MonFi: A Tool for High-Rate, Efficient, and Programmable Monitoring of WiFi Devices

Jaykumar Sheth and Behnam Dezfouli
Internet of Things Research Lab, Department of Computer Science and Engineering, Santa Clara University, USA
jsheth@scu.edu, bdezfouli@scu.edu

***Abstract*—The 802.11 standard, known as WiFi, is currently being used for a wide variety of applications. The increasing number of WiFi devices, their stringent communication requirements, and the need for higher energy-efficiency mandate the adoption of novel methods that rely on monitoring the WiFi communication stack to analyze, enhance communication efficiency, and secure these networks. In this paper, we propose MonFi, a publicly-available, open-source tool for high-rate, efficient, and programmable monitoring of the WiFi communication stack. With this tool, regular user-space applications can specify their required measurement parameters, monitoring rate, and measurement collection method as event-based, polling-based, or a hybrid of both. We also propose methods to ensure deterministic sampling rate regardless of the processor load caused by other processes including packet switching. In terms of sampling rate and processing efficiency, we show that MonFi outperforms the Linux tools used to monitor the communication stack.**

***Index Terms*—WiFi stack, 802.11, Real-time monitoring, Overhead, Linux.**

I. Introduction

WiFi devices currently cost as low as $3, and WiFi devices running the Linux operating system cost as low as $9. WiFi chips are being used in various products, including laptops, smartphones, and IoT devices such as cameras, thermostats, light bulbs, door locks, and home appliances. Communication among these IoT devices is integral to developing Building Management Systems (BMSs), smart cities, and Industry 4.0. The economic value of WiFi is expected to rise to $3.5 trillion in 2023, up from $2 trillion in 2018 [1]. An important observation is that by 2021, about 60% of the mobile devices' traffic will be offloaded onto WiFi networks. The rapid increase of WiFi devices has urged Federal Communications Commission (FCC) to allocate newer frequency bands for WiFi communication. The broad adoption of WiFi is supported by: Firstly, compared to other wireless technologies such as Bluetooth and ZigBee, WiFi provides higher data rates, making delay-sensitive and multimedia communication possible. For example, surveillance cameras such as Ring and Nest use WiFi. Secondly, since WiFi operates in unlicensed bands, it is relatively cheap and does not require a subscription, compared to cellular networks. Finally, the wide deployment of WiFi Access Points (APs) lowers the costs pertaining to adopting this standard for wireless connectivity.

Collecting *monitoring data*[1] from WiFi devices makes is possible to study network operation [2]–[5], improve performance [6]–[8], and secure these networks [9]–[12]. For example, methods have been proposed to manage channel assignment to APs, control the association of stations, and adjust transmission power, to mention a few [13]–[15]. Although collecting monitoring data every few seconds would be enough for some algorithms (e.g., channel assignment), immediate reactions to network dynamics require high-rate, real-time monitoring. A sample scenario is per-flow and per-packet scheduling methods in dense networks [8]. Also, existing research shows the importance of packet-level analysis for power management [6], [16]. Concerning the new 802.11ax standard, the allocation of resource blocks, quite time period, and Target Wake-up Time (TWT) requires monitoring the communication efficiency and packet backlog for each station [17]. We also observe the adoption of data-driven and machine-learning-based methods for performance enhancement (e.g., delay reduction [4], power management [18]–[20]) and security provisioning [9]–[12].

The existing works reveal a major challenge: *the lack of a tool for high-rate, real-time, efficient, and programmable monitoring of WiFi APs and stations*. Due to this shortcoming, a large number of existing works rely on simulation. Also, when high-rate monitoring is necessary, some works instead use packet capture and static data analysis [9]–[11]. Another category of works relies on tools that have been primarily designed for infrequent monitoring and configurations [3], [4]. For example, tools such as `iw` and `ethtool` can be used to collect *some* of the required parameters; however, their efficiency is far below what is required for high-rate monitoring. We also note that the WiFi management systems developed by companies such as Cisco's Meraki, HP's Aruba, and Arista are proprietary and cannot be used for research and development. The literature also includes several protocols proposed for remote configuration and management of APs. However, the proposed proprietary protocols and the standard telemetry protocols (e.g., OpenConfig [21] and NETCONF [22]) only provide a high-level interface for remote interactions, and they do not offer any means to monitor the WiFi stack.

In this work, we present MonFi, a Linux-based, open-source tool for efficient, high-rate, and programmable monitoring of WiFi devices[2]. This tool allows for monitoring the complete WiFi stack—Network Interface Card (NIC), driver, mac80211,

[1]Other terms used in the literature are measurement collection, performance monitoring, network inspection, statistics collection, and network telemetry.

[2]The implementation of MonFi can be found at the following link: https://github.com/SIOTLAB/MonFi

cfg80211, hostapd, and qdisc. The monitoring frequency and the type of measurements collected can be programmatically specified using the user-space component of MonFi. We present methods to monitor the WiFi stack, and also implement and study methods for reducing the overhead of kernel to user-space communication and stabilizing monitoring rate in the presence of interfering loads on the processor. Through empirical evaluations, we show the higher efficiency of MonFi in terms of monitoring rate, stability of monitoring rate, and processor utilization, compared to the existing tools. For example, MonFi achieves about 28% higher monitoring rate and 16% lower processor utilization compared to debugfs, which is a widely-used method for monitoring communication stack. Compared to ioctl, the monitoring rate of MonFi is 21x faster.

As a publicly-available open-source tool, MonFi offers new opportunities in developing WiFi systems, from smart homes to enterprise networks to real-time industrial systems. Besides, MonFi reduces the development costs associated with using additional measurement devices. For example, by providing per-station information, MonFi eliminates the need for using additional hardware tools to analyze stations' power status.

The rest of this paper is organized as follows: Section II overviews the literature and the tools used to monitor the WiFi stack. We present the design and implementation of MonFi in Section III. Section IV presents performance evaluation results. We conclude the paper in Section V.

II. Background

In this section, we first overview the existing works that leverage WiFi stack monitoring for various purposes. Then we overview the tools available for monitoring WiFi devices.

A. Enhancing WiFi Networks by Collecting Monitoring Data

Many studies highlight the importance of collecting monitoring data across the WiFi stack. Centrally-controlled WiFi networks heavily rely on measurement collection from APs to perform channel assignment, station association, and mobility control [7], [13], [14]. However, these works primarily focus on application-layer data collection protocols and programmability. For example, a majority of these works rely on OpenFlow and Open vSwitch (OVS) to collect per-flow statistics. Also, a large number of these works assume that the required statistical data are available either via Linux tools (e.g., debugfs [7], [23], procfs [24]) or simulation [13]. An overview of these works can be found in [15].

Nan et al. [3] utilized Linux tools (cf. Section II-B) to collect several parameters including queue length, Airtime Utilization (AU), physical data rate, and retry rate, every 100 ms. They show that WiFi latency accounts for more than 80% of video frame latency in personalized live-streaming applications. Pei et al. [4] proposed WiLy, a user-space application that time stamps all packet exchanges via the AP's wireless and wired interfaces. WiLy uses libpcap to capture packets and rsync to exchange the results with a server. Analyzing the data collected by WiLy shows that in about 50% of the cases, WiFi latency comprises more than 60% of Round-Trip Delay (RTT). To study the impact of various parameters on WiFi latency, they use Linux tools such as debugfs, ifconfig, and iw every 10 seconds to reduce the processing overhead of APs. Other similar studies are [25] and [26]. Pei et al. [27] introduced the WiFi Manager app to collect association time information across a large number of stations. Association delay data are then used along with information about stations and APs (e.g., number of associated stations, encryption type, AP model) to identify the causes of association delay. The effect of buffer size on the delay experienced by stations has been studied in [2], [28]. For example, [2] highlights the importance of monitoring queuing discipline (qdisc) layer and driver's queues to understand the causes of communication delay with stations.

Using MAC-level information such as inter-packet intervals has been used for adjusting power-saving methods [6], [16], [29]. For example, Xiao et al. [30] use parameters including rate, inter-packet interval, and MAC transmission to characterize traffic and derive energy consumption models. Pyles et al. [20] study the effect of incoming packet rate on the extra time spent in awake mode by stations employing Power Save Mode (PSM).

Several studies have collected WiFi communication statistics to classify IoT devices or detect attacks to or from these devices [9]–[11]. Sivanathan et al. [10] discuss the potential benefits of real-time data collection, but they do not propose any solution.

B. Existing Monitoring Tools

Most WiFi NIC drivers are implemented either as an extension of the kernel or as a loadable kernel module; thus, encapsulating hardware resources within the protected kernel space memory. To access the data maintained by the NIC and driver, user-space programs rely on network management tools such as iwconfig, iwpriv, ethtool, and ifconfig. These tools generally use ioctl, sysfs, or netlink for the communication between user-space processes and kernel.

ethtool allows for monitoring and configuring NIC. This tool constitutes a user-space module and a kernel-space module. These two modules communicate via ioctl, which extends the native Linux system call operations by providing functions for hardware configurations that use predefined data structures. These functions can be modified when new functionalities are added to the hardware. However, this may result in non-backward-compatible updated functions. Thereby, ioctl is not user friendly and it is difficult to extend. The information provided by ethtool with the `-S` (`--statistics`) option can be helpful in obtaining channel state information along with driver statistics such as the number of packets transmitted or received by the NIC. For example, AU can be retrieved using `ch_time_busy` and `ch_time`. However, the data retrieved by ethtool is not extensive and does not include some of the essential data about network performance. For example, per-station statistics, state of qdisc, and NIC's register values, such as current NAV or failed FCS count, cannot be collected. Additionally, ethtool retrieves all the

counters at once, preventing the user from specifying the list of measurement parameters. This results in the extra overhead of polling additional registers and driver's data structures, as well as searching for the required information in the collected data by user applications. Additionally, although the NIC updates AU per its internal clock cycle, ethtool reports AU time in milliseconds granularity, which results in lower accuracy and higher reporting delay.

SoftMAC-based wireless drivers commonly provide a debug mode utilizing Virtual File System (VFS). Unlike regular files that reside on disk, VFSs (e.g., sysfs, procfs, and debugfs) reside in the main memory and facilitate communication between user-space and kernel-space. These interfaces remain empty unless a user-space process requests the resources. When requested, the kernel gathers the required measurements and populates the interface. For example, the ath9k driver utilizes regdump-debugfs to retrieve the values stored in all registers exposed to the driver by the NIC. A major shortcoming of debugfs is that it does not allow to query an arbitrary set of registers of the NIC. This results in a large number of unnecessary PCIe bus interrupts. Furthermore, data transfer size is limited to one memory page [31].

Using netlink addresses some of the aforementioned challenges. Firstly, since it is a socket-based mechanism, netlink can be initiated by both kernel-space and user-space processes, whereas, VFSs and ioctl can be instantiated only by user-space processes. This is particularly useful for event-based data collection. For example, consider a scenario where the kernel sends measurements to a user-space application whenever a packet is received. Secondly, netlink facilitates asynchronous communication by storing messages in queues and initiating the receiver's reception handler. The receiver can process the information synchronously or asynchronously. In contrast, ioctl and VFSs communicate synchronously. netlink can also multicast to multiple processes at once, whereas, ioctl and VFSs support unicast messages only.

III. Design and Implementation of MonFi

In this section, we identify the measurements that are indicative of network operation and dynamics, and propose methods for collecting these measurements from various components of the networking stack. We also present methods to achieve monitoring rate determinism. Finally, we discuss the monitoring modes provided by MonFi.

A. Architecture

Figure 1 presents the architecture of MonFi. The Controller is a user-space module that configures and receives measurements from the Collector. The Collector is a kernel-space module that collects data across the network stack. The Collector is implemented as a part of the driver to share a set of functions and data structures. The Collector also interacts with other modules of the communication stack.

In general, current 802.11 drivers are categorized as either SoftMAC or FullMAC. SoftMAC drivers implement a part of the MAC layer management entity (MLME) in software, thereby utilizing the host system's processing resources. Only time-critical MAC functions, such as managing timeouts, interframe spacing, and channel access backoff, are implemented in the hardware. Whereas, FullMAC utilizes an additional processor on the NIC to implement all MAC functions within the firmware. Currently, most commercial WiFi adapters rely on a SoftMAC architecture [32], especially considering the ease of updating. Thus, in this paper, we assume the underlying WiFi NIC's driver is a SoftMAC.

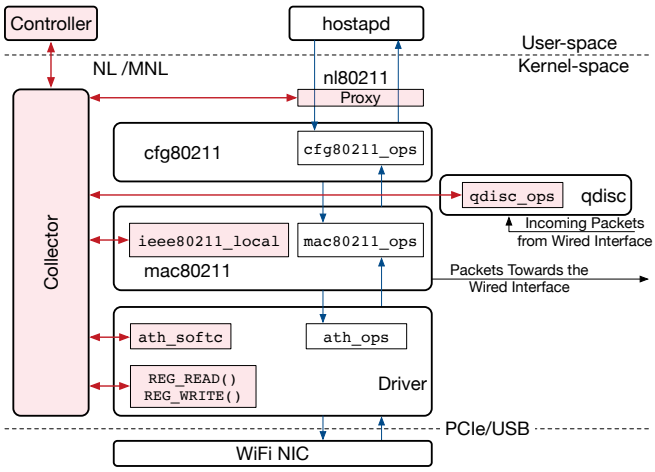

Fig. 1. The architecture of MonFi. The Controller allows use-space programs to specify the type, rate, and method of collecting measurements across the WiFi stack. The Controller collects the requested data from the stack. Various kernel modules have been modified to facilitate collecting monitoring data from them. Note that the hostapd daemon is only used when MonFi runs in an AP.

To perform the standard AP functionalities, we use hostapd [33], which is used by Commercial Off-The-Shelf (COTS) APs. hostapd is a user-space daemon that handles beacon transmission, authentication, and association of stations.

B. Extensive Data Collection Across the Protocol Stack

In this section, we identify and discuss the process of monitoring NIC, driver, qdisc, and the other components of the WiFi stack.

1) PHY and MAC measurements: WiFi NICs provide I/O memory regions consisting of registers that can be accessed via the PCIe bus. These registers allow to configure the NIC and obtain its internal status. Accessor functions, such as `ioread32()` and `iowrite32()`, are utilized to *get* and *set* the status of the registers, respectively. The Collector utilizes these functions to extract register values from the NIC. For example, to compute AU, which reflects the level of activity on the wireless channel, the Collector fetches the values of `AR_CCCNT` and `AR_RCCNT` registers (available in Atheros chipsets). The former register stores the time elapsed since the start time of the NIC, and the latter stores the amount of time the NIC has been sensing activity on the channel. We denote these measurements as T and B, respectively. The AU during an interval $t1$ to $t2$ is computed as $(B_{t2} - B_{t1})/(T_{t2} - T_{t1})$.

Most COTS NICs (e.g., Netely NET-900M and Compex WLE900VX) operate with the reference clock speed of 44 MHz when using the 2.4 GHz band and 40 MHz when using the 5 GHz band (considering a 20 MHz channel), and update the values of the registers corresponding to the clock rate. The Collector retrieves the values of these registers, and then, either decodes them to decimal values and in milliseconds format, or stores the raw measurements that can be decoded asynchronously when needed. The Controller also provides the functionality that allows user-space applications to specify the list of registers or register addresses.

Compared to the legacy 802.11 standards (i.e., a/b/g), the recent standards (i.e., n/ac/ax) offer numerous physical layer enhancements that result in bit rates beyond 600 Mbps. Some of these enhancements are concurrent Multiple-Input Multiple-Output (MIMO) streams, wider channel bandwidth, and higher-order Modulation and Coding Schemes (MCSs). These parameters can be configured using hostapd's configuration file or `hostapd_cli reconfigure`. hostapd utilizes netlink to communicate with the WiFi Configuration API—cfg80211. The cfg80211 module acts as a bridge between user-space and kernel, and provides a unified interface in the form of callback functions to control the NIC. Each callback implemented in cfg80211 is associated with a corresponding function in the driver to configure the NIC. The Collector intercepts the netlink events triggered by hostapd to keep track of any modifications applied to the NIC.

The physical layer configuration of the NIC may not necessarily represent the parameters used by each AP-station pair when communicating. For example, even when an AP announces supporting 40 MHz channels, the station may not support this channel width, and instead use a 20 MHz channel. Also, the two ends of a communication link dynamically change their MCS, based on multiple factors such as Received Signal Strength Indicator (RSSI) and retransmission rate. Therefore, each link's physical layer parameters are essential for characterizing communication efficiency in terms of metrics such as throughput and Packet Error Rate (PER) on a per-station basis. To this end, MonFi collects RSSI, MCS, and the number of MAC layer retransmissions, *on a per-station basis*, as follows. The kernel maintains a global list of devices utilizing the `net_device` structures in mac80211. The `*priv` field contains driver-specific structures and maintains the statistics for respective stations. This field is represented by `ath_softc` in Atheros NICs. The Controller module allows user-space applications to specify the stations that must be monitored. The Collector collects per-station measurements requested by the Controller by polling the corresponding fields of the `softc` data structure and reports them to the Controller along with the MAC address of the station. In order to avoid data copy, the pointer to the `softc` data structure is shared between all tasks in the driver. However, race condition happens when multiple tasks in the driver try to access `softc` concurrently. We utilize a mutex lock before accessing `softc` to avoid race conditions.

The 802.11 standard supports various power-saving methods to allow stations to switch to sleep mode and conserve energy. The power state of each associated station is maintained by mac80211 (cf. Figure 1). Whenever a PS-POLL or Null packet is received by the AP, mac80211 notifies the respective driver about the change in power state via the `drv_sta_notify()` function. Whenever the driver receives an update, the Collector forwards the updated power state and the MAC address of the station to the Controller.

To reduce the overheads pertaining to channel access and PHY and MAC headers, the MAC layer of high-throughput WiFi standards (i.e., 802.11n/ac/ax) aggregates multiple MAC Protocol Data Units (MPDUs) into an Aggregated-MAC protocol data unit (A-MPDU). MonFi monitors the number of packets aggregated in each A-MPDU, the number of MPDUs currently enqueued in each queue of the driver, and the instance each packet is sent, as follows. Frame aggregation is performed in the software queues maintained in the driver. These queues act as buffers between the NIC's queue and the upper layers in the protocol stack. When there are multiple packets in a queue of the driver, they are aggregated into a single MPDU. NIC notifies the driver via a callback function after processing each packet from the hardware queue. For example, in Atheros NICs, `ath_tx_tasklet()` is called for informing the driver to process the next packet for transmission.

2) Monitoring qdisc: In Linux systems, there is a qdisc assigned to each NIC to buffer egress traffic. The Linux kernel introduces a rich set of queuing disciplines between the network subsystem in the kernel and mac80211, enabling a flexible traffic control framework. The efficiency of the qdisc layer is dependent on its packet scheduling method, the size of the queues, the rate of incoming traffic, and the rate of WiFi transmission. When the queue size is small, the qdisc layer may not be able to absorb bursts of incoming traffic while waiting for wireless transmission, thereby causing packet drops. Alternatively, longer queues may cause long end-to-end delays (a.k.a., bufferbloat [2]). Given the high impact of qdisc on packet scheduling and delay, we collect the number of packets currently enqueued in each queue of the qdisc layer. By default, every network interface is assigned a pfifo_fast qdisc as its transmission qdisc. This mechanism contains three bands, and dequeuing from a band occurs when the upper bands are empty. Each variant of qdisc is implemented as a kernel module (in `/net/sched`) and contains `.enqueue` and `.dequeue` functions. In MonFi, we modified these kernel modules to report the status of qdisc by logging the number of packets currently enqueued in each band. Our current implementation supports pfifo_fast and PRIO qdiscs. This method can easily be applied to other qdiscs such as Hierarchical Token Bucket (HTB).

C. Monitoring the Host system

In an ideal use-case, the behaviour of packet reception and transmission can be determined with the help of the data collection methods described earlier in this section. However, we need to note that the tasks performed by various components

of the networking sub-systems (e.g., mac80211, driver) and the additional processes introduced by MonFi are scheduled by the operating system to run on the processor cores available in the system. Hence, we evaluate the available and occupied computational resources. Depending on the available hardware resources, this allows the user to specify monitoring parameters (e.g., monitoring rate) that do not impose high processing overhead. Also, MonFi allows to keep track of the latency between requesting and receiving a measurement by the Controller. For real-time systems that react to network dynamics, this latency determines the validity and usefulness of the measurements obtained.

1) Dedicating computing resources to MonFi: Most modern processing platforms are based on Symmetric Multi-Processing (SMP) architecture consisting of multiple physical processor cores that are capable of executing multiple threads concurrently. User-space threads, software interrupts, and hardware interrupts are evenly scheduled to be served by processor cores. Hence, the performance of MonFi can be easily interfered by background processes and excessive context switching.

To address this problem, we bind the execution of MonFi's components with an isolated processor core, such that no other processes or interrupts are scheduled on this core. For example, consider an Intel i5 processor that includes two physical cores, PC1 and PC2, where each core contains two logical cores, LC1/LC3, and LC2/LC4, respectively. Utilizing `isolcpus=1,3` kernel boot parameter, we isolate the physical core PC1 for operating system and other background tasks. Thus, the physical core PC2 is dedicated to the execution of MonFi's components. To this end, first, LC2 is dedicated to the Controller via `sched_affinity()` or `taskset` system calls. Second, LC4 is dedicated to the driver. Since the Collector is an extension of the driver, it runs on LC4. Since all the software interrupts executed as the result of hardware interrupts are scheduled on the same core [34], [35], all the software interrupts generated by the driver are also scheduled on LC4. Third, utilizing the `/proc/interrupts` file, we obtain all possible components that can generate hardware interrupts and set the `smp_affinity` of these components to LC1 and LC3; thereby, no hardware interrupt will be scheduled on LC2 and LC4.

D. Sharing the Collected Data with User-space

The Collector shares its data with user-space for further processing. As discussed in Section II, both procfs and ioctl are less efficient compared to netlink. Therefore, we use netlink sockets for communication between kernel-space and user-space. We simply refer to this method of Collector and Controller communication as MonFi w/ netlink (MonFi-NL). To further reduce the overhead of this communication, a memory-mapped region is established for netlink's receive and transmit buffers. These buffers are shared by the Controller and Collector to prevent data copying overhead. We refer to this mechanism as MonFi w/ mmaped-netlink (MonFi-MNL).

User-space applications can use the Controller to specify three types of monitoring modes: (i) Event-based Data Collection (EDC), (ii) Polling-based Data Collection (PDC), and (iii) Event and Polling-based Data Collection (EPDC), which is a hybrid of EDC and PDC. We explain these modes as follows.

1) Event-based data collection (EDC): In this mode, the Collector sends monitoring data to the Controller whenever an event occurs. The event type is specified by the Controller. For example, sample events causing the NIC to generate an interrupt are packet reception, channel availability after Distributed Inter Frame Space (DIFS), and the expiration of software beacon alert timer (a.k.a., `bcntimer` used to trigger periodic beacon transmission). NIC interrupts are handled by the driver to determine the interrupt type and call an appropriate tasklet. For example, with Atheros NICs, `ath9k_tasklet()` and `ath_isr()` handle interrupts and then call tasklets such as `ath_tx_tasklet()` and `ath_rx_tasklet()`. We have modified the driver to monitor these interrupts and trigger data collection whenever a match in the list of events provided by the Controller is found.

2) Polling-based data collection (PDC): This mode is particularly useful in applications such as channel allocation, station handoff, packet scheduling, and intrusion detection. In this mode, the Controller collects data from the Collector in fixed or variable intervals, depending on the user-space application's demand. Variable intervals can be specified by using various distributions such as the Poisson distribution. We eliminate the overhead of user-space to kernel-space communication as follows. Measurement collection parameters are passed by the Controller to the Collector once. These parameters primarily include: (i) the Inter-Measurement Interval (IMI), either as a fixed value or the parameters of the distribution used to determine IMI, and (ii) the total number of measurements. Once received, the Collector generates reports according to IMI configuration. Also, we reduce the overhead of Collector to Controller by allowing the Controller to specify when data must be sent up. Specifically, instead of sending all measurements, the Controller can instruct the Collector to send upward reports only when the difference from the previously reported value is higher than a particular value. This threshold is specified on a per-measurement-type basis.

3) Event and Polling-based Data Collection (EPDC): In addition to the set of measurements collected at the time of an event, this mode allows for collecting the past n measurements preceding the occurrence of the event. We have implemented a circular queue in the Collector module. At each IMI, the Collector gathers and places a new measurement into the queue. Once an event occurs, the Collector sends the most recent measurement, as well as the values in the circular queue, to the Controller. This approach is particularly useful for time series analysis and neural network algorithms such as Long Short-Term Memory (LSTM), which require recent history of n measurements in the temporal domain to estimate the *trend* in a time series.

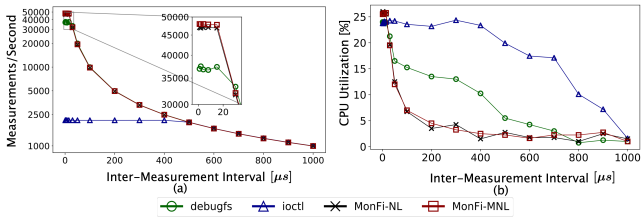

Fig. 2. (a) Measurement collection rate (per second) and (b) average CPU utilization as a function of IMI when using debugfs, ioctl, MonFi-NL, and MonFi-MNL. The monitoring rate of MonFi-MNL is 28% faster than debugfs.

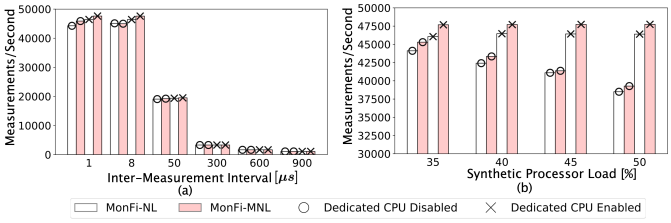

Fig. 3. (a) Measurement collection rate in the presence of a synthetic 30% processor load. (b) Maximum monitoring rate when the synthetic processor load increases from 35% to 50%. Based on these results, using dedicated cores to achieve deterministic monitoring performance is essential.

IV. Performance Evaluation

In this section, we evaluate the monitoring rate and processing overhead of MonFi when used to monitor the operation of an AP. The wireless NIC used is Compex WLE900VX, which includes the QCA9880 chipset and supports 3x3 MIMO 802.11ac. The driver used is ath10k. The processor is a dual-core Core i5. A similar hardware configuration is used by the station associated with the AP.

1) MonFi vs ioctl and debugfs: We first evaluate monitoring rate and its effect on processor utilization. Using debugfs, MonFi-NL, and MonFi-MNL, we collect the following parameters: AU, power state of the associated station, and five registers of the NIC. Using ioctl, we only collect AU. These experiments were conducted in the presence of regular CPU load, which is less than 10% and is caused by the normal functioning of the operating system tasks and AP functionality.

Figure 2(a) shows measurement collection rate and Figure 2(b) shows average CPU utilization as a function of IMI. As we reduce IMI, the monitoring rate achieved with each tool increases up to a certain point. The highest monitoring rate is provided by MonFi-MNL at 48005; whereas, ioctl, debugfs, and MonFi-NL plateau at 2115, 37481, and 47033 measurements per second, respectively. Although ioctl only collects AU, it demonstrates significantly lower monitoring rate and higher CPU utilization over all IMI values, compared to the other tools. Specifically, the monitoring rate of ioctl is 21x slower compared to MonFi-MNL. Compared to MonFi-MNL, the highest measurement collection rate provided by debugfs is 22% lower and its processor utilization is 20% higher, on average. Considering the higher performance of MonFi-NL and MonFi-MNL, we only consider these tools in the rest of studies presented in this section.

2) Impact of processor load on monitoring performance: We now evaluate how a higher processor utilization caused by a user-space daemon affects monitoring rate. This represents a scenario where a daemon is performing real-time data analysis and decision making. We utilize the stress-ng tool to generate a synthetic processor load. We evaluate using both dedicated and non-dedicated cores for the execution of MonFi processes, as explained in Section III-C1. Note that when using dedicated cores, the synthetic load is not scheduled on the cores assigned to MonFi. Figure 3 shows that reducing IMI and increasing processor load affect monitoring rate when the cores are shared. Considering MonFi-MNL, Figure 3(a) reveals that using dedicated cores increases the maximum monitoring rate by 1725 compared to shared cores. Figure 3(b) shows that using dedicated cores results in sampling rate stability versus processor load. When the synthetic load is increased from 35% to 50%, using dedicated cores achieves a constant monitoring rate of about 47000, while shared cores drops monitoring rate from 44707 to 38891, on average. These results confirm the importance of using MonFi with dedicated cores when a stable monitoring rate is required. Figure 3(b) also demonstrates the impact of using mmap-netlink instead of netlink to achieve a higher monitoring rate. For example, when the synthetic load is 35%, using MonFi-MNL collects 1621 and 1186 measurements higher than MonFi-NL with and without using dedicated cores, respectively.

3) Impact of packet switching on monitoring performance: In this section, we study the effect of packet switching on the monitoring rate of MonFi-MNL. Figures 4(a) and (b) show the results for 100-Bytes and 1400-Bytes packets being switched by the AP. These packets are received over the wireless interface (using the 802.11ac standard) and then switched to the wired interface. We were able to achieve the maximum throughput of 500 Mbps and 700 Mbps for 100-Byte and 1400-Byte packets, respectively. Reducing packet size results in a lower throughput due to the higher overhead of header transmission and waiting time caused by channel access back-off. For a given throughput rate, using smaller packets increases AP's processing overhead, which is caused by the higher number of interrupts, header processing, and packet copying operations. As both figures show, using shared cores results in a significant drop in the number of measurements per second. For example, when using 100-Byte packets, increasing the AP's switching rate from 100 Mbps to 500 Mbps causes the number of measurements per second to drop from 46508 to 42525. In contrast, using dedicated cores shows a relatively steady behaviour (less than 1% variation).

Our results confirm that MonFi can be used on average-grade dual-core APs for very high speed, efficient WiFi stack monitoring. With the denser deployment of APs and the prevalence of the 802.11ac and ax standards, the need for microsecond-level monitoring escalates.

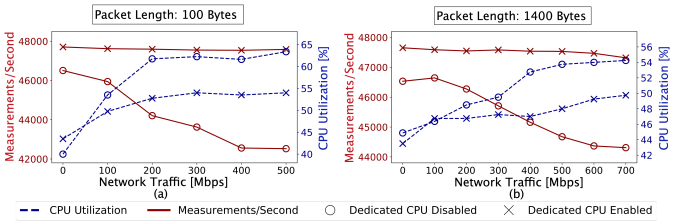

Fig. 4. Measurement collection rate (left y-axis) and CPU utilization (right y-axis) versus packet switching rate for 100-Byte (a) and 1400-Byte (b) packets. These results show that using dedicated cores is necessary to ensure deterministic monitoring rate in the presence of packet switching overhead.

V. Conclusion

With the higher number and demand of WiFi devices, supporting efficient, high-rate monitoring of the WiFi stack is an essential requirement to analyze network operation, enhance performance, and enforce security methods. In this work, we presented MonFi, which allows user-space applications to specify the type and rate of collecting measurements. Our empirical performance evaluations confirm the higher sampling rate and processing efficiency of MonFi compared to the existing Linux tools.

References

[1] WiFi Alliance. Global economic value of Wi-Fi nears $2 trillion in 2018. [Online]. Available: https://www.wi-fi.org/download.php?file=/sites/default/files/private/Value_of_Wi-Fi_Highlights.pdf

[2] T. Høiland-Jørgensen, M. Kazior, D. Täht, P. Hurtig, and A. Brunstrom, "Ending the anomaly: Achieving low latency and airtime fairness in WiFi," in *Proceedings of USENIX*, 2017, pp. 139–151.

[3] G. Nan, X. Qiao, J. Wang, Z. Li, J. Bu, C. Pei, M. Zhou, and D. Pei, "The Frame Latency of Personalized Livestreaming Can Be Significantly Slowed Down by WiFi," in *IPCCC*. IEEE, 2018, pp. 1–8.

[4] C. Pei, Y. Zhao, G. Chen, R. Tang, Y. Meng, M. Ma, K. Ling, and D. Pei, "WiFi can be the weakest link of round trip network latency in the wild," in *INFOCOM*. IEEE, 2016, pp. 1–9.

[5] S. Liu, V. Ramanna, and B. Dezfouli, "Empirical Study and Enhancement of Association and Long Sleep in 802.11 IoT Systems," in *Global Communications Conference (GLOBECOM)*, 2020.

[6] S. Y. Jang, B. Shin, and D. Lee, "An adaptive tail time adjustment scheme based on inter-packet arrival time for IEEE 802.11 WLAN," in *ICC*. IEEE, 2016, pp. 1–6.

[7] J. Schulz-Zander, C. Mayer, B. Ciobotaru, S. Schmid, and A. Feldmann, "OpenSDWN: Programmatic control over home and enterprise WiFi," in *ACM SOSR*, 2015, pp. 1–12.

[8] J. Sheth and B. Dezfouli, "Enhancing the energy-efficiency and timeliness of iot communication in wifi networks," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 9085–9097, 2019.

[9] A. J. Pinheiro, J. d. M. Bezerra, C. A. Burgardt, and D. R. Campelo, "Identifying IoT devices and events based on packet length from encrypted traffic," *Computer Communications*, vol. 144, pp. 8–17, 2019.

[10] A. Sivanathan, H. H. Gharakheili, F. Loi, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman, "Classifying IoT devices in smart environments using network traffic characteristics," *IEEE Transactions on Mobile Computing*, vol. 18, no. 8, pp. 1745–1759, 2018.

[11] E. Anthi, L. Williams, M. Słowińska, G. Theodorakopoulos, and P. Burnap, "A supervised intrusion detection system for smart home IoT devices," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 9042–9053, 2019.

[12] B. Tushir, Y. Dalal, B. Dezfouli, and Y. Liu, "A Quantitative Study of DDoS and E-DDoS Attacks on WiFi Smart Home Devices," *IEEE Internet of Things Journal*, 2020.

[13] J. Chen, B. Liu, H. Zhou, Q. Yu, L. Gui, and X. S. Shen, "QoS-driven efficient client association in high-density software-defined WLAN," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 8, pp. 7372–7383, 2017.

[14] H. Moura, G. V. Bessa, M. A. Vieira, and D. F. Macedo, "Ethanol: Software defined networking for 802.11 wireless networks," in *IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2015, pp. 388–396.

[15] B. Dezfouli, V. Esmaeelzadeh, J. Sheth, and M. Radi, "A review of software-defined WLANs: Architectures and central control mechanisms," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 431–463, 2018.

[16] B. Peck and D. Qiao, "A practical PSM scheme for varying server delay," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 1, pp. 303–314, 2014.

[17] M. Nurchis and B. Bellalta, "Target wake time: Scheduled access in IEEE 802.11 ax WLANs," *IEEE Wireless Communications*, vol. 26, no. 2, pp. 142–150, 2019.

[18] J. Sheth, C. Miremadi, A. Dezfouli, and B. Dezfouli, "EAPS: Edge-Assisted Predictive Sleep Scheduling for 802.11 IoT Stations," *arXiv preprint arXiv:2006.15514*, 2020.

[19] F. Wilhelmi, S. Barrachina-Muñoz, B. Bellalta, C. Cano, A. Jonsson, and V. Ram, "A flexible machine-learning-aware architecture for future WLANs," *IEEE Communications Magazine*, vol. 58, no. 3, pp. 25–31, 2020.

[20] A. J. Pyles, X. Qi, G. Zhou, M. Keally, and X. Liu, "SAPSM: Smart adaptive 802.11 PSM for smartphones," in *UbiComp*, 2012, pp. 11–20.

[21] Vendor-neutral, model-driven network management designed by users. [Online]. Available: https://openconfig.net

[22] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, "Network Configuration Protocol (NETCONF)," RFC 6241, 2011.

[23] Y. Han, K. Yang, X. Lu, and D. Zhou, "An adaptive load balancing application for software-defined enterprise WLANs," in *2016 International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE, 2016, pp. 281–286.

[24] T. Hühn, "A measurement-based joint power and rate controller for IEEE 802.11 networks," Ph.D. dissertation, Technische Universität Berlin, 2013.

[25] K. Sui, Y. Zhao, D. Pei, and L. Zimu, "How bad are the rogues' impact on enterprise 802.11 network performance?" in *INFOCOM*. IEEE, 2015, pp. 361–369.

[26] K. Sui, M. Zhou, D. Liu, M. Ma, D. Pei, Y. Zhao, Z. Li, and T. Moscibroda, "Characterizing and improving WiFi latency in large-scale operational networks," in *MobiSys*, 2016, pp. 347–360.

[27] C. Pei, Z. Wang, Y. Zhao, Z. Wang, Y. Meng, D. Pei, Y. Peng, W. Tang, and X. Qu, "Why it takes so long to connect to a WiFi access point," in *INFOCOM*. IEEE, 2017, pp. 1–9.

[28] A. Showail, K. Jamshaid, and B. Shihada, "Buffer sizing in wireless networks: challenges, solutions, and opportunities," *IEEE Communications Magazine*, vol. 54, no. 4, pp. 130–137, 2016.

[29] K.-Y. Jang, S. Hao, A. Sheth, and R. Govindan, "Snooze: energy management in 802.11n WLANs," in *CoNEXT*, 2011, pp. 1–12.

[30] Y. Xiao, Y. Cui, P. Savolainen, M. Siekkinen, A. Wang, L. Yang, A. Ylä-Jääski, and S. Tarkoma, "Modeling energy consumption of data transmission over Wi-Fi," *IEEE Transactions on Mobile Computing*, vol. 13, no. 8, pp. 1760–1773, 2013.

[31] P. Neira-Ayuso, R. M. Gasca, and L. Lefevre, "Communicating between the kernel and user-space in Linux using Netlink sockets," *Software: Practice and Experience*, vol. 40, no. 9, pp. 797–810, 2010.

[32] G. Cena, S. Scanzio, and A. Valenzano, "SDMAC: a software-defined MAC for Wi-Fi to ease implementation of soft real-time applications," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, pp. 3143–3154, 2018.

[33] J. Malinen. (2020) hostapd: IEEE 802.11 AP, IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authenticator. [Online]. Available: https://w1.fi/hostapd/

[34] S. Muramatsu, R. Kawashima, S. Saito, and H. Matsuo, "VSE: Virtual switch extension for adaptive CPU core assignment in softirq," in *IEEE 6th International Conference on Cloud Computing Technology and Science*, 2014, pp. 923–928.

[35] P. Mejia-Alvarez, L. E. Leyva-del Foyo, and A. Diaz-Ramirez, "Interrupt handling in classic operating systems," in *Interrupt Handling Schemes in Operating Systems*. Springer, 2018, pp. 15–25.