2023 IEEE Wireless Communications and Networking Conference (WCNC) | 978-1-6654-9122-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/WCNC55385.2023.10119047

Traffic Characterization for Efficient TWT Scheduling in 802.11ax IoT Networks

Jaykumar Sheth*, Vikram K. Ramanna*†, and Behnam Dezfouli*†
*Internet of Things Research Lab, Department of Computer Science and Engineering, Santa Clara University, USA
†Infineon Technologies, San Jose, USA
{jsheth, vramanna, bdezfouli}@scu.edu

***Abstract*—To reduce packet collisions and enhance the energy efficiency of stations, Target Wake Time (TWT), which is a feature of the 802.11ax standard (WiFi 6), allows the allocation of communication service periods to stations. While effective TWT allocation requires characterizing the traffic pattern of stations, in this paper, we empirically study and reveal that the existing methods (i.e., channel utilization estimation, packet sniffing, and buffer status report) do not provide adequate accuracy. To remedy this problem, we propose a traffic characterization method that can accurately capture inter-packet and inter-burst intervals on a per-flow basis in the presence of factors such as channel access and packet preparation delay. We empirically evaluate the proposed method and confirm its superior traffic characterization performance against the existing ones. We also present a sample TWT allocation scenario that leverages the proposed method to enhance throughput.**

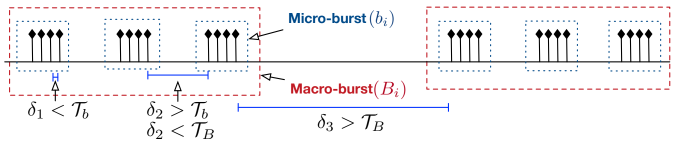
***Index Terms*—Energy efficiency, Throughput, Target Wake Time (TWT), WiFi 6, In-band Network Telemetry (INT), Buffer Status Report (BSR).**

I. Introduction

There will be about 14.7 billion Internet of Things (IoT) connected devices in 2023, up from 6.1 billion in 2018 [1]. The adoption of WiFi technology, especially for IoT connectivity, is gaining momentum and enables dense deployments due to the following reasons: First, WiFi technology provides higher communication rates compared to technologies such as Bluetooth and ZigBee. Second, WiFi communication utilizes unlicensed spectrums, thereby, deployments are considerably less expensive than cellular technologies. Third, the distributed and customer-oriented deployment of WiFi networks in residential and enterprise settings reduces deployment costs and offers an omnipresent infrastructure for connectivity.

To reduce deployment costs and facilitate mobility, many IoT devices rely on limited energy resources such as batteries or energy harvesting. For such devices, the standard offers various power-saving modes, including Power Save Mode (PSM), Adaptive PSM (APSM), and Automatic Power Save Delivery (APSD). The effectiveness of these methods, however, is highly affected by channel access contention and buffering delay in Access Point (AP) [2]–[4]. The newly-introduced 802.11ax standard provides a method called Target Wake Time (TWT) for assigning *Service Periods (SPs)* to stations (STAs). Compared to the earlier power-saving modes, TWT allows for potentially higher energy efficiency and throughput. Specifically, by properly assigning SPs to STAs, channel contention is reduced, packet buffering delay in the AP drops, and packet aggregation efficiency enhances [5]. Nevertheless, to realize the benefits of TWT scheduling, accurate characterization of the traffic flows of STAs is required by an AP to allocate SPs that address STAs' traffic requirements [5]–[7]. To this end, the most-widely used methods are Channel Utilization (CU) estimation, packet sniffing, and Buffer Status Report (BSR) [8], [9]. However, as we will show in this paper, none of these methods provide high accuracy, especially in the presence of channel access delay. Another approach is to gradually and dynamically adjust TWT assignments over time, based on STAs' demands. This approach, nonetheless, introduces additional communication overhead for TWT scheduling and cannot quickly address dynamic changes in traffic patterns [9]. The existing works also include scheduling methods that rely on specific assumptions regarding STAs' traffic pattern [10], or they propose various methods to classify STAs' traffic pattern (e.g., using machine learning) to determine TWT scheduling parameters. The primary shortcoming of traffic classification methods is imposing high processing overhead on STAs, AP, or both. Due to the challenges of traffic characterization, we observe that the existing works assign each service period to one STA only [9], [11], thereby they cannot fully utilize available channel time.

In this paper, we argue that accurate characterization of the traffic pattern of STAs is necessary to allocate TWT SPs that result in both high throughput and energy efficiency. For example, consider an IoT device collecting a batch of sensor samples every few seconds. The process of sample collection from an Analog-to-Digital Converter (ADC), packet preparation, and transfer from the application subsystem to the Network Interface Card (NIC) introduces a non-negligible inter-packet interval that would result in bandwidth waste if the time period is not utilized by other STAs. We study and reveal that the existing traffic characterization methods demonstrate the following shortcomings: (i) CU provides only the accumulated channel time usage by all the STAs; (ii) packet sniffing approach is affected by channel access delay and collision; (iii) BSR is affected by channel access delay; (iv) some devices generate BSR only when requested by the AP; (v) reported BSR values are fixed for all the MAC Protocol Data Units (MPDUs) inside an Aggregated MPDU (A-MPDU). To address these shortcomings, we propose Source-assisted Traffic Characterization (SATRAC), a method based on In-band Network Telemetry (INT). We leverage the eBPF


Micro-burst (b_i)
Macro-burst (B_i)
$\delta_1 < \mathcal{T}_b$
$\delta_2 > \mathcal{T}_b$
$\delta_2 < \mathcal{T}_B$
$\delta_3 > \mathcal{T}_B$
Fig. 1. Micro-burst and macro-burst characterization.

technology to add the difference between packet generation time instances to the TCP Options field (for TCP traffic) or IP Options (for UDP traffic). This information allows the AP to accurately and quickly determine traffic generation patterns and assign TWT SPs to STAs. We compare the performance of SATRAC against the existing methods and show its superior performance and robustness against factors such as CU and channel access delay.

The rest of this paper is organized as follows. Motivation for traffic characterization is given in Section II. Section III studies the shortcomings of existing methods. We present the design and evaluation of SATRAC in Section IV. We conclude the paper in Section V.

II. Traffic Pattern Analysis

This section analyzes the characteristics of network traffic generated in real-world IoT scenarios. We separate micro-bursts and macro-bursts and justify the importance of traffic characterization for TWT assignment.

We study the following IoT scenarios: (i) **Sensing**: We use an RTOS development kit (CYW54907) for collecting accelerometer readings. The device periodically collects 3920 samples, equivalent to 5880 bytes $((3920 \times 12 \text{ bits})/8)$, prepares packets, and then sends them via a TCP connection. (ii) **Camera**: We built a security camera using Raspberry Pi (RPi) camera module (version 2) attached to a RPi 3B+. The camera continuously captures and sends images via a TCP connection. Each images is processed by the H.264 codec. (iii) **Video Streaming**: A YouTube video is streaming on an Amazon Echo Show device. All the experiments were run in interference-free environments.

To build a generalized traffic analysis framework, we consider three inter-packet intervals and use the traffic structure of Figure 1. A *micro-burst*, denoted as b_i, is defined as a train of packets with inter-arrival time less than a specific threshold value $\mathcal{T}_b$ [4]. The interval between packets in a micro-burst is denoted as δ_1. If the interval between two packets is larger than $\mathcal{T}_b$ and less than $\mathcal{T}_B$, a new micro-burst is detected. The interval between micro-bursts is denoted as δ_2. If the interval between packets is larger than $\mathcal{T}_B$, a new macro-burst is detected. A macro-burst is represented as B_i, and the interval between macro-bursts is denoted as δ_3.

Figure 2 presents the results for the Sensing scenario. We observe that even within a micro-burst, the mean interval between packets (δ_1) is about 400 μs. This delay is primarily caused by packet preparation delay, as well as the timing requirements of the 802.11 standard (e.g., channel access contention, SIFS, DIFS). Regarding packet preparation delay,

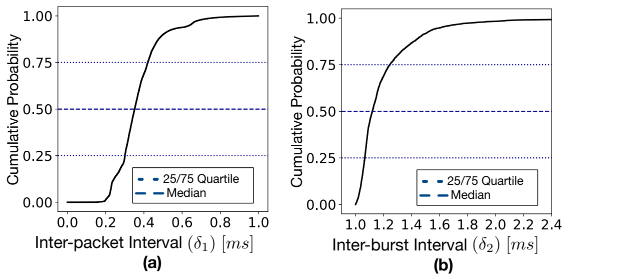


Fig. 2. Inter-packet intervals for the Sensing scenario. δ_1 is primarily caused by packet preparation delay. The difference between δ_1 and δ_2 is mainly caused by the delay of collecting 3920 samples.

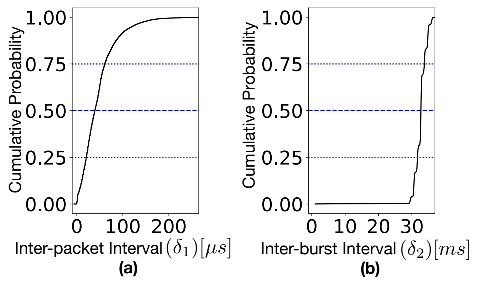


Fig. 3. Inter-packet intervals for the Camera scenario. δ_2 is about 33 ms, corresponding to 30 frames per second.

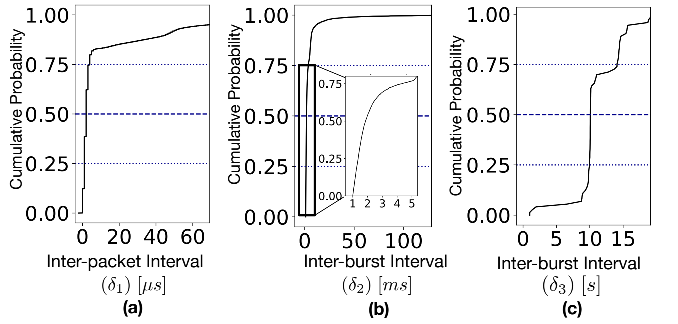


Fig. 4. Inter-packet intervals for the Video Streaming scenario.

we modified the code and added probes to each stage of the packet preparation process and observed that, for example, the transmission of a packet from driver to NIC introduces a non-negligible delay of about 28 μs. Comparing Figures 2(a) and (b), the interval between micro-bursts is due to collecting 3920 samples. Specifically, this delay is caused by the communication between the processor and ADC over the Serial Peripheral Interface (SPI) to collect samples [12]. Therefore, δ_2 would increase if a higher number of samples were collected per round.

Figure 3 shows the inter-burst intervals for the Camera scenario. Each micro-burst constitutes multiple packets. The camera captures a frame and then prepares multiple packets to send the frame. The amount of data in each frame depends on the resolution of the video stream requested (e.g., 480p, 720p, 1080p). As we see in Figure 3(b), the interval between micro-bursts (δ_2) is about 33 ms, which corresponds to 30 frames per second. Figure 4 shows the results for video streaming scenario. The mean interval between macro-bursts (δ_3) is 10 seconds, the mean interval between micro-bursts (δ_2) is 2 ms, and the mean interval between packets of a micro-burst (δ_1) is 9 μs.

These studies demonstrate the intervals between packets in a micro-burst, the intervals between micro-bursts, and the interval between macro-bursts. Characterizing these delays is essential for three purposes: (i) allocating TWT SPs based on each STA's demands, (ii) utilizing inter-packet intervals by other STAs to enhance throughput, and (iii) enhancing packet aggregation performance, which results in shorter communication delays and higher energy efficiency. For example, suppose a TWT service period ends before sending all the packets of a micro-burst. In this case, the STA either needs to wait for the next TWT service period (causing communication delay) or contend with other STAs for channel access (lower energy efficiency).

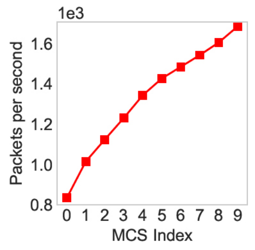


Fig. 5. Packets per second

To confirm the feasibility of utilizing inter-packet intervals within a micro-burst by other STAs, we measure the actual packet (1500 bytes) transmission rate in an 802.11ax testbed communicating over a 20 MHz channel. As Figure 5 shows, using Modulation and Coding Scheme (MCS) 5, a STA can send about 1427 packets/second, which means the duration of each packet transmission (including backoff and channel access) is around 700 μs. Therefore, a station sharing a SP with other stations can utilize inter-packet and inter-burst intervals of other station to transmit its packets.

III. Traffic Characterization via Channel Utilization, BSR, and Packet Sniffing

In this section, we study the shortcomings of the three available and most-widely used traffic characterization methods.

A. Channel Utilization (CU)

CU is defined as $t_{activity}/t_{total}$, where $t_{activity}$ is the time duration the NIC has sensed signal power higher than a prespecified threshold value during time duration $t_{overall}$. CU values can be collected from the driver via various methods such as the 'proc' file system (`procfs`) in Linux. However, since the information provided by CU is cumulative, it cannot be used to characterize per-STA traffic patterns.

B. Packet Sniffing

Several real-world deployments and Commercial Off-The-Shelf (COTS) enterprise APs utilize an external NIC operating as a sniffer to monitor the traffic patterns of STAs [8]. Since each AP acts as the central point of communication for all the traffic to and from the STAs, collecting an AP's driver logs can also be utilized for determining the traffic pattern of STAs.

The shortcomings of this approach are as follows. First, the timestamps of sniffed packets do not represent the actual packet generation instances by STAs. This is due to factors such as channel access contention delay, internal prioritization of packets belonging to different ACs, and packet preparation delay. Second, in addition to requiring extra hardware (e.g., additional NIC), the sniffed packets must be processed by the AP's operating system and user application; thereby increasing processing overhead. Third, any inconsistency between the hardware and antenna configurations of the AP's primary NIC and those of the sniffer results in a mismatch between the sniffed packets and those exchanged by the AP's primary NIC [13]. We will further analyze this method in Section IV.

C. Buffer Status Report (BSR)

In the 802.11ac standard, the Queue Size (QS) sub-field, contained inside the QoS Control field, reports the total data queued in the STA's queues. The AP primarily utilizes this information to allocate Transmit Opportunity (TXOP) to each STA. BSR is a new functionality introduced by the 802.11ax standard to enhance the exchange of information on the transmission buffer size of STAs. For example, compared to the QS field, BSR provides more specific information, such as the Queue size of the highest-priority Access Category (AC). The Queue Size All (QSA) field of BSR conveys the cumulative amount of data in all queues.

In this section, we reveal the challenges of using BSR for traffic characterization. First, the 802.11ax standard does not mandate the inclusion of queue statistics in each packet. To verify this, we selected several COTS 802.11ax NICs and noticed that Intel AX200 and Realtek RTL8852A transmit BSR intermittently, based on the amount of traffic queued. In contrast, Compex WLT639 includes a BSR in every packet. Additionally, none of the evaluated APs and STAs support requesting or generating BSR manually.

Secondly, by empirical analysis of packet exchange traces, we observed that for those 802.11ax devices that include BSR in each packet, all the MPDUs included in an A-MPDU report the same value, even though the payloads they are carrying have been generated at different time instances. The reported value is the state of queues before the transmission of A-MPDU. To demonstrate this behavior, we captured BSR values by a STA using Compex WLT639. Also, to generate various A-MPDU sizes, we gradually increase the amount of data pushed by the application to the transport layer socket. Figure 6 shows the number of packets per A-MPDU (left y-axis) and the QSA values in BSR (right y-axis). The squares denote the number of MPDUs in the succeeding A-MPDU, and red curve denotes QSA value reported by each incoming packet. As the results show, the BSR value reported per A-MPDU is fixed and reports the amount of data in the driver's buffer plus the size of A-MPDU being sent. In general, assume the QSA (denoted as Q) values are received at time instances t_n and t_{n+1} from a STA. The amount of traffic generated during this interval can be represented as: $Q_{t_{n+1}} - Q_{t_n} + \sum_{[t_n,t_{n+1})} p_{tx}$, where $Q_{t_{n+1}}$ and Q_{t_n} are the received BSR values at time t_{n+1} and t_n, respectively, and $\sum_{[t_n,t_{n+1})} p_{tx}$ is the sum of the size of packets transmitted by the STA during time interval $[t_n, t_{n+1})$, which excludes the packet received at t_{n+1}. In summary, the

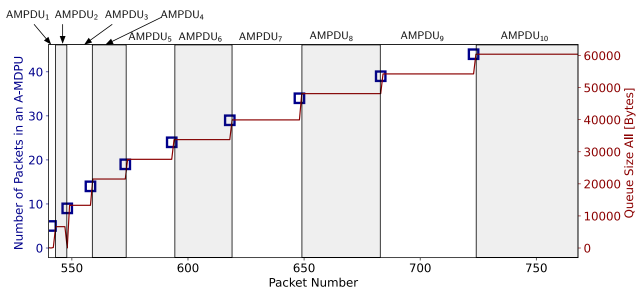


Fig. 6. Traffic characteristics and BSR values for a flow with increasing burst size. The blue squares denote the number of packets in the succeeding A-MPDU, and the red curve represents BSR's QSA field of packets received by the AP. The results show that the reported buffer size is fixed for all the MPDUs inside an A-MPDU.

BSR approach neither provides the actual data generation time instances nor provides real-time snapshots of driver's queue sizes.

Since BSR does not provide any timing information, we augment this method as follows to estimate packet generation time instances. In this paper, we refer to this method as BSR$'$ and evaluate it in Section IV-C. Assume the BSR values received from two packets at time instances t_n and t_{n+1} are Q_{t_n} and $Q_{t_{n+1}}$. If $Q_{t_{n+1}} > Q_{t_n}$, we estimate inter-packet generation instances during interval t_{n+1} to t_n as $(t_{n+1} - t_n)/((Q_{t_{n+1}} - Q_{t_n} + \sum_{[t_n, t_{n+1})} p_{tx})/\bar{p_{tx}})$, where $\bar{p_{tx}}$ is the average size of packets received during this interval. If $Q_{t_{n+1}} < Q_{t_n}$, we use the time instance of sniffed packets.

IV. Source-assisted Traffic Characterization

In this section, we propose a method named Source-assisted Traffic Characterization (SATRAC) and evaluate it in terms of traffic characterization accuracy and its effect on TWT allocation.

A. Design and Implementation

The basic idea of SATRAC is that, if we keep track of packet generation time instances in each STA, the AP can construct the traffic pattern of the STA, regardless of the effect of packet preparation delay, channel access contention, interference, and packet loss. To this end, we require each STA to modify packets in their protocol stack's data-path and add timing information—an approach similar to INT. In order to reduce packet overhead, instead of including an absolute timestamp in each packet, we include only a 2-byte value encoding the difference between the generation time of the current packet and the previous packet of the same flow. To this end, each STA computes a unique 5-tuple hash value for each flow and keeps track of the timestamp of the last generated packet. In this paper, to simplify compatibility with existing implementations, we chose the TCP header's Options field to include timing information. Alternatively, for non-TCP traffic, the timing information can be added to the IP header's Options field of IPv4 or a 'next header' field for IPv6.

In order to add timing information to packets, we consider two approaches, as follows.

1) Packet Modification in the MAC or NIC: As explained in Section II, packet preparation increases inter-packet delay. To capture packet processing delay, we need to add timing information to each packet when it is ready to be sent; therefore, we need to add the timing information when the packet arrives in the NIC. However, since modifying NIC's firmware is infeasible, the alternative is to add timing information when MAC processing completes and add driver-to-NIC handoff delay as a constant value to this delay. The challenge with this approach is that any modification to the TCP header requires recalculation of TCP checksum and MAC checksum, and any changes to the IP header requires MAC checksum recalculation; in both cases, packet preparation overhead is unnecessarily increased.

2) Packet Modification in the TCP Layer: To eliminate the need for checksum recalculation, we add timing information in the TCP layer when the TCP protocol prepares the TCP header. To account for packet preparation delay, we add the delays caused by the IP layer, MAC layer, and driver-to-NIC handoff to the timing information.

A straightforward approach to modifying the TCP Options field on Linux systems is to use `setsockopt`; however, only a specific set of options can be modified with this API. An alternative is to craft a raw packet with appropriate TCP Options fields hardcoded; nevertheless, this method requires modifying the applications. Instead of these two approaches, we leverage eBPF and build an application-agnostic middleware for setting the TCP Options field (note that this approach can be used for setting the IP fields as well). This eBPF module can easily be executed from the user-space on each STA to embed the timing information without any modifications to the kernel's code-base. Also, this approach eliminates the need to modify applications, as it acts as a shim layer between applications and the transport layer. eBPF enables real-time patching of the Linux kernel by allowing users to insert user-defined logic (programs) into the kernel. eBPF programs are associated with hook points that are triggered on the execution of either a *syscall* or a kernel function. We use `BPF_PROG_TYPE_SOCK_OPS` program type that allows the modification of socket options on a per-packet basis. When an event, such as `sendmsg` call, TCP connection, or TCP retransmit timeout occurs, `bpf_sock_ops` structure is returned, which provides the context of the event along with the "op" field identifying the source of the event. We hook the eBPF program to the `tcp_write_options` function, which is responsible for adding the TCP Options field. For measuring packet preparation delay, similarly, we use eBPF hooks in the TCP and MAC layer. For driver-to-NIC delay, the driver is modified to measure the duration of Direct Memory Access (DMA) transactions.

Since APs usually run Linux, a similar eBPF program extracts and parses the values included in TCP Options field of packets received from STAs to characterize uplink traffic. For characterizing downlink traffic, the same AP module keeps track of the packet arrival instances on the AP's wired NIC for each STA. In this paper, we primarily address characterizing uplink traffic.

The implementation of SATRAC on RTOS-based STAs depends on the protocol stack used. On a platform using ThreadX with NetXDuo stack, we simply added timers to the TCP and driver codes to measure packet preparation delay. Also, to embed the timing information in TCP Options field, we modified the TCP code.

B. Analytical Comparison

We first analyze the benefits of SATRAC by presenting a simple analytical scenario, demonstrated in Figure 7. Row (a) shows that at each time instance t_1, t_2, t_3 and t_4, the application generates ψ bytes. The interval between data generation instances is denoted as Δ. Row (b) shows that each ψ-byte message is segmented (by transport layer) into three packets. The interval between these packet generations, denoted as δ_1, depends on packet preparation delay. The third (c) and fourth rows (d) present the actual packet transmission instances (note that the time instances are similar).

When using BSR (row (c)), the STA includes in each packet the amount of data in the buffer. For example, the packet generated at time t_1 is transmitted at time $t_{2,1}$, and this packet indicates only the buffer size at the beginning of packet transmission, which is 2ψ. Since the packet does not convey packet generation time, this data could have been generated any time during the time interval between the transmission of this packet and the previous packet. Additionally, if the four packets transmitted sequentially at time instance $t_{3,1}$ are aggregated as an A-MPDU, all these four packets report value 2ψ (as explained in Section III-C), further affecting the accuracy of traffic characterization. Similarly, the packet sniffing method cannot be used for identifying packet generation times. Specifically, as it can be observed, the packet transmission instances do not represent packet generation instances. Also, note that BSR and packet sniffing methods cannot be used to determine a STA's required CU during a specific time period because we need to know the interval during which packets have been generated to calculate CU.

Using SATRAC, the difference between packet generation timestamps is added to each packet. Regardless of packet transmission time, the first packet of each micro-burst includes the time stamp $\Delta - 2\delta_1$, and the second and third packets include timestamp δ_1. This method allows the AP to determine both Δ and δ_1 to characterize traffic accurately.

C. Empirical Evaluations of SATRAC

In this section, we empirically compare the performance of SATRAC versus packet sniffing (Section III-B) and BSR$'$ (Section III-C). We use an 802.11ax testbed including one AP and multiple STAs running Linux. We characterize the accuracy of traffic characterization for one STA. Other STAs are used to introduce variations in CU. We consider two CU scenarios: (i) *low*, where the measured CU is around 15%, and (ii) *high*, where the measured CU is around 70%. We evaluate the accuracy of SATRAC when inter-packet intervals are small. To this end, a STA runs a program that generates

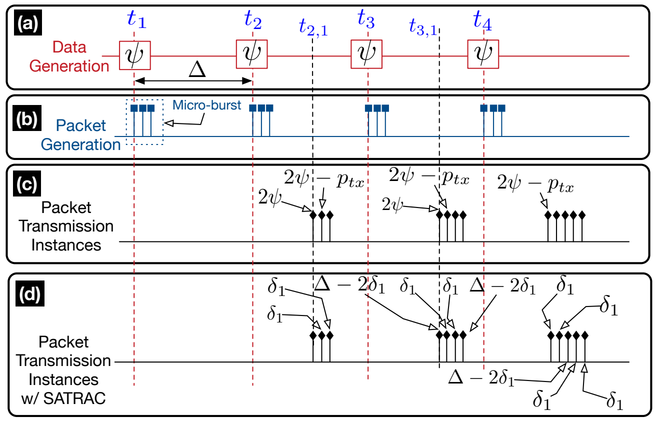


Fig. 7. (a): Data generation time instances by application. (b): Packet generation time instances. (c): Packet transmission instances. This row shows the time instance of packet capture by sniffer, as well as the BSR value of each packet. (d): Packet transmission instances and timing information added to each packet by SATRAC.

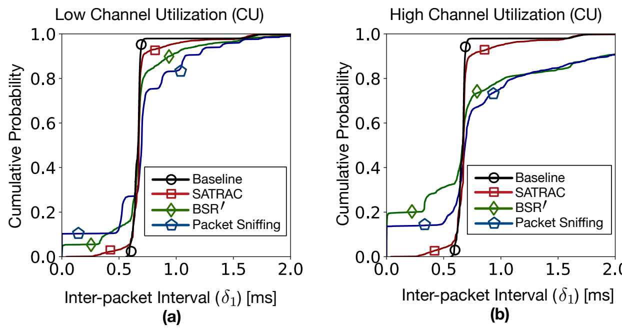


Fig. 8. Empirical evaluation of SATRAC versus the baseline (actual data generation time instances), packet sniffing, and BSR$'$ in low (a) and high (b) CU scenarios. SATRAC demonstrates the highest accuracy even in the presence of high CU.

and sends a 1400-byte message every 500 μs. Since TCP employs a buffering delay of about 200 ms to accumulate data before transmission, we utilize the `TCP_NODELAY` flag for the TCP socket to send the generated data as soon as received from the application. Also, we are using the voice AC for the transmission of generated packets. Note that this AC does not employ packet aggregation (i.e., A-MPDU). To establish a baseline for accuracy comparison, we denote the actual data generation instances by the application as *baseline*.

Figure 8 presents the results collected in low and high CU scenarios. We can observe that the baseline curve is not a vertical line. The variations of the baseline are caused by multiple factors, including timer inaccuracy, context switching, the delay of copying data from the user-space to the kernel-space, and the delay of logging time stamps. The closest curve to the baseline curve is that of SATRAC, in both low and high CU scenarios, therefore demonstrating the high accuracy of this approach. Note that some of the timing inaccuracies affecting the baseline also affect SATRAC; context switching delay and copying data from user-space to kernel-space are

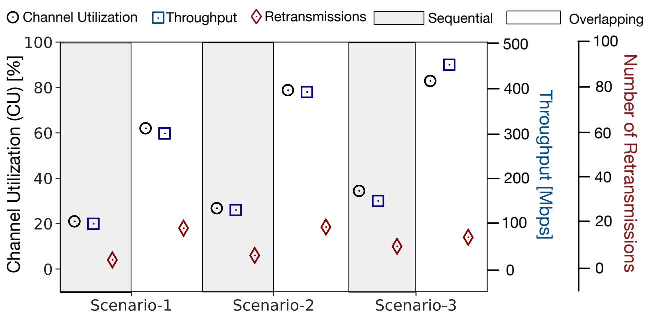


Fig. 9. *Sequential* and *overlapping* TWT allocations to three STAs. Throughput, CU, and number of retransmissions are per second. The three scenarios on the x-axis refer to incrementally higher CU levels by the STAs. By leveraging SATRAC, the AP can assign each SP to multiple STAs to enhance both CU and throughput.

among the factors. Nevertheless, while both packet sniffing and BSR′ are considerably affected in the high CU scenario, the accuracy of SATRAC remains unaffected. We also observe that although the accuracy of traffic characterization is slightly improved by the BSR′ method, the accuracy of this method is considerably affected by increasing CU.

D. Sample TWT Allocation Scenario

To show the benefits of utilizing SATRAC for TWT allocation, we use a testbed including three STAs. First, similar to the existing works [9], [11], we assign non-overlapping SPs to the STAs; we refer to this approach as 'sequential' allocation. Then, we enable the AP to characterize traffic, determine the possibility of higher channel utilization, and assign overlapping SPs to the STAs; this is referred to as 'overlapping' allocation. By changing the packet generation instances of each STA, we adjust inter-packet intervals and introduce various CU levels, corresponding to three scenarios. The results are presented in Figure 9. We observe that the overlapping allocation enhances throughput and CU, while the number of retransmissions (caused by collisions) are slightly increased. For instance, in Scenario-3, where the mean per-STA CU is 36% using the sequential allocation, the overlapping allocation increases the mean to 86%. Similarly, the throughput increases from 145 Mbps to 445 Mbps in overlapping allocation compared to sequential allocation. Although the overlapping allocation increases the mean number of retransmissions per second from 10 to 14, these are 0.00025% and 0.00035% of the total number of transmissions per second, respectively. Therefore, these results confirm the effectiveness of accurate traffic characterization for TWT allocation.

V. Conclusion

Allocation of TWT SPs to IoT STAs requires accurate traffic characterization to meet applications' demands while enhancing energy efficiency and throughput. In this paper, we empirically studied traffic burstiness and the causes of inter-packet delays in WiFi-based IoT networks. We analyzed the shortcomings of existing traffic characterization methods and introduced a novel approach based on packet modification in STAs' protocol stack. We showed that using eBPF to embed inter-packet generation times in TCP header (or IP header) provides an effective solution for determining per-flow traffic patterns in AP.

While in this paper we focused on traffic characterization and TWT allocation in the time domain, the proposed method can be used to enhance the allocation of Resource Units (RUs) to STAs in 802.11ax networks. In particular, by enhancing the accuracy of conveying STAs' demands to the AP, more efficient time and frequency (TWT and RU) allocation algorithms can be developed. This is left as future work. We also note that the proposed method can be leveraged by data-driven and machine learning methods to enhance the accuracy of device and traffic characterization, which can be used for applications such as network security.

Acknowledgment

The authors would like to thank Infineon Technologies and Arista Networks for donating some of the materials used to conduct this research. This work was supported by NSF grant 2138633.

References

[1] Cisco Systems. (2020) Cisco annual internet report (2018–2023) white paper.
[2] 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, 2015.
[3] 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.
[4] J. Sheth, C. Miremadi, A. Dezfouli, and B. Dezfouli, "EAPS: Edge-assisted predictive sleep scheduling for 802.11 IoT stations," *IEEE Systems Journal*, vol. 16, no. 1, pp. 591–602, 2022.
[5] 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.
[6] Q. Chen and Y.-H. Zhu, "Scheduling channel access based on target wake time mechanism in 802.11 ax wlans," *IEEE Transactions on Wireless Communications*, vol. 20, no. 3, pp. 1529–1543, 2020.
[7] Q. Chen, G. Liang, and Z. Weng, "A target wake time based power conservation scheme for maximizing throughput in ieee 802.11 ax wlans," in *IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2019, pp. 217–224.
[8] A. Bhartia, B. Chen, D. Pallas, and W. Stone, "Clientmarshal: Regaining control from wireless clients for better experience," in *The 25th Annual International Conference on Mobile Computing and Networking*, 2019, pp. 1–16.
[9] C. Yang, J. Lee, and S. Bahk, "Target wake time scheduling strategies for uplink transmission in ieee 802.11 ax networks," in *IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2021, pp. 1–6.
[10] B. Schneider, R. C. Sofia, and M. Kovatsch, "A proposal for time-aware scheduling in wireless industrial iot environments," in *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2022, pp. 1–6.
[11] W. Qiu, G. Chen, K. N. Nguyen, A. Sehgal, P. Nayak, and J. Choi, "Category-based 802.11 ax target wake time solution," *IEEE Access*, vol. 9, pp. 100 154–100 172, 2021.
[12] C.-C. Li, V. K. Ramanna, D. Webber, C. Hunter, T. Hack, and B. Dezfouli, "Sensifi: A wireless sensing system for ultrahigh-rate applications," *IEEE Internet of Things Journal*, vol. 9, no. 3, pp. 2025–2043, 2022.
[13] L. Song, A. Striegel, and A. Mohammed, "Sniffing only control packets: A lightweight client-side wifi traffic characterization solution," *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6536–6548, 2020.