FLIP: A Framework for Leveraging eBPF to Augment WiFi Access Points and Investigate Network Performance

Jaykumar Sheth
Internet of Things Research Lab
Santa Clara University
Santa Clara, USA
jsheth@scu.edu

Vikram Ramanna
Internet of Things Research Lab
Santa Clara University
Santa Clara, USA
vramanna@scu.edu

Behnam Dezfouli
Internet of Things Research Lab
Santa Clara University
Santa Clara, USA
bdezfouli@scu.edu

ABSTRACT

Monitoring WiFi networks is essential to gain insight into the network operation and develop methods capable of reacting to network dynamics. However, research and development in this field are hindered because there is a lack of a framework that can be easily extended to collect various types of monitoring data from the WiFi stack. In this paper, we propose FLIP, a framework for leveraging eBPF to augment WiFi access points and investigate the performance of WiFi networks. Using this framework, we focus on two important aspects of monitoring the WiFi stack. First, considering the high delay experienced by packets at access points, we show how switching packets from the wired interface to the wireless interface can be monitored and timestamped accurately at each step. We build a testbed using FLIP access points and investigate the factors affecting packet delay experienced in access points. Second, we present a novel approach that allows access points to track the duty-cycling pattern and energy consumption of their associated stations accurately and without the need for any external energy measurement tools. We validate the high energy measurement accuracy of FLIP by empirical experiments and comparisons against a commercial tool.

CCS CONCEPTS

• **Networks → Network performance evaluation**; **Wireless local area networks**; *Programming interfaces*; *Network monitoring*; *Network management*; *Programmable networks*; *Network protocols*; *Wireless access points, base stations and infrastructure.*

KEYWORDS

802.11; monitoring; delay; packet switching; energy; duty cycle.

ACM Reference Format:
Jaykumar Sheth, Vikram Ramanna, and Behnam Dezfouli. 2021. FLIP: A Framework for Leveraging eBPF to Augment WiFi Access Points and Investigate Network Performance. In *Proceedings of the 19th ACM International Symposium on Mobility Management (MobiWac '21), November 22–26, 2021, Alicante, Spain.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3479241.3486700

1 INTRODUCTION

It is estimated that by 2022 about 75% of Internet traffic will be exchanged over WiFi links [1]. Multiple observations justify the broad adoption of WiFi. First, compared to cellular networks, WiFi operates in unlicensed bands, it is relatively cheaper, and does not require a user subscription. Second, compared to other wireless technologies such as Bluetooth and ZigBee, the considerably higher data rate of WiFi is essential for multimedia communication and high-rate sensing applications. For example, surveillance cameras such as Ring and Nest use WiFi, and the communication rate of WiFi can address the bandwidth demand of ultra-high-rate sensing systems used in industrial applications [17]. Third, the production cost of WiFi stations has reduced to less than \$5 in recent years, and this has enabled the broad deployment of WiFi Access Points (APs) that provide a low-cost and easy-to-use infrastructure for wireless connectivity. The cost reduction has also facilitated the adoption of this standard in a variety of smart home devices, from voice assistants running the Linux operating system to resource-constraint devices such as smart plugs that run different types of Real-Time Operating Systems (RTOSs). In the rest of this paper, for consistency with the 802.11 standards, we use the term 'station' to refer to any sort of devices associated and served by a WiFi AP.

With the densification of APs and the heterogeneity of stations and applications served by these APs, understanding and enhancing the performance of these networks becomes more critical. For example, nowadays, WiFi APs used in residences and campuses are used to serve both regular user stations such as smartphones and laptops as well as resource-constrained IoT stations such as cameras, thermostats, and smoke detectors. In such environments, however, the traffic of regular stations adversely affects the communication timeliness and energy consumption of IoT stations. These adverse effects are primarily because the MAC layer of 802.11 (n, ac, and ax) can only differentiate between four classes of service: voice, video, best-effort, and background, where voice and video access category traffic are statistically prioritized over best-effort and background access category traffic. On top of this MAC layer sits the pfifo_fast queuing discipline—the Linux's default queuing discipline (qdisc)—which by default has three bands to differentiate between three classes of service [41]. On the other hand, parameters such as interference and AP's traffic intensity affect the delivery delay of downlink traffic. The power-saving mechanisms of the 802.11 standards present highly diverse behavior subject to traffic pattern and station type, and selecting power-saving parameters (such as wake-up period) are entirely left to the system developer.

Despite its importance, understanding the operation of the WiFi stack in various settings is a challenging undertaking for the research community. The WiFi networking stack is complex and includes multiple layers across the Wireless Network Interface Card (WL-NIC), driver, Linux kernel modules, and user-space daemons. Although there exist tools that provide visibility into some of these layers, the performance and range of visibility of these tools are far from what is needed to analyze these networks and design solutions for performance enhancement effectively. Due to this shortcoming, a large number of existing works rely on simulation. Also, when high-rate monitoring is necessary, existing works rely on packet capture and static data analysis [2, 32, 46]. Another category of works relies on tools that have been primarily designed for *infrequent* monitoring and configuration [26, 30]. For example, Linux-based tools such as iw and ethtool can be used to collect *some* of the operational data from the WiFi stack; however, the sampling rate and efficiency of these tools are far below what is required for high-rate monitoring.

In this paper, we present FLIP, a framework to augment the networking stack of Linux-based WiFi devices using the eBPF technology to collect a wide range of monitoring data that can be used for both network operation investigation as well as developing methods that react to network dynamics. We first study the operation of the WiFi stack and then show how eBPF can be leveraged to interface with the components of the WiFi stack to monitor various aspects of network operation. We focus on two aspects of network performance analysis: *First*, since existing studies reveal the considerable effect of packet switching delay in APs [31, 43], we investigate and monitor the impact of queuing and channel access contention on the delay of switching packets from the AP's wired interface to the wireless interface. We build a testbed using the FLIP framework and study how various parameters such as traffic access category and the number of stations affect the switching delay. *Second*, we propose a novel method to track the duty-cycle pattern and energy consumption of stations. To this end, we rely on the fact that stations need to inform their associated AP whenever they change their power mode (sleep to awake and vice-versa), as mandated by the 802.11 power-save mechanisms. Therefore, monitoring the driver's pertaining data structures allows for tracking stations' duty cycle patterns. This approach eliminates the need for external power measurement tools when studying the energy efficiency of resource-constrained stations. To show the effectiveness of this approach, we rely on empirical measurements and compare the accuracy of FLIP with a commercial power monitor. Our results show that the error of FLIP is 6% compared to a commercial power monitoring tool. We provide FLIP as a publicly available tool that can be implemented on off-the-shelf APs[1].

The rest of this paper is organized as follows. We present the overall architecture of FLIP in Section 2. In Section 3, we first explain the approach employed to monitor packet switching delay from the wired interface to the wireless interface, and then present an empirical analysis of this delay. In Section 4 we show how FLIP can perform passive energy monitoring of stations. Section 5 overviews the related work. We conclude the paper in Section 6. Table 1 summarizes the key notations and abbreviations used in this paper.

[1]FLIP implementation can be found at the following link: https://github.com/SIOTLAB/FLIP

Table 1: Summary of key notations and abbreviations

Notation/Abbreviation	Meaning
AP	Access Point
NSM	Network State Monitor
NSM-U	Network State Monitor in User-space
NSM-K	Network State Monitor in Kernel-space
CU	Channel Utilization
PSM	Power Save Mode
APSM	Adaptive Power Save Mode
TIM	Traffic Indication Map
WL-NIC	Wireless Network Interface Card
W-NIC	Wired Network Interface Card
UL	Uplink
DL	Downlink
Ψ	Station Awake Time
$\mathcal{E}$	Energy Consumed by Station

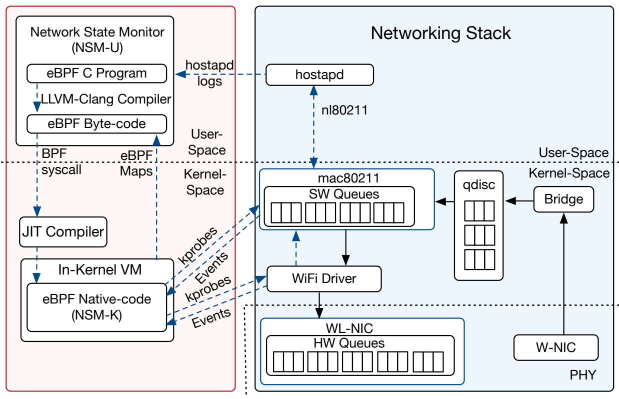

Figure 1: The FLIP architecture for augmenting APs. The right side presents the network stack, and the left side shows the Network State Monitor (NSM) module that relies on eBPF to interact with the network stack. The dotted arrows denote the collection of monitoring data. The solid arrows represent the path taken by data packets (wired-to-wireless switching data-path).

2 SYSTEM ARCHITECTURE

The WiFi stack of commercial, off-the-shelf APs includes components that span WL-NIC, driver, Linux's kernel-space modules, and user-space daemons. In this section, we present the architecture of these APs and then explain how eBPF can be leveraged to enhance visibility into the WiFi stack.

2.1 AP's Networking Stack

Apart from switching packets between the Wired Network Interface Card (W-NIC) and WL-NIC, an AP is responsible for operations such as beacon generation and handling the association and disassociation of stations. These functionalities are enabled by several components including `hostapd`, `wpa_supplicant`, `mac80211`, driver, `qdisc`, and WL-NIC, as illustrated in the right-half of Figure 1. `hostapd` is a user-space daemon that handles authentication, association, and disassociation of stations. To generate control and management frames, `hostapd` configures `mac80211` and driver via the netlink (`nl80211`) library. The `mac80211` module provides a unified interface between the driver, `qdisc`, and `hostapd`. The `mac80211` module also administers the MAC layer management entity (MLME) functions for SoftMAC drivers; sample functions are building MAC headers and assigning sequence numbers. SoftMAC drivers implement a part of layer-2 functionalities in software utilizing the host system's hardware and software resources. On the other hand, only time-critical MAC functions, such as managing timeouts, inter-frame spacing, and channel access backoff, are implemented in the WL-NIC. Currently, most commercial WL-NICs rely on the SoftMAC architecture [6], especially considering the ease of updating. Similarly, in this paper, we assume the AP's driver is based on the SoftMAC architecture. Finally, the driver is responsible for packet aggregation and transferring them to the WL-NIC for transmission on the channel.

2.2 Leveraging eBPF for Collecting Monitoring Data from the Kernel

The left-half of Figure 1 illustrates the extended Berkeley Packet Filter (eBPF) infrastructure. eBPF facilitates runtime patching of the kernel image by enabling the execution of user-defined logic when a system call or a kernel function is executed. eBPF programs (written in C) are compiled into byte code utilizing the LLVM-clang compiler. This byte code is executed in an *in-kernel* virtual machine and thus, reduces context-switching overhead. eBPF programs are attached to probe events (i.e., kprobe or a tracepoint) that mark as the instantiating points for the execution of user-defined logic. *Tracepoints* are required to be manually inserted into the kernel code by utilizing the `TRACE_EVENT()` macro. Whereas, *kprobes* are automatically defined in kernel's symbol table (`/proc/kallsyms`) along with their virtual addresses for almost all system calls and kernel functions that have been declared as neither inline nor static. Hence, we utilize kprobes because it does not require any modifications to the kernel. For a particular function (or a system call) in the kernel, BPF system calls replace the instruction at the address of the function's execution with the breakpoint instruction (e.g., `int3` for x86 platforms). Whenever this breakpoint is hit, the context of the function is saved and the user-defined logic in the eBPF program is invoked. Once the execution of user-defined logic completes, kprobe executes the instruction that was replaced by the breakpoint and continues the kernel's normal execution path. eBPF also allows accessing the kernel functions' arguments from user-space via eBPF data structures, a.k.a., eBPF Maps. eBPF Maps are transferred to the user-space via a ring buffer and can be accessed by high-level languages such as C, Python, and Lua.

The Network State Monitor (NSM) is composed of two components: *NSM user-space (NSM-U)* and *NSM kernel-space (NSM-K)*. These components utilize eBPF to log the timestamps of the kernel functions that are invoked as packets traverse the stages of the networking stack. Furthermore, the NSM-K module also logs the state of the function arguments when the function is called for execution and transfers the monitored data to NSM-U via eBPF Maps. In the subsequent sections, we will elaborate on this method of obtaining monitoring data from the kernel.

3 DELAY ANALYSIS OF SWITCHING PACKETS FROM THE WIRED INTERFACE TO THE WIRELESS INTERFACE

Once a packet is received over the wired interface of an AP, the packet needs to pass through multiple queuing disciplines before contending for channel access. Specifically, the packet will need to contend with other flows both internally in the AP's queues and also physically during the CSMA process. Investigating packet switching delay from the W-NIC to WL-NIC is particularly difficult because it depends on various factors including the AP's traffic intensity, queuing disciplines used at various layers, airtime utilization by other APs and stations, and access category of flows. Recent studies show that the delay experienced at APs is more than 60% of the total communication delay between a station and server, and this delay is between 20ms to 250ms, depending on traffic congestion [13, 30, 31, 40]. The queuing disciplines employed by the Linux's `qdisc`, driver and NIC further complicate investigating and understanding the causes of this delay [13, 14, 44]. Vendors heuristically design device drivers' packet scheduling algorithms [14, 44], and the operation of non-open-source drivers is unknown.

In this section, we present a novel approach towards measuring and monitoring packet delay from the instance it arrives on the AP's W-NIC until it is transmitted successfully by the WL-NIC. We then utilize this framework for packet delivery delay analysis.

3.1 Power Saving Methods of 802.11

An access point cannot deliver packets to an associated station in sleep mode. Therefore, the amount of time spent by packets in the AP is affected by the power-saving mode employed by the station.

Two of the most widely adopted power-save mechanisms are Power Save Mode (PSM) and Adaptive-Power Save Mode (APSM). With PSM, each AP periodically (every 102.4 ms) sends a beacon packet, and stations wake up at beacon instances to check if the Traffic Indication Message (TIM) bit in a beacon is set. If the TIM bit is set, the station sends a PS-Poll packet to indicate its transition to the awake state and to retrieve each of the queued packets from the AP. The station immediately transitions to sleep mode if the AP has no more buffered packets. The packets that arrive at the AP after the station's transition into sleep mode are queued until the next beacon instance. For example, in a request-response scenario, whenever a station sends an uplink request to a server via the AP, the response received from the server will need to wait until the next beacon announcement. For delay-sensitive communication, the APSM method allows the station to remain in an awake state for a fixed duration, known as *tail-time*, after each packet exchange with the AP. Considering the request-response scenario, if the response packet arrives before the tail-time expiry, the packet is immediately delivered to the station.

3.2 FLIP's Methodology for Monitoring Wired-to-wireless Switching Path

The right-half of Figure 1 illustrates the modules along the path taken between the W-NIC to WL-NIC. These delays are illustrated in Figure 2 and explained as follows.

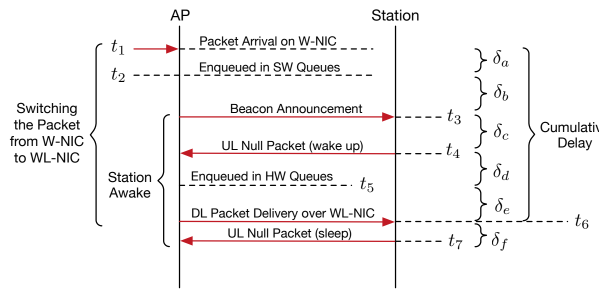

Figure 2: Delay components of a data packet being switched from the wired interface to the wireless interface. The station uses the APSM energy-efficiency mode.

3.2.1 Queuing Disciplines (qdisc). Whenever a packet arrives on the AP's wired interface at time t_1, after the MAC address table entry lookup, the packet is transferred to the network stack. Residing between the bridge and the WiFi subsystem, `qdisc` implements a programmable set of queues (a.k.a., bands), enabling a flexible traffic control framework. For example, every network interface is assigned a `qdisc`, which is pfifo_fast by default [13], consisting of three bands. The access category of each packet is inferred from the Type of Service (ToS) field in the IP header, then the packet is enqueued in one of these bands according to `qdisc`'s *priority map*. For example, pfifo_fast `qdisc`'s priority map specifies that a voice packet is enqueued in the first band, a video packet is enqueued in the second band, and background and best-effort packets are enqueued in the third band. A packet is dequeued from a band only when its higher-priority bands are empty. For example, unless the first band (corresponding to voice) is empty, packets in the second band (corresponding to video) are not dequeued. Hence, the queuing delay experienced by a packet enqueued in the lowest-priority queue depends on the current utilization level of that queue as well as the utilization of higher-priority queues.

3.2.2 mac80211. Packets are dequeued from the `qdisc` queues and handed over to the WiFi networking subsystem, whenever lower layer queues are not full.[2] Specifically, if there is available space in the `mac80211` module, packets are dequeued from `qdisc` and inserted into `mac80211` queues (at t_2), known as software (SW) Queues. For each associated station, the SW Queues include a queue per access category. Each queue is resembled by an `ieee80211_txq` structure. The `mac80211` module performs the following functions whenever a packet is enqueued in its SW Queues at t_2. First, in case the destination station is in low-power sleep state, the `ieee80211_beacon_add_tim()` function sets the TIM bit inside the next beacon to be sent at t_3. Second, the driver is notified of the pending packets via `drv_wake_tx_queue()` function. Finally, `ieee80211_sta_register_airtime` structure updates the airtime fairness metrics maintained for each station. Recent works [14] show that `qdisc` can cause latencies of higher than 100 ms due to increased queue sizes (a.k.a., bufferbloat). To remedy this problem, they propose to disable `qdisc`, and instead, an integrated traffic control mechanism (commonly known as FQ_CODEL) based on *airtime-fairness* of associated stations has been proposed. Several WiFi drivers (e.g., ath9k, ath10k, rtl8723) use this approach and employ a round-robin dequeue scheduling mechanism for each intermediate SW Queue based on the FQ_CODEL algorithm. This ensures that each station is provided with a fair-share of the channel's airtime.

[2]qdiscs such as Token Bucket Filter (TBF) and Common Applications Kept Enhanced (CAKE) allow to specify the maximum packet dequeue rate.

3.2.3 Driver and WL-NIC. The station conveys its transition into awake mode by sending a Null packet that its power-save-bit is '0'. At this point, the driver dequeues the packets from the SW Queues and passes them to the WL-NIC. WL-NIC includes five hardware (HW) queues. Four of these queues correspond to the voice, video, best-effort, and background access categories, and the last queue is used for management and control packets. Each queue is associated with a Distributed Coordination Function (DCF) unit that contends for channel access according to the Enhanced Distributed Channel Access (EDCA) parameters specified by the 802.11e amendment. These contention parameters increase the probability of longer wait times for lower access category queues, prioritizing higher access categories. The voice and background access categories have the highest and lowest priorities. Each DCF unit contends with other stations, as well as the DCF queues from the same station. The latter contention is known as an internal or virtual collision. In case of internal collision, the higher-priority access category is allowed to access the channel. Once a HW Queue obtains access to the channel for transmission, it transmits the packet to the station and generates an interrupt that triggers the dequeuing of packets from the SW Queues.

3.3 Empirical Evaluation of Wired-to-wireless Switching Delay

In this section, we use the FLIP framework for monitoring the delays across the stages of the WiFi networking stack.

The testbed setup includes an IoT station that utilizes the APSM energy efficiency mechanism. The station is associated with a FLIP AP. The testbed includes additional stations to introduce concurrent network traffic. The intensity of the traffic generated by these stations is represented as *Channel Utilization (CU)* in the results. CU is defined as $d_{activity}/d_{overall}$, where $d_{activity}$ is the time duration the radio sensed a signal higher than a pre-specified threshold value during time duration $d_{overall}$. A workstation (connected to the AP via the wired interface) sends a ping packet to the IoT station per second. This packet belongs to the best-effort access category. We refer to this traffic as the *IoT traffic*. We define *cumulative delay* as the duration between the time instance a packet arrives at the W-NIC of the AP and the time instance it is successfully transmitted over the WL-NIC. Referring to Figure 2, we also monitor delay components $\delta_a, \delta_b, \delta_c, \delta_d, \delta_e$, defined as follows:

- δ_a: The time spent by a packet in `qdisc`,
- δ_b: The interval between the insertion of a packet into the SW Queues and the next beacon announcement,
- δ_c: The time duration between the beacon announcement with TIM bit set and the NULL packet received by the AP,
- δ_d: The delay incurred in `mac80211's` SW Queues, when the HW Queues are full,

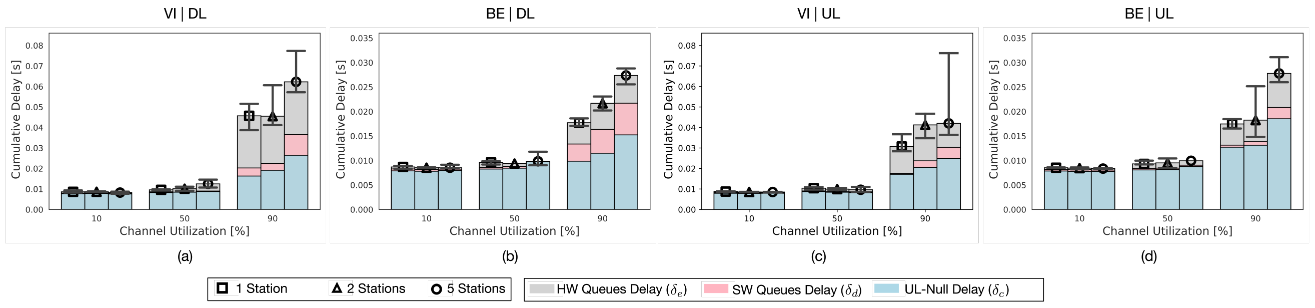

Figure 3: Delay components of wired-to-wireless switching delay in the presence of concurrent traffic. (a) Downlink (DL) video concurrent traffic. (b) Downlink (DL) best-effort concurrent traffic. (c) Uplink (UL) video concurrent traffic. (d) Uplink (UL) best-effort concurrent traffic.

- δ_e: The amount of time spent in the HW Queues when contending for channel access.

It is worth noting that with the recent improvements in Linux wireless networking, some drivers bypass the `qdisc` module. The ath9k driver used by the AP in our testbed implements this approach. Since we observed that δ_a was always less than 1 ms, we do not show δ_a in the empirical evaluation results of this paper. The duration between the time instance the packet was enqueued in the SW Queues and the beacon announcement with the station's TIM bit set (i.e., δ_b) depends on the packet arrival time and the time remaining till the next beacon announcement.

Figure 3 presents the components of wired-to-wireless switching delay in the presence of various CU levels. We consider the impact of different types of CU:

- Downlink (DL) traffic: when the traffic direction is from the AP to the stations
- Uplink (UL) traffic: when the traffic direction is from the stations to the AP
- Access category of the traffic: video and best-effort

Our results show that the cumulative delay of IoT traffic increases as CU intensifies. However, we observed that for a particular CU intensity, the values of packet delivery delays vary depending on the access category and direction of the concurrent traffic. As Figure 3(b) shows, the median cumulative delay in the presence of 90% best-effort CU is 27 ms, whereas, as the results of Figure 3(a) suggest, the cumulative delay with 90% video CU is 61 ms (i.e., 125% higher). This behavior is justified by the 802.11e amendment, which specifies the channel access contention parameters for the four access categories. Compared to the best-effort access category, the 802.11e amendment allows faster, more probable access of video HW Queue to the channel. Additionally, every time the video HW Queue grabs the channel, it can use the channel for 3.008 ms continuously; whereas, the best-effort HW Queue can send one packet per channel access. Because of these two reasons, IoT traffic (best-effort) spends more time in the HW Queue when competing with video traffic. When the concurrent traffic type is best-effort too, both traffic types can equally access the channel, thereby the delay drops. However, once the best-effort HW Queue is full due to the higher traffic rate of concurrent traffic, the dequeuing of packets from SW Queues is halted. Thus, the waiting times of the packet in the best-effort SW Queues increase as well. This is observed in Figure 3 (b) as we increase CU. The waiting duration in the SW Queue contributes to about 10% of the cumulative delay when best-effort concurrent traffic consumes 90% of the channel.

We also observe that δ_c accounts for a significant portion of the total delay. Specifically, as channel contention due to UL or DL traffic escalates, this delay increases too. This is because Null packets are regular data packets belonging to best-effort access category, and therefore, they need to contend with concurrent traffic. Additionally, even when the CU level is low, Null packets are sent at least 7 ms after the beacon. We observed that this is due to the guard times employed by the station around each beacon reception instance (these guard times have been identified in our previous work [33]).

Figures 3 (c) and (d) show that, when the concurrent traffic is UL, increasing the number of stations results in a higher delay of IoT traffic. From the CSMA point of view, this is because increasing the number of stations reduces the chance of winning the channel by the IoT station. We observe a similar trend when the concurrent traffic is DL and the AP is the only device transmitting packets. As explained in Section 3.2.3, the stations compete internally to gain channel access; therefore, increasing the number of stations receiving concurrent traffic reduces the chance of channel access for IoT traffic.

Discussion. In this section, we assumed that the IoT station wakes up at all the beacon instances. However, in [33] we showed that stations could skip some beacons instances and lower the overhead of beacon reception. This is achieved by using a *listen interval* value, denoted as τ, that represents the number of beacons skipped between wake-ups. For $\tau > 1$, the time spent in SW Queues is highly affected. For a station using listen interval τ, assume the beacon instances during the listen interval are $[t_k, ..., t_{k+\tau}]$. When a ping destined to the IoT station arrives at the AP's W-NIC during interval $[t_k, t_{k+1})$, τ beacon packets must be sent before the next wake-up instance of the station. Similarly, if the packet arrives during interval $[t_{k+1}, t_{k+2})$, the AP needs to send $\tau - 1$ beacon packets. Therefore, the expected number of beacons sent until station wake-up is $\frac{1}{\tau}(\tau + (\tau - 1) + ... + 2 + 1) = (\tau + 1)/2$, when

$\tau > 1$. The expected wake up delay is computed as $102.4 \times (\tau + 1)/2$ ms and maximum wake up delay is $\tau \times 102.4$ ms, for $\tau > 1$.

4 PASSIVE MONITORING OF STATIONS' ENERGY CONSUMPTION

In this section, we present a novel method to passively track the duty cycle and energy consumption of stations by the AP. Specifically, instead of using additional hardware to measure the energy consumption of stations, we collect the duty cycle pattern of stations from their associated AP's driver.

4.1 FLIP's Methodology for Monitoring the Energy Consumption of Stations

To monitor the duty cycle of stations, we rely on the observation that stations inform the AP whenever their WiFi subsystem transitions to another mode (sleep to awake and vice-versa). With PSM, the station sends a PS-Poll packet to the AP to express its transition into awake mode. The station retrieves queued packets until the *more-data* field is set to '0' in the data packet sent by the AP, and then transitions into sleep mode. With APSM, the station can wake up or transition into sleep mode anytime. The station informs the AP about these transitions by the power-save-bit ('0': waking up, '1': transitioning into sleep state) inside Null packets.

Within the FLIP framework, NSM is capable of keeping track of the timestamp, type, sub-type, direction, and the power-save-bit of each packets exchanged with the AP. To reduce processing overhead, the NSM-K module processes the packets belonging to the stations whose energy are being monitored. The set of these stations are programmed using the NSM-U module. Hence, for each station, via the power-save-bit and more-data fields, NSM logs the instances the station changes its operational mode. This allows FLIP to keep track of the wake-up duration of the station.

It must be noted that, stations do not inform the AP when they wake up for beacon reception; therefore, the duty cycle pattern inferred by the approach explained above does not include the overhead of beacon reception. In order to track the station's wake-up instances for beacon reception, we rely on the information provided by `hostapd`, `mac80211`, and driver. Expiration of beacon alert timer (a.k.a., `bcntimer`) in the WL-NIC generates an interrupt for beacon transmission. This interrupt is handled by the `ieee80211_beacon_get()` function, which generates and transmits a beacon. We probe this function to keep track of the timestamps and the number of beacons sent during the monitoring duration. However, the stations may not wake up at every beacon instance. As explained in Section 3.3, the stations may specify a listen interval value to reduce the overhead of beacon reception. The listen interval value is informed by the station to the AP during the association process; this value is maintained by AP's `hostapd` module (cf. Figure 1). NSM-U utilizes the `hostapd` logs to obtain the listen interval value for each associated station. With the number of beacons sent during the monitoring period and the listen interval of the station, the NSM-U module calculates the number of times the station woke up to receive beacons. To accommodate for time synchronization inaccuracy, stations allocate guard awake times around beacon reception instances[3]. In this section, we assume the station's wake-up duration per beacon instance is d_b. The total duration spent in awake mode by the station is calculated by NSM-U as $\Psi = (d_b \times c_b) + \psi$, where c_b denotes number of beacons during the monitoring period and and ψ is station awake time inferred from power-save-bit and more-data fields. The duty cycle during an interval $[t_m, t_n]$ is computed as $\mathcal{D} = \frac{\Psi}{t_n - t_m}$.

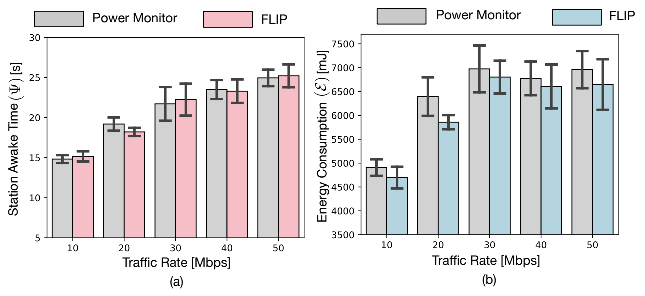

Figure 4: Comparison of FLIP's passive energy monitoring versus the energy consumption measured by a commercial power monitoring tool. The x-axis is the incoming data rate of the station. (a) Awake time (Ψ) and (b) energy consumption ($\mathcal{E}$). The station used is CYW43907 operating on APSM with tail-time set to 10ms.

To calculate the energy consumed by WL-NIC, its various operational modes must be considered: (i) sleep, (ii) reception: when the device is receiving packets, (iii) idle (a.k.a., idle listening): when the device is ready to receive packets, and (iv) transmission: when the device is transmitting packet. Devices' data-sheets provide the power consumption of these operational modes. Power consumption of idle and reception modes are very similar, and we denote their power consumption as p_{rx}. The energy consumed during idle and reception modes is calculated as $p_{rx} \times (\Psi - \Psi_{tx})$. To compute the energy consumed in transmission mode, we need to extract the time spent by the station while transmitting packets. This time is calculated as $\Psi_{tx} = \sum_{\forall p \in \mathbf{P}} \frac{l_p}{r_p}$, where $\mathbf{P}$ is the set of packets sent by the station during the monitoring duration, l_p is the length (bits) of packet p, and r_p is the physical-layer transmission rate (bps) of packet p. Within the FLIP framework, NSM-U extracts the data rate and the length of the received packets from the driver. For example, ath9k maintains the data rate and size of received packets inside the `rs_rate` and `rs_datalen` fields within the `ath_rx_status` structure. We calculate the energy consumption during a monitoring period $[t_m, t_n]$ as follows: $\mathcal{E} = p_{rx} \times (\Psi - \Psi_{tx}) + p_{tx} \times \Psi_{tx} + p_{sleep} \times (t_n - t_m - \Psi)$, where p_{tx} is the power consumption in transmission mode, and p_{sleep} is the power consumption of sleep mode. Accounting for various other factors, such as the preamble (i.e., specific signals that precedes every frame and helps in synchronizing the receiver, reduce channel noise, and reduce errors), and variations in the transmit power levels of the WiFi radio are left for future work.

[3]This was also observed in Section 3.3, where the UL Null packet sent by the station was at least 7 ms after beacon announcement. A thorough study of beacon reception overhead can be found in [33].

4.2 Empirical Evaluation of the Accuracy of Passive Energy Monitoring

In this section, we validate the accuracy of FLIP for measuring the duty cycle and energy consumption of stations' WL-NIC. We compare passive energy measurements against the results collected from a commercial energy monitoring tool [25]. In the first set of experiments, we use CYW43907 [36], which is a low-power, RTOS-based 802.11n SoC designed for IoT applications. The physical layer communication rate between this station and the AP is 54 Mbps. We use iperf to exchange traffic with the station. The station uses the APSM energy efficiency mechanism with its tail-time set to 10 ms. Figures 4 (a) and (b) compare the awake time and energy consumption results, respectively. Each experiment is 30 seconds long, each point in the graph is the median of ten experiments, and error bars represent lower and higher quartiles. These results confirm that the measurement error of FLIP is within 6% of the baseline.

In the second set of experiments, in addition to the CYW43907 station, we use WLE900VX [35] which relies on a Linux-based driver. We observed that WLE900VX does not support tail-time below 50 ms; hence, we changed the tail-time of CYW43907 to 50 ms. Also, we vary the experiments' duration to validate the accuracy of passive monitoring for various experimentation intervals. To this end, we send a ping packet to the station per second and vary the total number of pings sent during each experiment. Figure 5 presents the results. These results confirm that the measurement error of FLIP is within 9% of the baseline for experiments as long as 500 seconds.

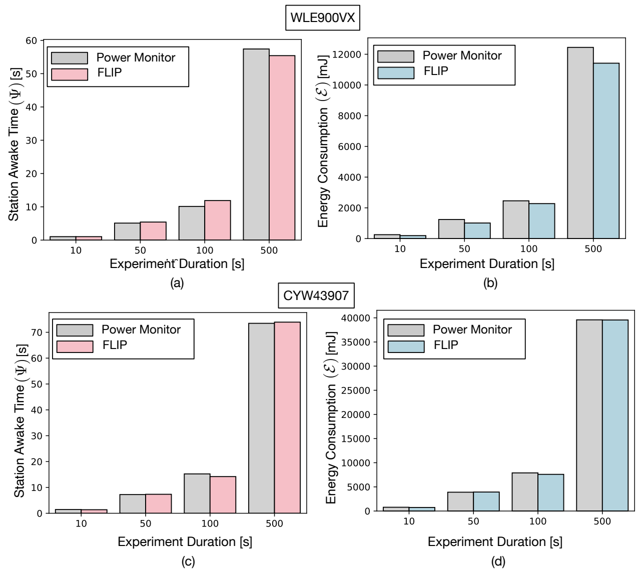

Figure 5: Awake time (Ψ) and energy consumption ($\mathcal{E}$) of stations measured by FLIP and a commercial power monitoring tool. The x-axis represents the experiment duration, which also corresponds with the number of ping packets received by the station. (a) and (b): The station is CYW43907. (c) and (d): The station is WLE900VX. For both stations the tail-time of Automatic Power Save Delivery (APSD) is set to 50 ms.

5 RELATED WORK

Monitoring WiFi Networks. Collecting monitoring data (a.k.a., network inspection or statistics collection) from WiFi APs makes it possible to study network operation [14, 20, 26, 30], enhance performance [16, 18, 24, 28, 49], and secure these networks [2, 32, 46, 48]. Measurements reflecting the state of the network enable making informed decisions on APs, central controller in Software Defined Networking (SDN), and stations. However, the existing works primarily rely on collecting monitoring data at the application and transport-layer levels. Additionally, they rely on a lower sampling rate of collecting monitoring data, whereas, for applications such as delay prediction and packet scheduling, efficient and high-rate collection of monitoring data is necessary [43]. In [38, 42] we proposed MonFi, a tool for programmable collection of monitoring data from the WiFi stack. In contrast to the approach proposed in this paper, MonFi does not utilize eBPF; therefore, adding new monitoring capabilities requires kernel modification. Additionally, MonFi cannot be used for fine-grained analysis of data path and energy monitoring of stations.

Delay Monitoring. Utilizing a large-scale testbed, [30] and [31] show that 50% of TCP packets incur a one-way latency longer than 20ms at APs and 10% of the packets incur a delay of 100ms or longer. To perform these evaluations, they utilized a custom application on stations to capture the Round-Trip Delay (RTT) experienced by ICMP packets. Apart from the logistics required for installing such user-level applications on hundreds of stations, the authors in [19] have noted that such applications are unreliable and report inflated RTT values. The primary reasons are: first, packet processing delay through the network stack and the delivery of packets to the application layer introduces a non-negligible delay; second, the WL-NIC introduces an additional packet transmission/reception delay; third, it is non-trivial to establish accurate time synchronization among the AP and stations. The FLIP framework proposed in this paper addresses these challenges by collecting monitoring data from the AP's kernel, thereby preventing the need to rely on stations to install specific monitoring applications.

Sniffer-based Approaches. Using sniffers to capture network traffic has been widely used for network monitoring [4, 7, 8, 15, 27, 29]. Manufacturers have adopted this method in commercial deployments as well. For example, Cisco Meraki's MR18 APs include integrated sniffers. However, sniffer-based approaches add additional cost as they require dedicated hardware and software resources. Specifically, once an additional WL-NIC is added to an AP, each incoming packet requires the operating system to process an additional packet. Furthermore, the monitored data obtained from the sniffer provides only higher-level information about the traffic patterns and network characteristics. For example, this approach does not provide any insight into the delays incurred at each packet processing stage. Also, depending on the processing resources available to the sniffer, it may miss capturing and logging some packets in dense environments; thus, multiple sniffers might be required to capture all the packets [19]. In particular, any difference between the location, type, and the antenna gain of the sniffer and those of the AP's WL-NIC result in discrepancies in the way the two WL-NIC perceive the channel.

Network Management Protocols. Network and flow management protocols, such as SNMP, NETCONF, and NetFlow, are commonly used for obtaining network statistics. For example, authors in [34] use the k-means clustering technique to isolate groups of users experiencing similar Quality of Service (QoS) levels in a large-scale enterprise network. Undesirable QoS is inferred based on users dropping the communication link. The authors collected data from 363 APs via `rsyslog` (for network layer monitoring) and NetFlow (e.g., number of clients associated with each AP). However, the set of monitoring data exposed by these protocols primarily target wired networking devices. Additionally, SNMP imposes a high overhead on network throughput [10, 45]. Hence, the rate of network monitoring is considerably lower. For example, in [3, 50], SNMP logs are polled every 5 minutes, and in [21], monitoring data is collected once every 30 minutes.

Obtaining Monitoring Data by Utilizing Linux Tools. Utilities such as `ethtool`, `iwpriv`, and `debugfs` allow the collection of monitoring data from Linux networking devices. `iw` is a netlink-based interface utility for configuring wireless devices. For example, the `'station dump'` option prints information about the connected stations; this information includes RSSI, the data rate of the received packets, and the timestamp of the stations' most recent activity. SoftMAC drivers such as ath9k, ath10k, and ath11k provide a debug mode utilizing the `debugfs` file system. For example, the most widely monitored performance metric using `ethtool` and `debugfs` is CU. However, these tools were developed to debug or configure the network devices, not for an efficient and high-rate collection of monitoring data. Additionally, the statistics reported by `debugfs` and `ethtool` are not extensive and in order to add a debug statement (in case of `debugfs`) or an ioctl call (in case of `ethtool`), the associated kernel modules must be modified.

Using eBPF. The importance of eBPF has been highlighted by academia and industry in various application domains such as system monitoring, enhancing security, and supporting programmability. Contrary to modifying or adding kernel modules, eBPF programs are verified before being loaded in the kernel, which makes it a safer mechanism for accessing kernel memory [22]. Hence, the networking industry relies on eBPF in production environments for system profiling and enhancing network functions by adding programmability to the data plane. For example, Facebook's *Katran* [37] and Sysdig's *Falco* [39] use eBPF for implementing a layer-4 load balancer and a Kubernetes runtime security and monitoring tool, respectively. Netflix has developed *Flow Exporter* for observing the per-flow transport layer (i.e., TCP/IP) statistics, and they reported that the overhead of using the Flow Exporter tool is less than 1% of the processor time; thereby enabling scalability over many Amazon Web Services (AWS) instances. One of the most notable technologies enabled by eBPF is eXpress Data Path (XDP) [12], which allows tapping into the reception path of the network stack before any memory allocation, resulting in significant performance benefits compared to other packet processing mechanisms. XDP is actively being used in various domains such as Distributed Denial-of-Service (DDoS) mitigation and packet inspection at Cloudflare and Facebook. Despite the wide adoption of eBPF, our work is the first to utilize eBPF for monitoring WiFi networks.

Energy Monitoring of WiFi Networks. Energy monitoring provides insights into the effect of networking protocols, deployment strategy, and application type on stations' energy efficiency, especially for resource-constrained IoT stations. In low-power stations, WL-NIC accounts for more than 50% of the station's total energy budget; therefore, monitoring WL-NIC's energy consumption is an essential performance monitoring metric [5, 23, 47]. Since commercial power monitors are bulky and expensive, various low-cost power measurement tools have been proposed [9, 11]. Nevertheless, using these tools introduces significant challenges. First, these platforms are required to be physically connected to the stations, thereby causing scalability concerns. Second, to separate the energy consumption of WL-NIC from other components such as processor, the power input-pins of the WL-NIC must be found and probed.

6 CONCLUSION

In this paper, we studied the internals of the WiFi networking stack and demonstrated how various components of this stack handle the operations pertaining to packet switching at APs and energy-efficient operation of stations. We then leveraged eBPF to augment WiFi APs and performed system monitoring in terms of packet switching delay and tracking the energy consumption of stations. The empirical studies of this paper show how the proposed framework can be used to investigate the operation of WiFi networks. In addition to investigating various aspects of WiFi networks, the FLIP framework can also be used to collect monitoring data and develop methods that react to network dynamics. For example, by providing insights into packet switching delay and the instantaneous energy efficiency of stations, FLIP facilitates the development of algorithms that manage the steering of stations among APs. Developing methods that rely on the FLIP framework is left as future work.

REFERENCES

[1] 2020. *Cisco Annual Internet Report (2018–2023) White Paper.* https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[2] Eirini Anthi, Lowri Williams, Małgorzata Słowińska, George Theodorakopoulos, and Pete Burnap. 2019. A supervised intrusion detection system for smart home IoT devices. *IEEE Internet of Things Journal* 6, 5 (2019), 9042–9053.

[3] Magdalena Balazinska and Paul Castro. 2003. Characterizing mobility and network usage in a corporate wireless local-area network. In *Proceedings of the international conference on Mobile systems, applications and services.* 303–316.

[4] Sanjit Biswas, John Bicket, Edmund Wong, Raluca Musaloiu-e, Apurv Bhartia, and Dan Aguayo. 2015. Large-scale measurements of wireless network behavior. In *Proceedings of the ACM Conference on Special Interest Group on Data Communication.* 153–165.

[5] Adam Bujnowski, Kamil Osinski, and Jerzy Wtorek. 2017. A navigation device utilizing body communication channel for mobile wearable systems. In *10th International Conference on Human System Interactions (HSI).* IEEE, 25–30.

[6] Gianluca Cena, Stefano Scanzio, and Adriano Valenzano. 2018. SDMAC: a software-defined MAC for Wi-Fi to ease implementation of soft real-time applications. *IEEE Transactions on Industrial Informatics* 15, 6 (2018), 3143–3154.

[7] Yu-Chung Cheng, Mikhail Afanasyev, Patrick Verkaik, Péter Benkö, Jennifer Chiang, Alex C Snoeren, Stefan Savage, and Geoffrey M Voelker. 2007. Automating cross-layer diagnosis of enterprise wireless networks. *ACM SIGCOMM Computer Communication Review* 37, 4 (2007), 25–36.

[8] Yu-Chung Cheng, John Bellardo, Péter Benkö, Alex C Snoeren, Geoffrey M Voelker, and Stefan Savage. 2006. Jigsaw: Solving the puzzle of enterprise 802.11 analysis. *ACM SIGCOMM Computer Communication Review* 36, 4 (2006), 39–50.

[9] Behnam Dezfouli, Immanuel Amirtharaj, and Chia-Chi Chelsey Li. 2018. EMPIOT: An energy measurement platform for wireless IoT devices. *Journal of Network and Computer Applications* 121 (2018), 135–148.

[10] Behnam Dezfouli, Vahid Esmaeelzadeh, Jaykumar Sheth, and Marjan Radi. 2018. A review of software-defined WLANs: Architectures and central control mechanisms. *IEEE Communications Surveys & Tutorials* 21, 1 (2018), 431–463.

[11] Karina Gomez, Roberto Riggio, Tinku Rasheed, Daniele Miorandi, and Fabrizio Granelli. 2012. Energino: A hardware and software solution for energy consumption monitoring. In *10th International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt)*. IEEE, 311–317.

[12] Toke Høiland-Jørgensen, Jesper Dangaard Brouer, Daniel Borkmann, John Fastabend, Tom Herbert, David Ahern, and David Miller. 2018. The express data path: Fast programmable packet processing in the operating system kernel. In *Proceedings of the 14th international conference on emerging networking experiments and technologies*. 54–66.

[13] Toke Høiland-Jørgensen, Per Hurtig, and Anna Brunstrom. 2015. The Good, the Bad and the WiFi: Modern AQMs in a residential setting. *Computer Networks* 89 (2015), 90–106.

[14] Toke Høiland-Jørgensen, Michał Kazior, Dave Täht, Per Hurtig, and Anna Brunstrom. 2017. Ending the anomaly: Achieving low latency and airtime fairness in WiFi. In *Proceedings of USENIX*. 139–151.

[15] Si Young Jang, Byoungheon Shin, and Dongman Lee. 2016. An adaptive tail time adjustment scheme based on inter-packet arrival time for IEEE 802.11 WLAN. In *ICC*. IEEE, 1–6.

[16] Glenn Judd and Peter Steenkiste. 2002. Fixing 802.11 access point selection. *ACM SIGCOMM Computer Communication Review* 32, 3 (2002), 31–31.

[17] Chia-Chi Li, Vikram Ramanna, Daniel Webber, Cole Hunter, Hack Tyler, and Behnam Dezfouli. 2021. Sensifi: A Wireless Sensing System for Ultra-High-Rate Applications. *IEEE Internet of Things Journal* (2021).

[18] Shenghong Li, Mark Hedley, Keith Bengston, David Humphrey, Mark Johnson, and Wei Ni. 2019. Passive localization of standard WiFi devices. *IEEE Systems Journal* 13, 4 (2019), 3929–3932.

[19] Weichao Li, Daoyuan Wu, Rocky KC Chang, and Ricky KP Mok. 2017. Toward accurate network delay measurement on android phones. *IEEE Transactions on Mobile Computing* 17, 3 (2017), 717–732.

[20] Simon Liu, V Ramanna, and Behnam Dezfouli. 2020. Empirical Study and Enhancement of Association and Long Sleep in 802.11 IoT Systems. In *Global Communications Conference (GLOBECOM)*.

[21] Feng Lyu, Ju Ren, Nan Cheng, Peng Yang, Minglu Li, Yaoxue Zhang, and Xuemin Shen. 2019. Big data analytics for user association characterization in large-scale wifi system. In *IEEE International Conference on Communications (ICC)*. IEEE, 1–6.

[22] Sebastiano Miano, Fulvio Risso, Mauricio Vásquez Bernal, Matteo Bertrone, and Yunsong Lu. 2021. A framework for eBPF-based network functions in an era of microservices. *IEEE Transactions on Network and Service Management* 18, 1 (2021), 133–151.

[23] Aleksandar Milenkovic, Milena Milenkovic, Emil Jovanov, Dennis Hite, and Dejan Raskovic. 2005. An environment for runtime power monitoring of wireless sensor network platforms. In *Proceedings of the Thirty-Seventh Southeastern Symposium on System Theory (SSST)*. IEEE, 406–410.

[24] Laudin Molina, Tanguy Kerdoncuff, Dareen Shehadeh, Nicolas Montavont, and Alberto Blanc. 2017. WMSP: Bringing the wisdom of the crowd to WiFi networks. *IEEE Transactions on Mobile Computing* 16, 12 (2017), 3580–3591.

[25] Monsoon Power Monitor. 2013. online]: http://www. msoon. com/LabEquipment. *PowerMonitor/, visited Nov* (2013).

[26] Guoshun Nan, Xiuquan Qiao, Jiting Wang, Zeyan Li, Jiahao Bu, Changhua Pei, Mengyu Zhou, and Dan Pei. 2018. The Frame Latency of Personalized Livestreaming Can Be Significantly Slowed Down by WiFi. In *IPCCC*. IEEE, 1–8.

[27] Ashish Patro, Srinivas Govindan, and Suman Banerjee. 2013. Observing home wireless experience through wifi aps. In *Proceedings of the 19th annual international conference on Mobile computing & networking*. 339–350.

[28] Changhua Pei, Zhi Wang, Youjian Zhao, Zihan Wang, Yuan Meng, Dan Pei, Yuanquan Peng, Wenliang Tang, and Xiaodong Qu. 2017. Why it takes so long to connect to a WiFi access point. In *IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 1–9.

[29] Changhua Pei, Youjian Zhao, Guo Chen, Yuan Meng, Yang Liu, Ya Su, Yaodong Zhang, Ruming Tang, and Dan Pei. 2017. How Much Are Your Neighbors Interfering with Your WiFi Delay?. In *International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 1–9.

[30] Changhua Pei, Youjian Zhao, Guo Chen, Ruming Tang, Yuan Meng, Minghua Ma, Ken Ling, and Dan Pei. 2016. WiFi can be the weakest link of round trip network latency in the wild. In *IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 1–9.

[31] Changhua Pei, Youjian Zhao, Yunxin Liu, Kun Tan, Jiansong Zhang, Yuan Meng, and Dan Pei. 2017. Latency-based WiFi congestion control in the air for dense WiFi networks. In *IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*. IEEE, 1–10.

[32] Antônio J Pinheiro, Jeandro de M Bezerra, Caio AP Burgardt, and Divanilson R Campelo. 2019. Identifying IoT devices and events based on packet length from encrypted traffic. *Computer Communications* 144 (2019), 8–17.

[33] Vikram K Ramanna, Jaykumar Sheth, Simon Liu, and Behnam Dezfouli. 2021. Towards Understanding and Enhancing Association and Long Sleep in Low-Power WiFi IoT Systems. *IEEE Transactions on Green Communications and Networking* (2021).

[34] Lucio Henrik A Reis, Luiz Claudio S Magalhães, Dianne Scherly V de Medeiros, and Diogo MF Mattos. 2020. An unsupervised approach to infer quality of service for large-scale wireless networking. *Journal of Network and Systems Management* 28, 4 (2020), 1228–1247.

[35] Compex Systems. [n.d.]. *Compex WLE900VX 3X3 MIMO wireless adapter.* https://compex.com.sg/shop/wifi-module/802-11ac-wave-1/wle900vx-

[36] Cypress Semiconductor. [n.d.]. *CYW43907: IEEE 802.11 a/b/g/n SoC with an Embedded Applications Processor.* http://www.cypress.com/file/298236/download

[37] Facebook. [n.d.]. *Facebook: Katran.* https://github.com/facebookincubator/katran

[38] SIOTLAB. [n.d.]. *MonFi: Tool for High-Rate, Efficient, and Programmable Monitoring of WiFi Devices.* https://github.com/SIOTLAB/MonFi

[39] Sysdig. [n.d.]. *Sysdig falco: Behavioral activity moni-toring with container support.* https://github.com/draios/oss-falco

[40] Jose Saldana, José Ruiz-Mas, and Jose Almodovar. 2017. Frame aggregation in central controlled 802.11 WLANs: The latency versus throughput tradeoff. *IEEE Communications Letters* 21, 11 (2017), 2500–2503.

[41] Jaykumar Sheth and Behnam Dezfouli. 2019. Enhancing the Energy-Efficiency and Timeliness of IoT Communication in WiFi Networks. *IEEE Internet of Things Journal* 6, 5 (2019), 9085–9097.

[42] Jaykumar Sheth and Behnam Dezfouli. 2021. MonFi: A Tool for High-Rate, Efficient, and Programmable Monitoring of WiFi Devices. In *IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 1–7.

[43] Jaykumar Sheth, Cyrus Miremadi, Amir Dezfouli, and Behnam Dezfouli. 2020. EAPS: Edge-Assisted Predictive Sleep Scheduling for 802.11 IoT Stations. *arXiv preprint arXiv:2006.15514* (2020).

[44] Ahmad Showail, Kamran Jamshaid, and Basem Shihada. 2016. Buffer sizing in wireless networks: challenges, solutions, and opportunities. *IEEE Communications Magazine* 54, 4 (2016), 130–137.

[45] Fabricio A Silva, Linnyer Beatrys Ruiz, Thais Regina M Braga, José Marcos S Nogueira, and Antonio Alfredo Ferreira Loureiro. 2005. Defining a Wireless Sensor Network Management Protocol.. In *LANOMS*. Citeseer, 39–50.

[46] Arunan Sivanathan, Hassan Habibi Gharakheili, Franco Loi, Adam Radford, Chamith Wijenayake, Arun Vishwanath, and Vijay Sivaraman. 2018. Classifying IoT devices in smart environments using network traffic characteristics. *IEEE Transactions on Mobile Computing* 18, 8 (2018), 1745–1759.

[47] Li Sun, Haotian Deng, Ramanujan K Sheshadri, Wei Zheng, and Dimitrios Koutsonikolas. 2016. Experimental evaluation of WiFi active power/energy consumption models for smartphones. *IEEE Transactions on Mobile Computing* 16, 1 (2016), 115–129.

[48] Bhagyashri Tushir, Yogesh Dalal, Behnam Dezfouli, and Yuhong Liu. 2020. A Quantitative Study of DDoS and E-DDoS Attacks on WiFi Smart Home Devices. *IEEE Internet of Things Journal* (2020).

[49] Jun Zhang, Guangxing Zhang, Qinghua Wu, Lei Song, and Gaogang Xie. 2017. LazyAS: Client-transparent access selection in dual-band WiFi. In *26th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 1–9.

[50] Mengyu Zhou, Kaixin Sui, Minghua Ma, Youjian Zhao, Dan Pei, and Thomas Moscibroda. 2016. Mobicamp: A campus-wide testbed for studying mobile physical activities. In *Proceedings of the 3rd International on Workshop on Physical Analytics*. 1–6.