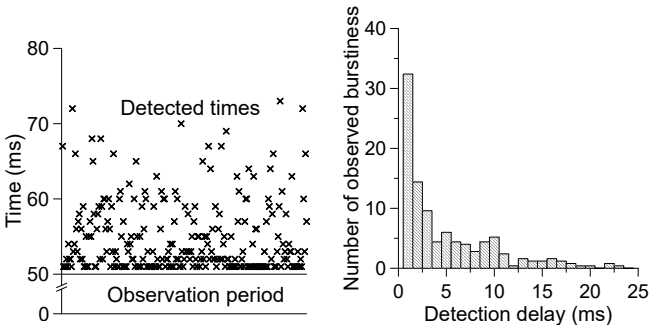
Recently, intense efforts in two main streams have been made to realize the "*victory*" of SDN-driven data centers like B4 [7] in the WAN domain. First, efforts have been made in WLAN management frameworks. Unlike OpenFlow [11], a *de facto* standard interface between a controller and switches, a WLAN management framework requires additional features such as wireless channel selection, interference mitigation, and mobility management. To achieve these, BeHop [27], Odin [21], and OpenSDWN [20] customized OpenFlow's configurability for WLAN by introducing the concept of virtual APs. The other optimization efforts have addressed WLAN monitoring frameworks (e.g., PayLess [3], OpenSketch [29], FlowSense [28], and OpenTM [26]). These monitoring frameworks tried to overcome the intrinsic limitations (i.e., the limited accuracy of default settings and resource-hungry nature of full sampling) of generic sampling based solutions—NetFlow [12] and sFlow [24].

Unlike Ethernet, WLAN requires a network management framework to monitor wireless network traffic at different target levels (i.e., *short-term* bursty users and *long-term* heavy down-loaders/up-loaders) because of its openness. In addition, the management framework needs to improve overall wireless bandwidth utilization by timely resource allocation actions (i.e., *immediate* action according to short-term monitoring results and *eventual* action according to long-term monitoring) as well as accommodate more users by dynamically providing capacity. To our best knowledge, no existing studies have ever included both approaches in its design considerations. To cope with these issues, we propose RFlow$^+$ to achieve two different levels of network monitoring—local (switch/AP level) and global (controller/collector level); thereby supporting application-specific actions (i.e., *immediate* and *eventual*) via a network management framework. The recyclable counter with confinement (RCC) [13] motivates RFlow$^+$'s major design as it provides reliable counting accuracy while efficiently managing its memory usage; this consequently reduces network overheads, as further detailed in Section II.

Mainly because of our two-level (i.e., global and local) monitoring framework design based on the RCC counter, RFlow$^+$'s major departure from existing work is that the *local agent* takes the first step toward supporting immediate actions (e.g., flow rate-limiting or flow quarantines), which can be flexibly managed by users/operators' high-level descriptions (see Section III-A). Concretely, this paper makes the following contributions:

- We proposed RFlow$^+$, a novel monitoring and management framework for WLAN, to support both *short-term* and *long-term* monitoring applications and enforce timely

978-1-5090-5336-0/17/$31.00 ©2017 IEEE



(a) Distribution of detected times (b) Distribution of delay

Fig. 1. Distribution of burstiness detections with RFlow$^+$ (250 trials)

treatments (i.e., rate-limiting and flow quarantines) based on their requirements (i.e., *immediate* and *eventual*).

- We proposed a distributed packet counting algorithm for heavy user detection. The central collector holds a global view on the traffic measurement, while each on-site AP has its own local view for connected user measurement. The counting algorithm performs short-term measurement (e.g., 50 ms time window) locally as well as long-term measurement (e.g., 1 month) globally.
- We prototyped RFlow$^+$ on top of OpenWrt on off-the-shelf access point hardware (TP-Link AC1750) as add-ons on OpenVSwitch (OVS) [17] and OpenDaylight [14]. The implemented RFlow$^+$ code is available upon request.
- To compare RFlow$^+$ with native OpenFlow and a sampling-based monitoring solution (sFlow), we performed real-world experiments by deploying our APs on our university campus.

II. Motivation

In this section, we discuss why in-network monitoring tools need to support two different types of flow measurements—local (switch/AP level) and global (controller/collector level)—based on their timelines and why existing flow monitoring solutions cannot fulfill this because of their inherent structural limitations. Consequently, we introduce a novel monitoring and management framework called RFlow$^+$ that fits such requirements and further supports timely treatments (i.e., executing predefined immediate action rules) . There are two different monitoring applications, namely long-term (e.g., heavy down/up-loaders and ISP billing) and short-term (e.g., bursty traffic [9], [19] and MAC flooding).

Long-term monitoring: To rate-limit heavy down/up-loaders or for ISP billing, we might need to measure in-network flows for a long period (say, a week or month). For accuracy and consistency, these measurements should be performed in a global manner. That is, aggregate statistics from every AP should be used for the final decision.

NetFlow and sFlow have been widely adopted to monitor wired and wireless in-networks. However, these sampling based monitoring solutions leads to low accuracy with per-flow counting (even missing transient flows), while increasing the sampling rate requires too many resources (e.g., CPU, memory, and network bandwidth).

With its advent, SDN technology provided control of the data plane from the controller by adding/removing forwarding rules in the range of Layers 2–4 and further provided resource-efficient statistics at different aggregation levels (e.g., `flow`, `port`, and `table`). These statistics are collected from OpenFlow enabled switches or OVS via the OpenFlow (hereafter, native-OF). The native-OF not only counts packets and bytes per-flow, it also provides an aggregated view of the statistics of `table` and `port`. In reality, however, the higher the layer at which the flow is defined, the more flows will be generated. When the SDN controller periodically sends an `OF_FLOW_STAT_REQUEST` message to the OpenFlow switch/OVS, the `OF_FLOW_STAT_REPLY` message that contains entire flow entries of userspace flow tables should be sent back to the controller, which is the first weak point of native-OF: *lack of scalability*. Second, native-OF's flow definition is *non-generic*, so it cannot generically handle undefined situations (i.e., flows). For OpenFlow, an application requires statistics on some specific layer 4 flows while flows of interest are not determined before the request. To reply to the request, the flows should previously be defined with all possible combinations of Layers 2–4 addresses and flows of interest should be exhaustively searched over the entire table. This will cause significant CPU[1]/network overheads (See Section V-A).

To monitor flows efficiently and in a timely manner (one of RFlow$^+$'s goals), we adopt RCC, an approximate counter. Nyang *et al.* introduced RCC in [13] as a low-memory-cost approximate packet counter designed for large-scale and real-time per-flow measurement in a high-speed router. RCC achieved its high accuracy (approximately 99%) using a small-but-recyclable virtual vector and obtained high speed access by confining virtual vectors within a CPU word instead of spreading them over the entire memory. Additionally, RCC provides two desirable features: (1) RCC decoding can be performed in real time (about two hash computations per decoding operation) and (2) RCC provides the top-K list.

RCC inspired RFlow$^+$, which resolves the two formerly mentioned issues of native-OF: unscalablility and non-generic implementation. RFlow$^+$ allows the OVS to define the minimum number of flows; thereby minimizing network overheads caused by statistical reports. Additionally, it provides generic statistics on every possible flow without requiring a long list of flow definitions thanks to RCC. As RCC can track the estimated packet counts with high precision, RFlow$^+$ at a switch reports only non-zero entries in *elephant-flow* counter to a central collector, as those flows are active (i.e., regions of interest) for a statistical collection period (e.g., a default setting of 3 s in native-OF). By doing so, RFlow$^+$ can reduce network overheads as well as achieve memory efficiency. Of course, locally-made microscopic statistics (i.e., *mice-flow* counters)

[1]Giotis *et al.* proved that CPU overhead is also caused by high level flow definition [6]; thus defining a larger number of flows at high levels finally leads to additional overheads in both CPU and network resources.

TABLE I
COMPARISON - RFLOW$^+$ WITH NATIVE-OF AND SFLOW REGARDING PERFORMANCE OF LOCAL SHORT-TERM MONITORING (LSM) AND GLOBAL LONG-TERM MONITORING (GLM) AND THEIR COSTS

	Counting type	Native-OF (Exact)	RFlow$^+$(Approximate)	sFlow (Sampling)
LSM	Memory	Proportional to the number of defined flows	2 MB for more than 100K flows (including for long-term detection)	N/A
	CPU	Proportional to the number of defined flows	Proportional to the number of active flows $\geq$ 1 hash ($\approx$ 30 $clocks$) for per-flow detection	N/A
	Accuracy	Exact	Standard error$\leq$ 5%	N/A
	Responsiveness	Not supported	Real-time	Not supported
	Implementation	Needs data structure redesign	Provided (200 lines of code)	Needs system re-design
GLM	Accuracy	Exact	Standard error $\leq$ 1%	6% or higher sampling rate for standard error $\leq$ 2%
	Network overhead	Proportional to the number of defined flows	Proportional to the number of active flows over a collection period	According to the polling interval and the sampling rate
	Implementation	Needs additional memory	Provided (200 lines of code)	Provided

can be sent to a central collector on demand or periodically. Note that the size of overall statistical message exchanges over the network is significantly low (see Section V). Finally, RFlow$^+$ performs *eventual* (not *immediate*) actions (e.g., limiting or blocking monthly heavy down/up-loaders, advanced persistent threat attackers, or slow scanning attackers and ISP billing) based on the collected long-term statistics.

Short-term monitoring: As well as long-lived flows, in-network overall flows contain short-lived flows. Unlike long-lived flows, short-lived ones are transient. Thus, the flow measurements and their treatments should take place on site and in real-time. Short-term bursty traffic causes other users in the same network to experience degraded network performance or intermittent disconnections. Sarvotham *et al.* reported that bursts are not caused by a "conspiracy" of many moderate flows, but rather by a few dominating connections (i.e., alpha traffic [19]). Thus, it is beneficial to immediately limit the dominating connections/flows to avoid saturating the link. As this type of activity is very short-term, it is hard (or impossible) to detect it with existing monitoring frameworks. Although detected, its treatment can be *post-mortem*. To remedy this, the major goals of RFlow$^+$ are (1) to design a network monitoring system that can detect these transient events, and (2) to provide a local flow regulation in an instantaneous manner. Here, the "+" in RFlow$^+$ means that RFlow$^+$ can apply an immediate action on switch in a pre-defined manner—for rate-limiting, flow quarantines, and other actions.

To correctly identify the dominating flows and limit them accordingly, reliable burstiness detection must first be implemented. To our best knowledge, none of the existing monitoring solutions can detect burstiness in real time. Lan *et al.* proposed three definitions of burstiness in an offline measurement study [9], namely variance, round trip time (RTT), and train burstiness. As RTT burstiness is not detectable because of the existence of uni-directional flows, RFlow$^+$ adopts the variance definition of burstiness, as our burstiness decision should be done on-the-fly and its reported accuracy is qualitatively similar to train burstiness. Variation burstiness is based on the variation of traffic at a time-scale of T[2]. Given a flow, it is divided into bins b_i and the number of bytes sent in b_i is defined as s_i. The variance burstiness of the flow is then defined as the standard deviation of all s_i. Fig. 1 shows the distribution of burstiness detections obtained by RFlow$^+$. Among 250 trials with $T = 50$ ms, RFlow$^+$ provides a 100% detection ratio within 23 ms delay.

[2]In our settings, we set $T = 50$ ms as in [19] and ignore flows shorter than T, as their variance is undefined. To avoid errors from boundary effects, we also ignore flows shorter than 3–5 T.

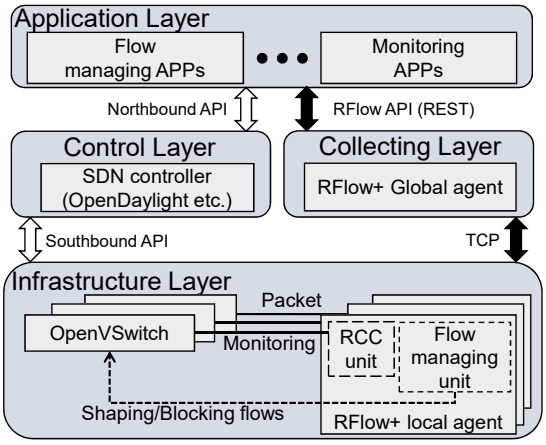


Fig. 2. Architecture of RFlow$^+$

Table I presents a summary of how RFlow+ provides a good tradeoff between accuracy and cost savings and supports both short-term and long-term applications. As mentioned earlier, none of the existing monitoring tools can detect transient network anomalies nor regulate them in real time.

III. RFLOW$^+$ FRAMEWORK DESIGN

As shown in Fig. 2, RFlow$^+$ extends a general SDN framework with a RFlow$^+$ global agent in the collecting layer and RFlow$^+$ local agent in the infrastructure layer. The application layer interacts with the SDN controller in the control layer and RFlow$^+$ global agent in the collecting layer to provide flow management and monitoring, respectively, for user billing, security functions, heavy user detection, and quality of service (QoS). The SDN controller also retrieves statistics collected from the OVS via the northbound API. OpenDaylight, a popular SDN controller, resides in the control layer. It provides flow management and statistics collection APIs for northbound applications. Through the southbound

API, the SDN controller sends control/data plane messages and requests for statistics to the OVS.

The RFlow$^+$ global agent is located in the collecting layer to store the statistics received from the RFlow$^+$ local agents periodically and obtain overall statistics (including macroscopic statistics) on demand. It provides a RESTful API for northbound applications to access statistics or to propagate predefined immediate action rules to the OVS. Finally, RFlow$^+$ local agent is located in the infrastructure layer for per-flow packet counting and executing predefined immediate action rules populated by application layer. Note that flows in RFlow$^+$ are not the same flows defined in native-OF, but flows defined in Layers 2-4, as shown in Fig. 3.

A. RFlow$^+$ local agent

RFlow$^+$'s local agent is composed of an RCC unit for measuring per-flow traffic and a flow managing unit for applying and executing immediate action rules.

RCC unit: The RCC unit is associated with the OVS for monitoring packets and updating statistics in different layers (Layers 2–4) in parallel. Thus, it requires three RCC counter pairs (i.e., RCC counters A and B) called RCC-L2, RCC-L3, and RCC-L4. Once the RFlow$^+$ local agent connects to the collecting layer through transmission control protocol (TCP), non-zero entries in counter B of RCC-L2, RCC-L3, and RCC-L4 mean that those flows are active for a statistics collection period. Those entries are accumulated in the global table in the global agent periodically using JSON, as Fig. 3 shows. Then, the RCC unit resets the packet counts of the entries to zero in counter B, as if we updated only the active flows' statistics. NodeID is the identifier of each OVS that was allocated by the SDN controller. RFlow$^+$ also provides an option to enable/disable measurement in each network layer.

Flow managing unit: This module is responsible for executing the predefined immediate action rules according to the statistics measured by the RCC unit. The immediate action rules are predefined by northbound applications and populated via APIs provided in the RFlow$^+$ global agent. There are various ways to limit the rate of a flow: deleting the flow, setting flow actions with drop, or redirecting the flow to bandwidth-limited paths. Of course, we can limit flow rates via Queuing Disciplines (`qdisc`) by the kernel. However, the current OpenWrt [15] implementation (the OVS runs on top of it) does not fully support this.

B. RFlow$^+$ global agent

The RFlow$^+$ global agent resides in collecting layer to store the statistics obtained from local agents, as shown in Fig. 4. Northbound applications can further collect statistics from the RFlow$^+$ global agent and can propagate immediate action rules (i.e., high-level descriptions) to the OVS through the RFlow$^+$ RESTful API. The RFlow$^+$ global agent consists of the following five modules:

1. Global counter table: The global counter table is a persistent storage that has the same structure as the counter B's in the RCC unit and is distinguished by switch NodeID. The structure is shown in Fig. 3.

2. Node selector: This supports customized statistics at switch level. Users can choose for single, multiple, or all nodes to collect statistics, as shown in Fig. 5. In addition, the node selector can interact with the other modules of the global agent to obtain combined statistics.

3. Flow selector: This module is responsible for aggregating flow statistics from the global counter table. By giving a partial flow definition, high level primitives also can be collected (e.g., TCP port 2424). The flow selector module can interact with the node selector module to collect statistics from specific switches.

4. Layers 2–4 Aggregator: This module aggregates statistics amongst different layers. For example, the layer 2 (L2) Aggregator may want to aggregate statistics from the MAC-level, L3 at IP-level, and L4 at Port-level. By selecting different combinations, the global agent can produce customized statistics to satisfy different applications' demands.

5. Immediate Action Rule Listener: This module is responsible for listening for predefined immediate action rules (from northbound applications) via RFlow$^+$ APIs and forwarding the rules to a flow management unit located in the RFlow$^+$ local agent; thereafter enforcing the rules on the switch.

C. RFlow$^+$ RESTful API

RFlow$^+$ provides a RESTful API for northbound applications to access statistics or populate switches with predefined immediate action rules. Every network application needs to create a JSON object called `RFlow+Request` (see Fig. 5) to customize statistics according to their purpose and define immediate action rules. `RFlow+Request` contains the following:

- Type: The network application needs to define what type of operations it wants (e.g., retrieval of long-term statistics) and define immediate action rules.
- Node: This designates the set of switches to be managed.
- AggregationLevel: The network application enables execution on a specific layer defined in the Type field. For example, if Type is "statistics," the entire table entries will be returned according to the selected layer (e.g., RCC-L2, RCC-L3, or RCC-L4). If Type is "manage," immediate action rules for specified layers will be defined.
- Flow: This expresses a specific flow for statistics collection. The seven variables used to express a flow are `MAC_src`, `MAC_dst`, `IP_src`, `IP_dst`, `Proto`, `Port_src`, and `Port_dst`. Each variable is set according to the selected AggregationLevel. For the "statistics" type, an application can specify the partial (or entire) flow definition for accepting high level primitives.
- StatFilter: This field is available for a "statistics" type request to filter statistics with parameters (e.g., `Top_N` and `condition`).
- ManageRules: This field is available for a "manage" type to express an immediate action rule with parameters: `time_window`, `threshold`, and other parameters.

{"TableUpdate": {
"Node": " ["$NodeID$"] ",
"L2": " ["$MAC_{src\_dst\_1}$", "est_1", … , "$MAC_{src\_dst\_N}$", "est_N"] ",
"L3": " ["$MAC_{sr\ \_dst\_1}$", "$IP_{src\_dst\_1}$", "$Proto_1$", "est_1", … , "$MAC_{src\_dst\_N}$", "$IP_{src\_dst\_N}$", "$Proto_N$", "est_N"] ",
"L4": " ["$MAC_{src\_dst\_1}$", "$IP_{src\_dst\_1}$", "$Proto_1$", "$Port_{src\_dst\_1}$", "est_1", … , "$MAC_{src\_dst\_N}$", "$IP_{src\_dst\_N}$", "$Proto_N$", "$Port_{src\_dst\_N}$", "est_N"] "}}

Fig. 3. TableUpdate object: Table indices are sent instead of flow definitions after the first accumulation to reduce network traffic.

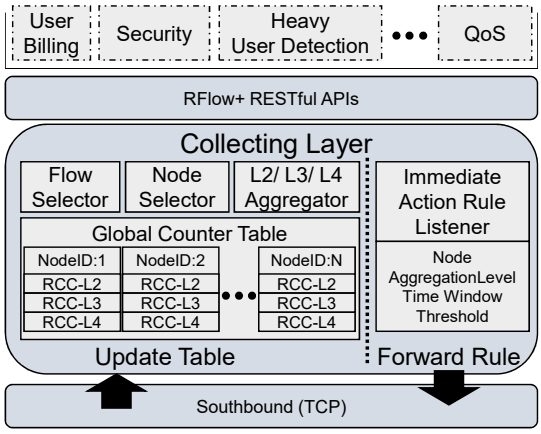


Fig. 4. Internals of RFlow$^+$ global agent

{"RFlow+Request": {
"Type": " ["statistics" | "manage"] ",
"Node": " ["$nodeID_1$", "$nodeID_2$", "$nodeID_3$", "…" | "ALL"] ",
"AggregationLevel": " ["$L2$", "$L3$", "$L4$" | "ALL"] ",
"Flow": " ["MAC_src": "MAC_addr_1", "MAC_dst": "MAC_addr_2" |
"IP_src": "IP_addr_1",
"IP_dst": "IP_addr_2",
"Proto": "$protocol$",
"Port_src" : "$port_1$",
"Port_dst" : "$port_2$"] "
"StatFilter": " ["Top_N": "top_N" | "Condition": "$condition$"] "
"ManageRule": " ["TimeWindow": "$time\_window$", "Threshold": "$threshold$", etc.] "}}

Fig. 5. `RFlow+Request` object

D. Algorithms

1) RFlow$^+$ local agent: As shown in Algorithm 1, the RFlow$^+$ local agent consists of four functions: *PacketMonitor, TableUpdate, FlowManagingRuleReceiver, and FlowManagingEventLisener*. All of them are non-blocking input/output functions that run in different threads.

`TcpConnect` creates TCP connection between the RFlow$^+$ global agent to a switch with `IP:PORT` and the switch sends its `NodeID`. After a connection is created, *PacketMonitor* starts the iterative monitoring process with the user-inputted `Interface` to capture its packets. When a packet is captured, it reads information in the `PacketHeader` of different layers: `MAC, IP, Proto, Port`. The extracted information is passed to `RCC_unit` by calling `Send_RCC_unit` for counting, which is described in Algorithm 2.

TableUpdate of the RFlow$^+$ local agent accumulates active counters in local tables and sends the updates to the RFlow$^+$ global counter table (located in the global agent) every $PollingInterval$ s. `TcpSend` sends active local counters in JSON format (shown in Fig. 3). *FlowManagingRuleReceiver* receives immediate action rules from the RFlow$^+$ global agent via the existing TCP connection. Once the rule is received, flow managing parameters (e.g., `AggregationLevel, TimeWindow`, and `Threshold`) will be forwarded to the RCC unit by calling `Send_RCC_unit`. The *FlowManagingEventListener* is a function provided to `RCC_unit`. The flow definition is passed as a parameter for flow management (i.e., immediate action rules) on the OVS by a system call.

2) RCC unit: As shown in Algorithm 2, the RCC unit provides approximate counting. `RCC_unit` first extracts $\texttt{packet}_{\texttt{Info}}$ according to `AggregationLevel`, and then it finds `VirtualVector` in `counterA` using `hash` for counting the flow. This procedure is executed repeatedly until one of the `VirtualVector` saturates. The estimated number (`est`) of saturated `VirtualVectors` is accumulated in the `est` field of `FlowEntry` which is a component in `counterB`. Counter values in `est` field are used to make the statistics in the RFlow$^+$ global agent.

3) Use case (short-term heavy user detection): Algorithm 2 also shows how an immediate rule is executed on a local agent (line 11). We use additional field $\texttt{est}_{\texttt{short}}$ to accumulate `est` for short-term (`TimeWindow`) measurement. After each accumulation, `RCC_unit` checks whether `timeWindow` has expired from the most recently detected time (`FlowEntry.Time`). When `timeWindow` is expired, it recalculates $\texttt{est}_{\texttt{short}}$ in proportion to `timeWindow` and compares it with `Threshold` to determine if the flow is heavy. The detected flow information is sent to *FlowManagingEventListener* for further action. `RCC_unit` then resets $\texttt{FlowEntry.est}_{\texttt{short}}$ and updates `FlowEntry.Time` with the current time for the next round of detection. The distribution of detected times in Fig. 2 is taken from this use case.

Algorithm 1 RFlow$^+$ local agent

Require: `Interface, PollingInterval, IP, PORT, NodeID`
1: $tcp \leftarrow$ TcpConnect($\texttt{IP}_{\texttt{RFlow+\_global\_agent}}$: PORT)
2: tcpSend(tcp, NodeID)
3: /* RCC unit */
4: PacketMonitor(Interface){
5: PacketHeader ← Monitoring(Interface)
6: $\texttt{packet}_{\texttt{Info}} \leftarrow$ GetL2L3L4$_{\texttt{Info}}$(PacketHeader)
7: Send_RCC_unit($\texttt{packet}_{\texttt{Info}}$)
8: }
9: /* TableUpdate */
10: TableUpdate(tcp, PollingInterval){
11: **IF** PollingInterval $expired$ **THEN**
12: TcpSend(tcp, TableUpdate($Fig.\ 3$))
13: }
14: /* Flow managing unit */
15: FlowManagingRuleReceiver(tcp){
16: Limit_rule ← TcpReceive(tcp)
17: AggregationLevel ← GetLevel(Limit_rule)
18: TimeWindow, Threshold ← GetRule(Limit_rule)
19: Send_RCC_unit(AggregationLevel, TimeWindow, Threshold)
20: }
21: /* Called by RCC_unit()) */
22: FlowManagingEventListener($\texttt{packet}_{\texttt{Info}}$){
23: ManageFlow($\texttt{packet}_{\texttt{Info}}$)
24: }

Algorithm 2 RCC unit

Require: $\texttt{packet}_{\texttt{Info}}$, AggregationLevel, TimeWindow, Threshold
1: /* Approximate counting */
2: $\texttt{packet}_{\texttt{Info}} \leftarrow \texttt{Get\_packet}_{\texttt{Info}}(\texttt{AggregationLevel})$
3: $hash \leftarrow \texttt{Hash}(\texttt{packet}_{\texttt{Info}})$
4: $\texttt{VirtualVector} \leftarrow \texttt{GetVirtualVector}(\texttt{CounterA}, hash)$
5: RCCCounting(VirtualVector)
6: **if** VirtualVector $saturation$ **then**
7: $est \leftarrow$ Destimation(VirtualVector)
8: $\texttt{FlowEntry} \leftarrow \texttt{CounterB}(hash, \texttt{packet}_{\texttt{Info}})$
9: $\texttt{FlowEntry}.est+ = est$
10:
11: /* Short-term heavy user detection */
12: $\texttt{FlowEntry}.est_{short}+ = est$
13: **if** $\texttt{CurrentTime}() - \texttt{FlowEntry}.Time \geq \texttt{TimeWindow}$ **then**
14: $est_{short} \leftarrow \dfrac{\texttt{FlowEntry}.est_{short} \times \texttt{TimeWindow}}{\texttt{CurrentTime}() - \texttt{FlowEntry}.Time}$
15:
16: /* Flow managing */
17: **if** $est_{short} \geq$ Threshold **then**
18: FlowManagingEventListener($\texttt{packet}_{\texttt{Info}}$)
19: **end if**
20: $\texttt{FlowEntry}.est_{short} = 0$
21: $\texttt{FlowEntry}.Time =$ CurrentTime()
22: **end if**
23: recycle(VirtualVector)
24: **end if**

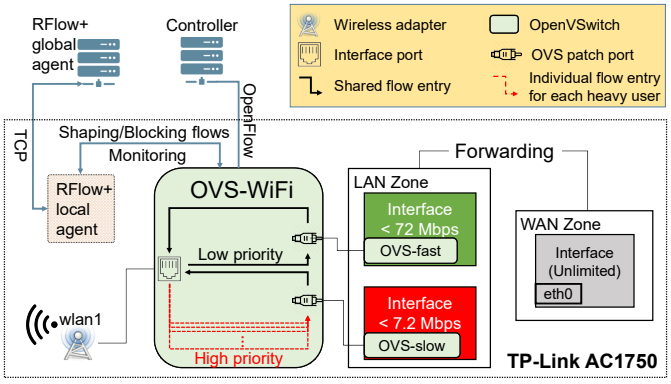


Fig. 6. Testbed layout: an AP with RFlow$^+$ global agent and controller

IV. Implementation

To show the feasibility of RFlow$^+$, we prototyped an SDN-based WLAN monitoring and management system and implemented a use case: monitoring and limiting short-term and long-term heavy users, as shown in Fig. 6.

A. Testbed description

As shown in Fig. 6, we constructed our own testbed with three OpenVSwitches (i.e., OVS-WiFi, OVS-fast, and OVS-slow) in off-the-shelf AP hardware (TP-Link AC1750 ver.2), which ran OpenVSwtich (2.3.90) on top of OpenWrt (15.05, Chaos Calmer). Basically, OVS-fast and OVS-slow worked as gateway interfaces in the LAN Zone and obtained Internet access from the WAN Zone. The bandwidth of these two interfaces were configured with different rates (OVS-fast at 72 Mbps and OVS-slow at 7.2 Mbps) using a QoS software called `qos-script` [16], and they were both connected to the OVS-WiFi. To provide Internet access, the wireless interface `wlan1` was connected to the OVS-WiFi. Overall, the OVS-WiFi works as a central switch that communicates with an SDN controller and forwards packets referring to the flow definition in the flow tables. In the control layer, we used the OpenDaylight (Helium-SR4) as an SDN controller.

B. Monitoring scenario

1) Monitoring with native-OF: To monitor each client with the native-OF, we needed to define two flows for each user with his/her MAC address. For a normal user, two flows (i.e., both directions) were established between `wlan1` and OVS-fast. When the normal user becomes a heavy user, we used a northbound application to change the interface of the heavy user to OVS-slow on both flows.

2) Monitoring with RFlow$^+$: As many flows as devices (MAC per flow) cause a heavy burden on the OVS (i.e., the AP in our setting). Recent WLAN measurement studies reported the number of devices per AP at a conference (110 devices for four days [18]) and public hot-spots (673.30 devices for a month [5]). A global-scale measurement study also reported that the number of devices per network (including residential networks) is 269.90 for a week [2]. Accordingly, the generation of a large number of flows (at least two flows per device) and customized flow definitions (hundreds of combinations among Layers 2–4) exacerbate the load of the OVS.

To better cope with this, we persistently defined three shared flows at the OVS port level (black solid lines in the OVS-WiFi). As described in Section III, RFlow$^+$ can monitor all packets (on the shared three flows only) passing through the OVS in the RFlow$^+$ local agent and sends statistics to the RFlow$^+$ global agent. Therefore, a flow managing application northbound of the controller can obtain specific statistics through RFlow$^+$ API to obtain a list of heavy users. To rate-limit heavy users, the northbound application adds MAC-level individual flows (red dash lines in the OVS-WiFi) for each heavy user. The individual flow for a heavy user has a relatively higher priority than the shared flow. Thus, a heavy user obtains an IP address and accesses the Internet from the OVS-slow gateway with limited bandwidth (7.2 Mbps instead of 72 Mbps). The flow from OVS-slow to `wlan1` could be bidirectional because it can be shared at the OVS port level.

C. Heavy user detection

Heavy user detection is classified into two types according to the detection period: short-term or long-term. The time frame for a long-term heavy user might be a day, week, month, or longer, and that for the short-term user might be 500 ms, 50 ms, or less.

Long-term heavy user detection: Obviously, the time frame for a given quota should be longer than the statistics update period to the RFlow$^+$ global agent (in case of OpendayLight, a default of 3 s). In addition, the threshold for classifying a heavy user should be large enough so that normal users should not easily reach the limit in a given term (false positives). Even though both RFlow$^+$ and native-OF could be used to detect long-term heavy users at server-side for performing eventual actions (usually policies), native-OF costs additional storage for accumulating statistics and significant network overheads

caused by substantially larger number of flows (compared to RFlow$^+$) and customized flows.

Short-term heavy user detection: The time frames for the short-term detections (say, burstiness or MAC flooding) should be shorter than periodic statistics updates; otherwise, the detections may fail to deliver the statistics to the northbound application in time for it to perform immediate actions/treatments. Therefore, it is desirable to measure the short-term heavy user locally (i.e., *on-site*) and execute immediate actions according to predefined rules. The RCC provides a good real-time estimation performance for detecting a short-term heavy user using a very small amount of memory. Using RCC, RFlow$^+$ is designed to execute locally the predefined immediate action rules in the RFlow$^+$ local agent.

D. Parameters

As shown in Nyang *et al.*'s work [13], 8 to 32 bits are sufficient for the virtual vector size of RCC. For RFlow$^+$, we use a unit of 8-bit virtual vectors, as this value obtained the best counting accuracy. We further confined the writing space of virtual vectors as a 32-bit word, which was the CPU word size of the testing devices. In an RCC unit, we allocated 2 Mbytes for each RCC counter (i.e., RCC-L2, RCC-L3, and RCC-L4). In each counter, we allocated 4 Mbits memory for counter A and 12 Mbits for counter B. Thus, in total, 48 Mbits (= 6 Mbytes) of memory were allocated to the RCC unit.

V. Evaluation

A. Network overhead

In the SDN environment, a controller sends `OF_FLOW_STAT_REQUEST` messages periodically to the APs (OVSs) for the collection of per-flow statistics. Then, each OVS packs the statistics and the definition of flow entries, which is defined in the userspace flow tables, and generates an `OF_FLOW_STAT_REPLY` message for replying. For the experiment, the statistics updating period was set to 3 s, and the amount of traffic generated by `OF_FLOW_STAT_REQUEST` and `UpdateTable` was measured for 2 min in our testbed. Fig. **??**fig-overhead shows the size of the `OF_FLOW_STAT_REPLY` messages and the number of TCP segments according to increments in the flow entry number defined in the flow tables. When 100 flows were defined in the OVS flow table, the size of an `OF_FLOW_STAT_REPLY` message was 9.78 KB and it was segmented into three TCP packets.

For 2,000 flows, the size was increased to 191 KB and 57 packets. In reality, more than 57 segmented packets were sent owing to the packet loss, and ACKs were also transferred to the collector in a collecting period. We claim that assuming 2,000 (or more) flows is reasonable, even in a small scale environment (e.g., wireless LAN) because defining flows in the IP layer or higher, like `IP&PORT`, is necessary for various applications (e.g., billing or traffic analysis in various layers). In contrast, with the same rate of periodic statistics updates, RFlow$^+$'s message size was not proportional to the number of flows, as only active flows in a collecting period are packed into an update message.

Comparison with native-OF: Fig. 8 was obtained by conducting experiment in our testbed. When there were 250 clients (500 flows for both directions) in the network, the amount of traffic was the same irrespective of the liveness of the flows. Because RFlow$^+$ updates only the flow statistics that have been changed in an updating period, the amount of traffic was only in proportion to the number of active users, which was much less than that of native-OF. The amount of traffic required for native-OF is 64.4 times more than that of RFlow$^+$ with 10 active clients, and 7.68 times more for 250 active clients.

Comparison with sFlow: Fig. 9 shows RFlow$^+$'s and sFlow's standard error when the number of flows and sampling rate are varied. The dataset used for this experiment was a 1 min network trace from CAIDA. The traffic was collected at the Equinix Chicago data center from 13:10–13:11 on the November 21, 2013 [1]. Obviously, to get higher accuracy, a larger sample is needed, as confirmed in the figure. In addition, for sFlow to achieve an accuracy that is comparable with RFlow$^+$, the sampling rate should be at least 1/16 in Fig. 9, but Fig. 8 shows that the network overhead of sFlow with a 1/16 sampling rate is significantly higher than that of RFlow$^+$. The amount of traffic required for sFlow is 276.2 times more than that of RFlow$^+$ with 10 active clients, and 33.1 times more for 250 active clients.

B. Accuracy of RFlow$^+$

1) Short-term measurement: To test the accuracy of RFlow$^+$ for a 50 ms period, we played a 3 min video on YouTube at 4K quality to generate traffic. The estimated and actual packet numbers for each 50 ms were compared. Fig. 10(a) shows the estimated number (Y-axis) collected by RFlow$^+$ as a function of ground-truth (actual packet number). The closer a point is to the guideline y = x, the more accurate the estimation is. Standard error for the short-term measurement by RFlow$^+$ is 5% in the estimation range of 0–1,000 packets, which means that the measurement is underestimated or overestimated only by 25 packets for a 500 packet flow. As far as we know, there is no other monitoring system that provides this level of accuracy for short-term monitoring.

2) Long-term measurement: To evaluate RFlow$^+$'s accuracy for long term monitoring, we installed RFlow$^+$ on 10 off-the-shelf TP-Link APs and deployed them on our campus. The 10 AP provided free Internet service for students on campus during summer vacation. Fig. 10(b) shows the estimated number (Y-axis) collected by RFlow$^+$ for a week. For the long-term measurement, we compared the packet number estimated by RFlow$^+$with the ground-truth. As shown in the figure, the estimations lie on the guideline y = x, which verifies that RFlow$^+$ provides high precision for long-term monitoring. The standard error for the long-term measurement by RFlow$^+$ is around 1% while consuming an extremely small amount of network overhead.

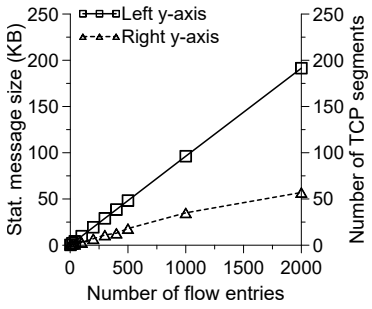


Fig. 7. Native-OF network overhead

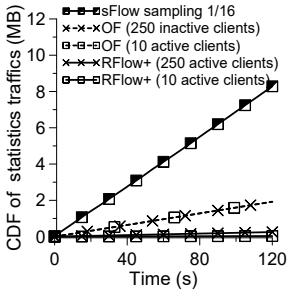


Fig. 8. Network overhead

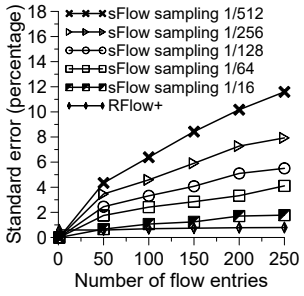
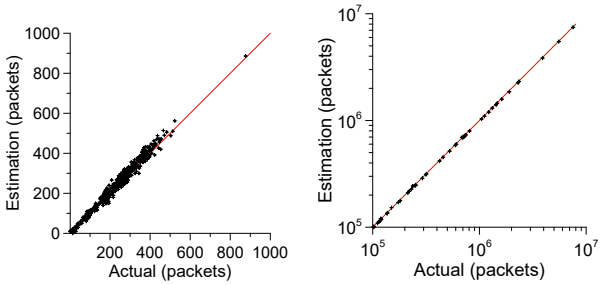


Fig. 9. Counting accuracy



(a) Measurements for 50 ms (b) Measurements for a week

Fig. 10. Estimation accuracy of RFlow$^+$

C. Effectiveness of the MAC flooding attacker quarantine

To test the effectiveness of the short-term monitoring and enforcement of predefined immediate action in the local agent, we artificially created two normal users who sent user datagram protocol (UDP) packets and consumed about 15–25 Mbps of bandwidth, and one attacker with `macof` [10], who sent randomly generated UDP packets and consumed more than 35 Mbps of bandwidth to cause MAC flooding by filling in the AP's content addressable memory (CAM) table and neutralizing its MAC learning. In Fig. 12(a), native-OF shows that normal users experienced significant throughput degradation owing to the traffic bursts caused by the attacker. Before 25 s, only the normal users send packets in a low-fluctuating bandwidth, but right after 25 s, the attacker starts to send a huge number of packets. This bandwidth hogging creates a heavy load over the router's saturation bandwidth. As a result, the normal users' bandwidth started to fluctuate severely.

In Fig. 12(b), however, the RFlow$^+$ local agent continues to monitor every flow and penalized the attacker hogging bandwidth with a quarantine. When the RFlow$^+$ local agent detects that the trial attacker exceeds a pre-defined threshold, the agent quarantines the attacker's flow to suppress ruthless sending so that the normal users can recover from the degraded bandwidth utilization (returning to normal).

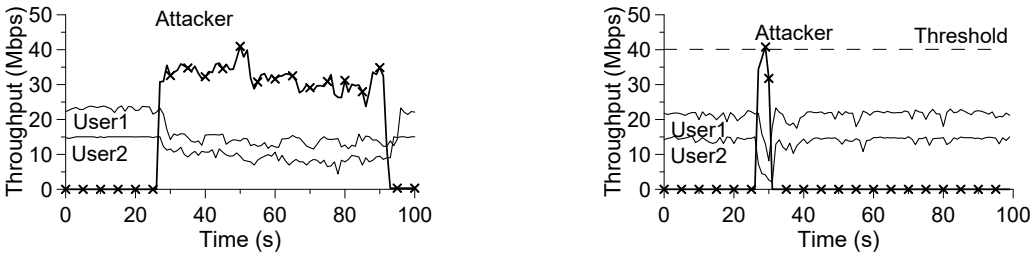
VI. Related Work

RFlow$^+$ falls into two fields of SDN-based frameworks, namely management and monitoring.

SDN-based WLAN management frameworks: In broadband access networks, bandwidth allocation from competing flows can be an important problem, as it degrades the overall network performance. One reliable solution is bandwidth allocation based on the application type. To achieve this, Seddiki *et al.* proposed FlowQoS, which contains two modules, namely a flow classifier and an SDN-based rate limiter [23]. FlowQoS, implemented on top of OpenWrt, demonstrates enhanced performance for both adaptive video streaming and VoIP in home settings in the presence of active competing traffic. However, this work has a limited view of the in-network traffic monitoring compared to RFlow$^+$ and its offloading technique for traffic classification on the controller is inherently limited to short-term monitoring applications.

SDN-based WLAN monitoring frameworks: Because of their generic support for different measurement tasks, NetFlow and sFlow have been widely adopted to monitor wired and wireless in-networks. However, these sampling based monitoring tools cannot accurately report counts per flow. To better cope with this, a fair number of network monitoring tools based on OpenFlow have been proposed recently. OpenTM uses OpenFlow's built-in per-flow statistics reported from OpenFlow switches to directly and accurately measure the traffic matrix with a low overhead. OpenTM exploits the routing information obtained from the controller to intelligently choose flow statistics of interest; thereby reducing the load on switching elements [26]. Yu *et al.* proposed FlowSense [28], a push-based monitoring tool that exploits passive (not on-demand) update messages sent by OpenFlow switches to the controller to inform it of in-network changes (e.g., `PacketIn` or `FlowRemoved` messages) and to efficiently infer link utilization in flow-based networks. However, these solutions are limited to transient traffic behaviors (i.e., burst traffic), as they did not consider short-term monitoring applications.

To support long term-monitoring applications with eventual actions, Yu *et al.* also proposed OpenSketch, which separates the measurement data plane from the control plane [29]. In the data plane, OpenSketch provides a simple three-stage pipeline (hashing, filtering, and counting) to support many measurement tasks (e.g., heavy hitters [4], DDoS, flow size distribution [8], traffic change detection [22], and count traffic). In the control plane, OpenSketch provides a measurement library that automatically configures the pipeline and allocates resources for different measurement tasks. Chowdhury *et al.* proposed PayLess [3], a monitoring framework for flow



(a) Native-OF

(b) RFlow$^+$ with predefined attacker quarantine

Fig. 11. Effectiveness of the MAC flooding attacker quarantine

statistics collection at different aggregation levels. To further enhance effectiveness, PayLess use an adaptive scheduling algorithm (i.e., variable rate sampling based on link utilization) for flow statistics collection.

However, RFlow$^+$'s major departure from existing solutions is the practical consideration for WLAN monitoring and management from its design space; thereby supporting both *short-term* and *long-term* monitoring applications and enforcing treatments (i.e., rate-limiting and flow quarantine) based on their requirements (i.e., immediate or eventual).

VII. Conclusion

In this paper, we presented RFlow$^+$, a novel SDN-based WLAN monitoring and management framework for separately handling immediate action for short-term (e.g., 50 ms) monitoring results and eventual action for long-term (e.g., 1 month) results. We compared the accuracy and network overhead of RFlow$^+$ with existing solutions (i.e., OpenFlow and sFlow) and verified the practicality of RFlow$^+$ by showing the effectiveness of the detection and quarantine of a MAC flooding (bandwidth-hogging) attacker.

Acknowledgment

This research was supported by Global Research Lab. (GRL) Program of the National Research Foundation (NRF) funded by Ministry of Science, ICT (Information and Communication Technologies) and Future Planning(NRF-2016K1A1A2912757). This work was supported in part by the Basic Science Research Program through the National Research Foundation of Korea funded by the Ministry of Education under Grant NRF-2016R1C1B2011415. DaeHun Nyang is the corresponding author.

References

[1] The cooperative association for internet data analysis, equinix chicago data center. https://www.caida.org. [Nov 21 2013].

[2] S. Biswas, J. Bicket, E. Wong, R. Musaloiu-E, A. Bhartia, and D. Aguayo. Large-scale measurements of wireless network behavior. in SIGCOMM'15.

[3] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba. Payless: A low cost network monitoring framework for software defined networks. in NOMS'14.

[4] G. Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, Apr. 2005.

[5] A. Ghosh, R. Jana, V. Ramaswami, J. Rowland, and N. K. Shankaranarayanan. Modeling and characterization of large-scale wi-fi traffic in public hot-spots. in INFOCOM'11.

[6] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris. Combining openflow and sflow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments. *Computer Networks*, 62:122–136, 2014.

[7] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat. B4: Experience with a globally-deployed software defined wan. in SIGCOMM'13.

[8] A. Kumar, M. Sung, J. J. Xu, and J. Wang. Data streaming algorithms for efficient and accurate estimation of flow size distribution. *SIGMETRICS Perform. Eval. Rev.*, 32(1):177–188, June 2004.

[9] K.-c. Lan and J. Heidemann. A measurement study of correlations of internet flow characteristics. *Comput. Netw.*, 50(1):46–62, Jan. 2006.

[10] macof. https://github.com/ggreer/dsniff/blob/master/macof.c.

[11] N. McKeown, T. Anderson, H. Balakrishnan, G. M. Parulkar, L. L. Peterson, J. Rexford, S. Shenker, and J. S. Turner. Openflow: enabling innovation in campus networks. *CCR*, 38(2):69–74, 2008.

[12] NetFlow. http://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html.

[13] D. Nyang and D. Shin. Recyclable counter with confinement for real-time per-flow measurement. *IEEE/ACM Transactions on Networking*, PP(99):1–1, 2016.

[14] OpenDaylight. https://www.opendaylight.org/.

[15] OpenWrt. https://www.openwrt.org/.

[16] OpenWrt qos-scripts. https://wiki.openwrt.org/doc/uci/qos.

[17] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado. The design and implementation of open vswitch. in NSDI'15.

[18] M. Rodrig, C. Reis, R. Mahajan, D. Wetherall, and J. Zahorjan. Measurement-based characterization of 802.11 in a hotspot setting. in E-WIND'05.

[19] S. Sarvotham, R. Riedi, and R. Baraniuk. Connection-level analysis and modeling of network traffic. pages 99–103, 2001.

[20] J. Schulz-Zander, C. Mayer, B. Ciobotaru, S. Schmid, and A. Feldmann. Opensdwn: Programmatic control over home and enterprise wifi. in SOSR'15.

[21] J. Schulz-Zander, L. Suresh, N. Sarrar, A. Feldmann, T. Hühn, and R. Merz. Programmatic orchestration of wifi networks. in USENIX ATC'14.

[22] R. Schweller, A. Gupta, E. Parsons, and Y. Chen. Reversible sketches for efficient and accurate change detection over network data streams. in IMC'04.

[23] M. S. Seddiki, M. Shahbaz, S. Donovan, S. Grover, M. Park, N. Feamster, and Y.-Q. Song. Flowqos: Qos for the rest of us. in HotSDN'14.

[24] sFlow. http://www.sflow.org/.

[25] W. Sun, O. Lee, Y. Shin, S. Kim, C. Yang, H. Kim, and S. Choi. Wi-fi could be much more. *IEEE Communications Magazine*, 52(11):22–29, Nov 2014.

[26] A. Tootoonchian, M. Ghobadi, and Y. Ganjali. Opentm: Traffic matrix estimator for openflow networks. in PAM'10.

[27] Y. Yiakoumis, M. Bansal, A. Covington, J. van Reijendam, S. Katti, and N. McKeown. Behop: A testbed for dense wifi networks. in WiNTECH'14.

[28] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. V. Madhyastha. Flowsense: Monitoring network utilization with zero measurement cost. in PAM'13.

[29] M. Yu, L. Jose, and R. Miao. Software defined traffic measurement with opensketch. in NSDI'13.