IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 16, 2021, accepted June 27, 2021, date of publication July 13, 2021, date of current version July 21, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3096940

Category-Based 802.11ax Target Wake Time Solution

WENXUN QIU[1], (Member, IEEE), GUANBO CHEN[1], (Member, IEEE), KHUONG N. NGUYEN[1], (Member, IEEE), ABHISHEK SEHGAL[1], (Member, IEEE), PESHAL NAYAK[1], (Member, IEEE), AND JUNSU CHOI[2], (Member, IEEE)

[1]Standards and Mobility Innovation Laboratory, Samsung Research America, Plano, TX 75023, USA
[2]Samsung Electronics Company Ltd., Suwon 16677, South Korea

Corresponding author: Guanbo Chen (guanbo.h@samsung.com)

ABSTRACT IEEE 802.11ax newly introduced Target Wake Time (TWT) function which enables joint reduction of device power consumption and wireless medium congestion through negotiated TWT wake interval and duration. We develop a novel solution which can dynamically configure station (STA) wake interval and duration based on run time recognition of the Quality of Service (QoS) requirement including the required latency and throughput. Specifically, a Machine Learning (ML) based Network Service Detection (NSD) module is developed which can detect the current network service in real time and update the STA wake interval according to the corresponding service latency requirement. Moreover, a novel data time estimation model is developed which can estimate the required throughput and wake duration with the observed throughput, linkspeed, and contention level. In addition, a state-machine based method is developed to detect three different traffic types (random, stable, bursty) and their throughput variation pattern for further optimization of the STA wake duration. Our solution is implemented and extensively tested on commercial smart mobile platform under various network conditions and use cases. The results show that our ML based NSD could reach 99.2% and 96.5% accuracy respectively for coarse-grain and fine-grain network service classification, which ensures our TWT solution can accurately recognize and satisfy the QoS requirements. More importantly, while maintaining the QoS and user experience, our solution can substantially reduce the Wi-Fi duty cycle to 29.6% on average, which leads to lower device power consumption and network contention level.

INDEX TERMS 802.11ax, target wake time, machine learning, service classification, traffic detection.

I. INTRODUCTION

Target Wake Time (TWT) is a power saving mechanism adopted in 802.11ax amendment [1]. TWT was first introduced by 802.11ah, which targets on low power consumption for Internet of Things (IoT) devices. TWT enables Station (STA) and Access Point (AP) to negotiate for wake interval and duration. After successful TWT negotiation, STAs are allowed to enter doze state after the negotiated wake duration, which can reduce the power consumption of STA and the contention level in a wireless local area network (WLAN). Implementing TWT on IoT devices would be straightforward in that they usually have clear latency and throughput requirements which are consistent over time. On the other hand, implementing TWT on smart mobile devices such as smart phones, tablets, laptops is complicated. Users run different types of services on smart mobile devices, which have different latency and throughput requirements according to the running services. Moreover, users may switch to different services in realtime causing time-varying requirements on latency and throughput. In this scenario, having an adaptive wake interval and duration control solution is very important. However, this problem has not been addressed in the previous literature.

In the literature, several works have been proposed to leverage TWT feature to reduce the collision using scheduling [2]–[4], which leads to improvement on aggregate network-wise throughput. However, to the best of our knowledge, there does not exist an adaptive TWT control solution which can reduce the power consumption while maintaining the Quality of Service (QoS) according to the dynamically varying running services and traffic statistics. In cellular

The associate editor coordinating the review of this manuscript and approving it for publication was Yulei Wu.

system, a power-saving feature called discontinuous reception (DRX) [5] in LTE and 5G is introduced to reduce the User Equipment (UE) power consumption by configuring the sleep/wake-up scheduling. The methods developed for DRX have good potential to apply to TWT with some necessary changes. In [6]–[10], semi-Markov and Markov chain models are used to evaluate the DRX latency and power saving performance for a given latency requirement. In [11], two different configurations for DRX were proposed which can minimize power consumption and assure the required latency respectively. Another method to adapt the DRX parameters by tracking the queue delays was proposed in [12], which could provide bounded latency. However, all the works above only solves the problem with a constraint that the latency requirements need to be known and given to the algorithm in prior, which is obviously not practical in real systems that the running services and the required latency are dynamically changing. The latency requirements are tightly related to the running network services. For example, power saving and QoS impact in DRX were analyzed just for VOIP service in [13]. This shows the necessity of an adaptive TWT control solution with the automatic detection of running network service types and then the estimation of the corresponding latency. For the detection of service types, there are some existing work that classify the service types according to the traffic statistics. In [14], K-mean algorithm was used to identify different services in messaging application. A Naive Bayes classifier was used in [15] to differentiate Android malware and normal traffic. In [16], ensemble learning methods were used to identify different mobile applications. In [17], a multi-level classifier was used to distinguish known and unknown applications. These works either are constrained in certain types of service or classify the service for a different purpose other than deriving the required latency.

In this paper, we propose an adaptive TWT control solution which can reduce the power consumption and maintain the QoS with dynamically changing network services and latency requirements. A Machine Learning (ML) based Network Service Detection (NSD) module is proposed to detect the running service types and map to their corresponding required latency. Consequently, it can adaptively configure wake interval according to the detected service type. Moreover, we propose a dynamic wake-duration control algorithm to accommodate time varying traffic throughput and network conditions. In addition, the wake duration configuration strategy is optimized according to three different traffic patterns, which are detected by a state-machine based algorithm. Lastly, we implement this solution on commercial smart mobile devices, and evaluate it with commercial APs. Extensive tests validate that this solution can recognize running network service with high accuracy and can automatically configure the TWT wake interval and duration to reduce the WiFi duty cycle while not affecting the QoS.

The rest of this paper is organized as follows. In Section II, an introduction about TWT and its parameters are provided. In Section III, an overview of our proposed adaptive solution are presented. In Section IV, we discuss the category detection algorithms including the ML based network service type detection and State Machine (SM) based traffic type detection. In Section V, the details of category based TWT parameter design and customization are introduced. The validation on our proposed solution in commercial-ready system is presented in Section VI. Lastly, this paper is summarized in Section VII.

II. TARGET WAKE TIME INTRODUCTION

Besides conventional CSMA/CA-based random access, IEEE 802.11ax newly adopts scheduling-based access protocol, i.e., TWT, where STA and AP negotiate a wake interval and duration for data communication. At every negotiated interval, STA wakes up and communicates with AP for negotiated duration which is called Service Period (SP). After the end of SP, the STA is allowed to go to doze state. TWT brings benefits in two aspects: reduced power consumption at STA; reduced contention level of the network. There are two types of TWT: broadcast TWT and individual TWT [18]. In broadcast TWT, AP broadcasts predefined negotiations of wake interval and duration over beacon, and then STA retrieves memberships to desired negotiations. In individual TWT, STA requests individual wake interval and duration negotiation, as shown in Fig. 1. Eventually STA and AP would make an agreement on the TWT parameters called TWT agreement. In terms of meeting various and time-varying QoS requirements of smart mobile devices, each STA should be capable to negotiate its own desired wake interval and duration agreement, which favors the usage of individual TWT. In this context, we develop an adaptive solution based on individual TWT.

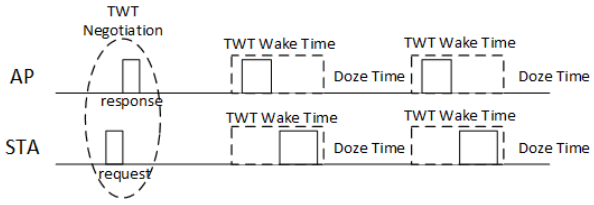


FIGURE 1. Individual TWT.

Element ID	Length	Control	Request Type	Target Wake Time	TWT Group Assignment	Nominal Minimal TWT Wake Duration	TWT Wake Interval Manitissa	TWT Channel	NDP Paging
Octets: 1	1	1	2	8 or 0	9 or 3 or 0	1	2	1	0 or 4

FIGURE 2. TWT element format.

For STA to negotiate individual TWT agreement, it transmits TWT setup request frame whose primary component is shown in Fig. 2. The detailed definition of each field in TWT element can be found in [18]. By configuring the two key TWT parameters shown below, we can control the duty cycle and the added latency of the 802.11ax link.

1) **TWT Wake Interval**: The wake-up time interval between two consecutive TWT sessions.

2) **Minimal TWT Wake Duration**: The minimal TWT time duration a STA shall stay awake since the starting time of the TWT SP. In this duration, STA is able to receive/transmit data frames from/to AP or other STA.

Here, wake interval is jointly determined by TWT Wake Interval Mantissa (M), and TWT Wake Interval Exponent (E_{w}, defined in "Request Type" field) by $M * 2^{E_{\mathrm{w}}}$, while wake duration is determined by Nominal Minimum TWT Wake Duration whose unit is adaptively determined between 256 us and 1024 us according to Wake Duration Unit (defined in "Control" field).

In the following sections, we use T_{inv} and T_{wd} to represent TWT wake interval and minimal TWT wake duration respectively. Note that although T_{wd} is the minimal TWT wake duration, the actual TWT wake time (T_{SP}) can be shorter than T_{wd}. This is because T_{wd} is the minimum in TWT mechanism, but Wi-Fi legacy Power Save (PS) mode [19] can work on top of TWT operation. In PS mode, the AP has the discretion to command the termination of TWT service period by indicating that there is no more buffered data or putting the end of service period (EOSP) in the MAC header. The different TWT behavior in PS mode and non-PS mode is shown in Fig. 3. On one hand, in PS mode, device can go to doze state before reaching T_{wd}, which is called early termination. On the other hand, in non-PS mode, device always waits until T_{wd} to finish a T_{SP}, which is called non-early termination. Therefore, in early termination case, T_{SP} could be smaller than T_{wd}. In practical system, early termination case could be common, as it saves more power.

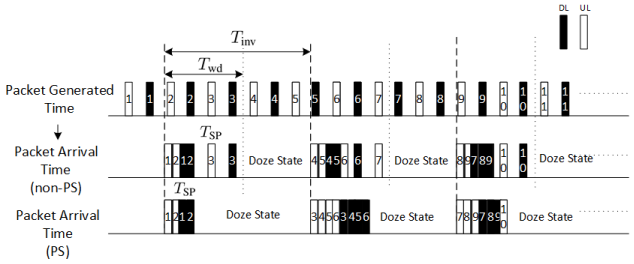


FIGURE 3. Comparison of early termination and non-early termination.

III. SOLUTION OVERVIEW

In this paper, we propose a fully adaptive solution for TWT parameter run-time configuration, in which the latency requirement is based on algorithm detection instead of external inputs. The TWT parameters mainly include two key parameters: T_{inv} and T_{wd}. First, let us look into the relation among T_{inv}, T_{wd} and the effective latency T_{L}. In ith TWT SP, T_{inv}, T_{wd} and T_{SP} are denoted as $T_{\mathrm{inv},i}$, $T_{\mathrm{wd},i}$ and $T_{\mathrm{SP},i}$, respectively. Assuming all the packets generated before the end of each SP can complete the transmission and reception within current $T_{\mathrm{SP},i}$, we can have the maximal latency for ith TWT Session as

$$T_{\mathrm{L},i} = T_{\mathrm{inv},i-1} - T_{\mathrm{SP},i-1} + T_{\mathrm{pkt},i}, \tag{1}$$

where $T_{\mathrm{pkt},i}$ is the time duration from the start of the ith TWT SP to the completion of tranceiving the first data packet. Usually $T_{\mathrm{pkt},i}$ is a small number. In reality, the TWT re-negotiation could require up to several rounds of Wi-Fi management frame exchange, which would introduce unignorable overhead. Therefore, for practical reasons, the TWT renegotiation would not happen for every TWT SP, and the same TWT parameters will be kept for a while (e.g., several seconds). In this case, (1) can be rewritten as

$$T_{\mathrm{L},i} = T_{\mathrm{inv}} - T_{\mathrm{SP},i-1} + T_{\mathrm{pkt},i}, \tag{2}$$

Considering that $T_{\mathrm{L},i}$ needs to be no greater than the latency requirement ($T_{\mathrm{L,Req}}$), we can see the requirement on T_{inv} is

$$T_{\mathrm{inv}} \leq T_{\mathrm{L,Req}} + T_{\mathrm{SP},i-1} - T_{\mathrm{pkt},i}, \tag{3}$$

As shown in Fig. 4, for the case without early termination, we have $T_{\mathrm{SP},i} = T_{\mathrm{wd},i}$. In the early termination case, we have $0 \leq T_{\mathrm{SP},i} \leq T_{\mathrm{wd},i}$. In order to develop a solution which can apply to both the cases with and without early termination, we need to satisfy the most strict requirement in (3) as

$$T_{\mathrm{inv}} \leq T_{\mathrm{L,Req}} - T_{\mathrm{pkt},i} \approx T_{\mathrm{L,\,Req}}. \tag{4}$$

Additionally, the duty cycle ($T_{\mathrm{SP},i}/T_{\mathrm{inv},i}$) needs to satisfy

$$\frac{T_{\mathrm{SP},i}}{T_{\mathrm{inv},i}} \geq \frac{P_{\mathrm{TF},i}}{P_{\mathrm{THP},i}}, \tag{5}$$

where $P_{\mathrm{TF},i}$ is the traffic in ith TWT SP, and $P_{\mathrm{THP},i}$ is the maximal throughput that the WiFi link could reach if running full duty cycle. Ideally, as long as $T_{\mathrm{inv},i}$ satisfies (3), same duty cycle ($T_{\mathrm{SP},i}/T_{\mathrm{inv},i}$) would provide same system throughput and power consumption. However, there are two factors that encourage the selection of larger T_{inv} while still satisfying (3). First, there is unignorable overhead power consumption for each sleep/wake-up switching. Longer interval would lead to less portion for overhead, consequently less power consumption. Second, in practical system, some AP and STA would have some minimum supported T_{wd} value. In this case for a given T_{inv}, there is a minimal effective duty cycle.

From (2) and (5), we can see that we can configure the TWT parameters ($T_{\mathrm{inv},i}$ and $T_{\mathrm{wd},i}$) with the latency requirement, the traffic information and link quality information.

To detect the latency requirement is a key challenging for this fully adaptive solution. To our best knowledge, all the existing solutions (such as [8], [11], [12]) assume the latency requirement/preference is known and given to the algorithm in prior. In this paper, we propose a ML based algorithm to detect the service type in real time which is then mapped to the required latency. In the proposed algorithm, we first classify the applications into different network service types according to the service (such as video call, audio call, streaming, etc), latency requirement and data characteristics. Then, we use ML method to detect the service type with the selected features. For each service type, we assign a predefined required latency, which is obtained from the test before hand. More details will be elaborated in Section IV.

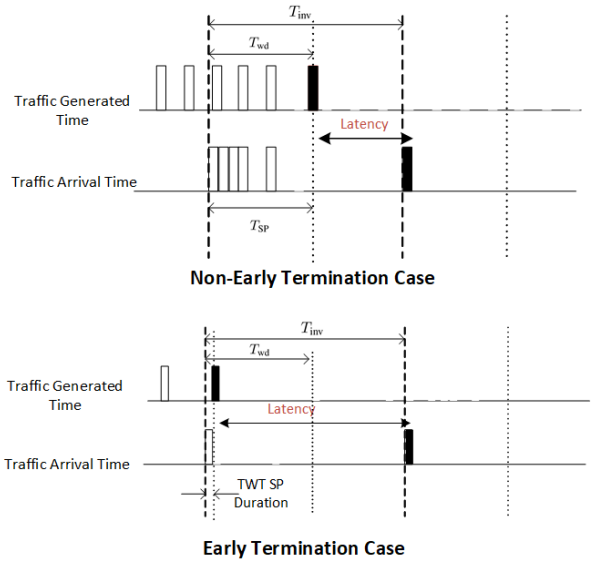


FIGURE 4. Latency illustration.

Additionally, for parameter T_{wd}, we need to assure that the duty cycle ($T_{\text{wd}}/T_{\text{inv}}$) is able to accommodate the traffic throughput. We propose a model to estimate the time (called as data time T_{dt}) which is needed for transmitting and receiving certain amount of transport layer data. Utilizing this model, we are able to map the traffic information into required data time. When using current T_{dt} to predict the future T_{dt}, some guard time is needed so that it could handle the traffic fluctuation. Different strategies of designing the guard time can be applied for different traffic types. According to the traffic characteristic, we can classify the traffic into 3 types: random, stable and bursty. For each traffic types, we use different strategies for estimating the required T_{wd}. For example, the video call and audio call application have relatively stable traffic, which can use a relative small guard time. It is also observed that some video streaming applications (e.g., YouTube, Netflix, etc) are with bursty traffic, as shown in Fig. 19. As the video streaming service is not sensitive to the latency, somehow flatting the traffic (assigning smaller T_{wd} comparing with peak T_{dt}) is allowed. The detailed algorithm for configuring T_{wd} will be described in Section V-B, and the traffic type detection algorithm will be discussed in Section IV-B.

The overall system architecture is shown in Fig. 5, which mainly includes two parts: ML & SM based category detection module, and category based TWT parameter generation module. In category detection module, there are two submodules. ML based service detection sub-module takes the traffic information (e.g., packet count, average packet size) and the packet timing information (e.g., interval-packet arrival time) in each time window to detect the current network service types. SM based traffic type detection sub-module takes the traffic information in each time window to detect the traffic type as one of random/stable/bursty. In TWT parameter generation module, the TWT parameters T_{inv} and T_{wd} are

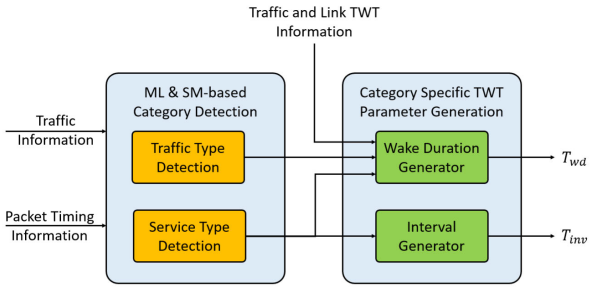


FIGURE 5. System architecture.

generated according to its category (service type and traffic type). TWT interval T_{inv} is determined by the service type. TWT wake duration T_{wd} is generated by jointly considering the service type, traffic type, the traffic information (e.g. throughput), and the link quality.

IV. CATEGORY DETECTION

This section describes in details the Network Service Detection (NSD) sub-module and the Traffic Type Detection sub-module.

A. NETWORK SERVICE TYPE DETECTION

With the NSD sub-module, we target on classifying different service types so that we can map the detected service to the latency requirement. The definition of the service type is essential to the performance of the sub-module. On one hand, applications in the same service need to have similar latency requirement so that the classification is meaningful. On the other hand, the applications assigned to the same service type need to contain strong common signature, so that the detection accuracy could be high enough. Considering both the latency requirement and signature, we design the sub-module as a hierarchical classifier. There are two layers in the NSD sub-module (see Fig. 6).

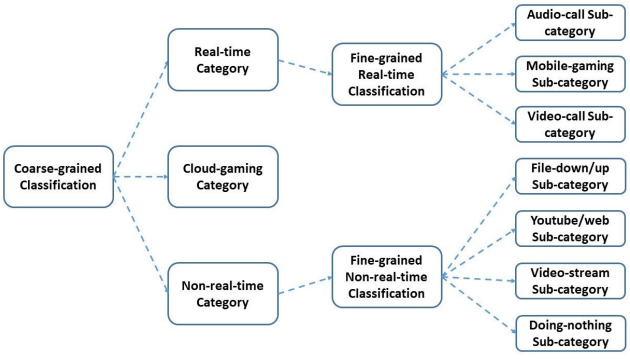


FIGURE 6. The pipelines of the 2-layers NSD sub-module.

The first layer (called layer 1) is a coarse grain classifier. It is in charge of classifying the service types into three main categories. At first, we will try to achieve a rough range of the latency requirement with strong signature in the

first classification layer. From then, the second layer (called layer 2) which includes two sub-classifiers will perform fine grain classifying on the results from layer 1. In layer 2, the goal is to achieve a finer range of the latency requirement if possible. We require the layer 1 classification to have very high accuracy, while we can tolerate a relatively lower accuracy of the layer 2 finer classification. With the 2-layer approach, even with some errors on layer 2 classifier, the impact of service detection error to the mapped latency requirement would be limited comparing to having one single classifier.

In layer 1, we classify the applications into 3 categories, which corresponds to requiring very low latency, low latency and high latency, respectively. For the very low latency applications, they are not even affordable to enable TWT function. We need to turn off TWT function for this category. From the latency analysis in Section V-A, we can see that only cloud gaming needs to have very low latency and the TWT needs to be disabled in this scenario. Therefore, we name the class requiring very low latency requirement as Cloud Gaming (CG). For defining low latency category and high latency category, we leverage the interaction characteristic of the application. If the applications require frequent user interaction or uplink/downlink interaction (e.g., online gaming, calling), they would require relative low latency. Otherwise, if there is no much interaction, then the latency requirement could be very loose (e.g., streaming, web-browsing, etc). From the signature perspective, interactive application maintains bi-directional traffic, while non-interactive applications normally have one of the directions dominating the traffic flow. Therefore, this property could help us to distinguish these two categories. We call the category with frequent interaction as Real-time (RT) service type, while the other category as Non-real-time (NRT) service type. Overall, there are three categories in layer 1: CG, RT, and NRT. For the RT category, services such as video call and audio call from applications such as WhatsApp, Zoom, Viber, and high interaction mobile games such as PUBG will fall in this category (see Tab. 1). The NRT category includes services that don't require real-time interaction. Examples of these are video-streaming (such as Netflix, Disney+), audio streaming (such as Pandora and Spotify), web-browsing, File-downloading (DL), File-uploading (UL) etc. Fig. 7 shows an instant for each of the layer 1 categories. These plots indicate distinct patterns from these categories which should yield high accuracy with the prediction of layer 1.

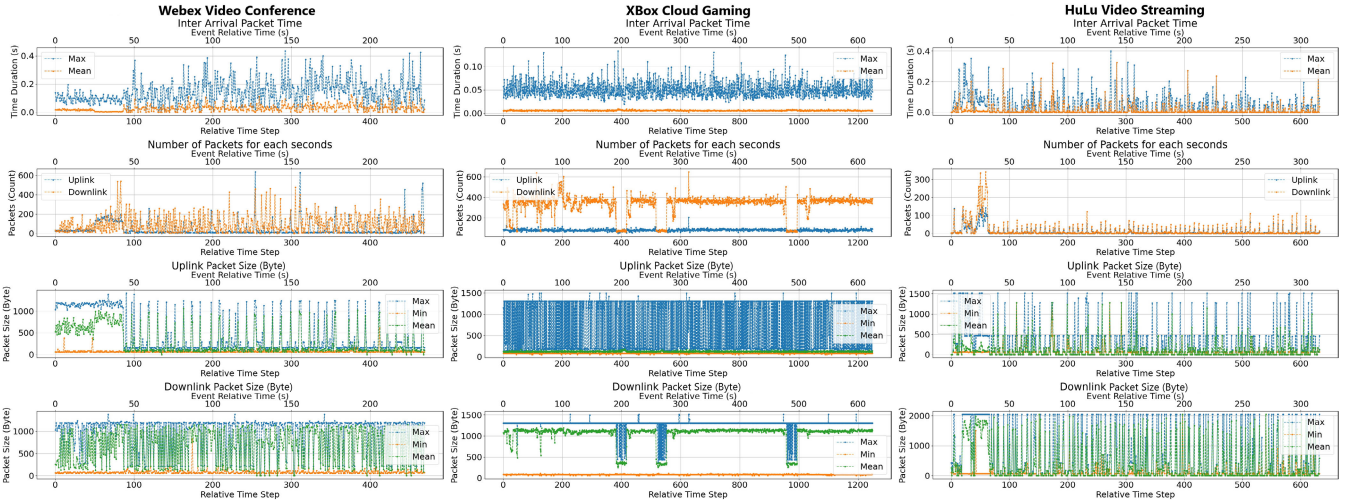
In layer 2, we only focus on classifying sub-categories of RT category and sub-categories of the NRT category. Therefore, there are two sub-classifiers. The first sub-classifier is to categorize the RT applications into sub-categories including audio-call service, mobile-gaming service, and video-call service. Fig. 8 shows samples of the selected features for these RT sub-categories. After classifying on these 3 classes, according to the latency requirement, we further combine the audio-call and mobile-gaming as relative High Latency (HL) RT application, while video-call is treated as relative Low Latency (LL) RT application. The detail latency analysis can be found in Section V-A. The second sub-classifier groups the NRT applications into four sub-categories including file-DL/UL service, youtube/web (YT/W) service, video-stream service, and no-APP-running. Similarly, after the NRT classification, we further combine youtube, web, streaming and no-APP-running as HL NRT type, while file-DL/UL as LL NRT type. Fig. 9 shows samples of the selected features for NRT sub-categories.

We used a set of 10 statistics features to help classify the categories and sub-categories which are computed over a duration of 500ms. We define this time duration as a time step. These features consist of:

- *Uplink maximum inter-arrival time*: the maximum time difference between arrival of one packet and the next packet within a time step (1 value).
- *Uplink average inter-arrival time*: the average time difference between arrival of one packet and the next packet within a time step (1 value).
- *Uplink & downlink packet counts*: The uplink and downlink number of packets within a time step (2 values).
- *Uplink & downlink minimum packet size*: The uplink and downlink minimum packet size in Mb within a time step (2 values).
- *Uplink & downlink maximum packet size*: The uplink and downlink maximum packet size in Mb within a time step (2 values).
- *Uplink & downlink average packet size*: The uplink and downlink average packet size in Mb within a time step (2 values).

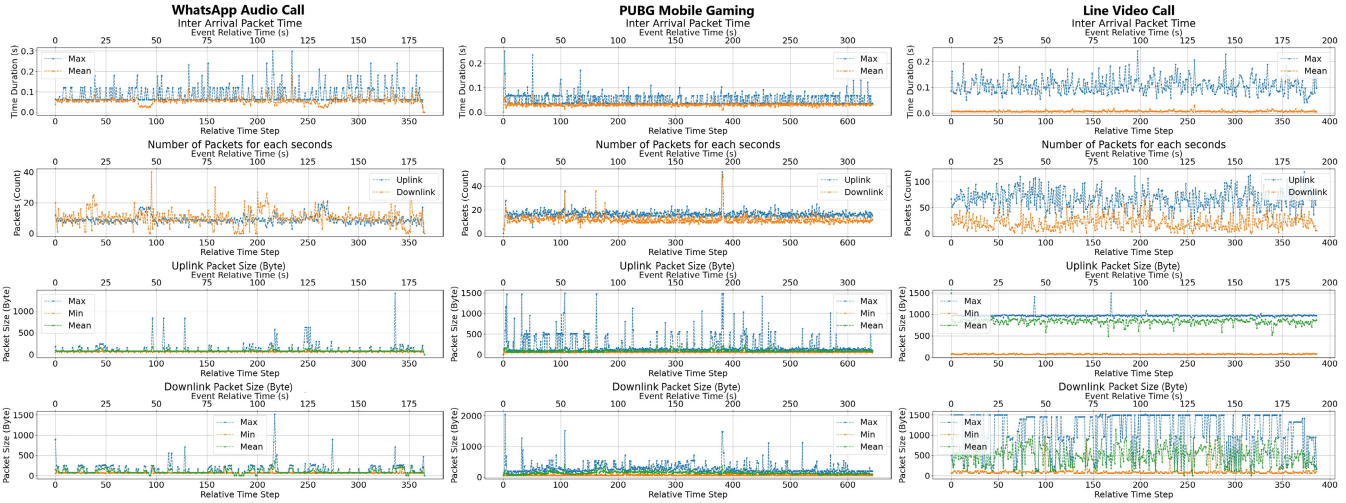
For layer 1, we use a sequence of 3 seconds, which means 6 time steps. For layer 2, we use a longer sequence with a time length of 6 seconds which is 12 time steps. Note that only uplink inter-arrival time is used, but not downlink. This is because the downlink packet timing has been disordered by TWT mechanism.

We experimented with different types of classifiers for layer 1. The first one is Random Forest (RF) from Scikit-learn [22]. This RF has a size of 30 trees and use the default hyper-parameters. The input to the RF is a vector that has a size of 60 since there is a total of 6 time steps and each time step provides 10 statistical features. The output is the index of the class which is 0 for real-time, 1 for cloud-gaming, and 2 for non-real-time. Fig. 11a shows the offline validation of the RF. The second one is RNN. The RNN that we used has the core of a Gated Recurrent Unit (GRU) [20] with a size of 64 hidden units. Fig. 10a depicts the architecture of the network. The input to the RNN is a vector that has a shape of [6, 10], for which 6 is the number of time-steps (3 seconds duration) and 10 is the number of features. The offline testing result of RNN first layer is shown in Fig. 11. The next classifier is the Conv1D. Fig. 10b shows the architecture. The idea is to use convolutional layer to convolute along the time dimension. The model extracts features from sequences and maps the internal features of the sequence. It's effective for deriving features and analysis of signal data



(a) Real-time category (b) Cloud-gaming category (c) Non-real-time category

FIGURE 7. **Plotted statistic features of some instances of layer 1 categories including real-time, cloud-gaming, and non-real-time.**

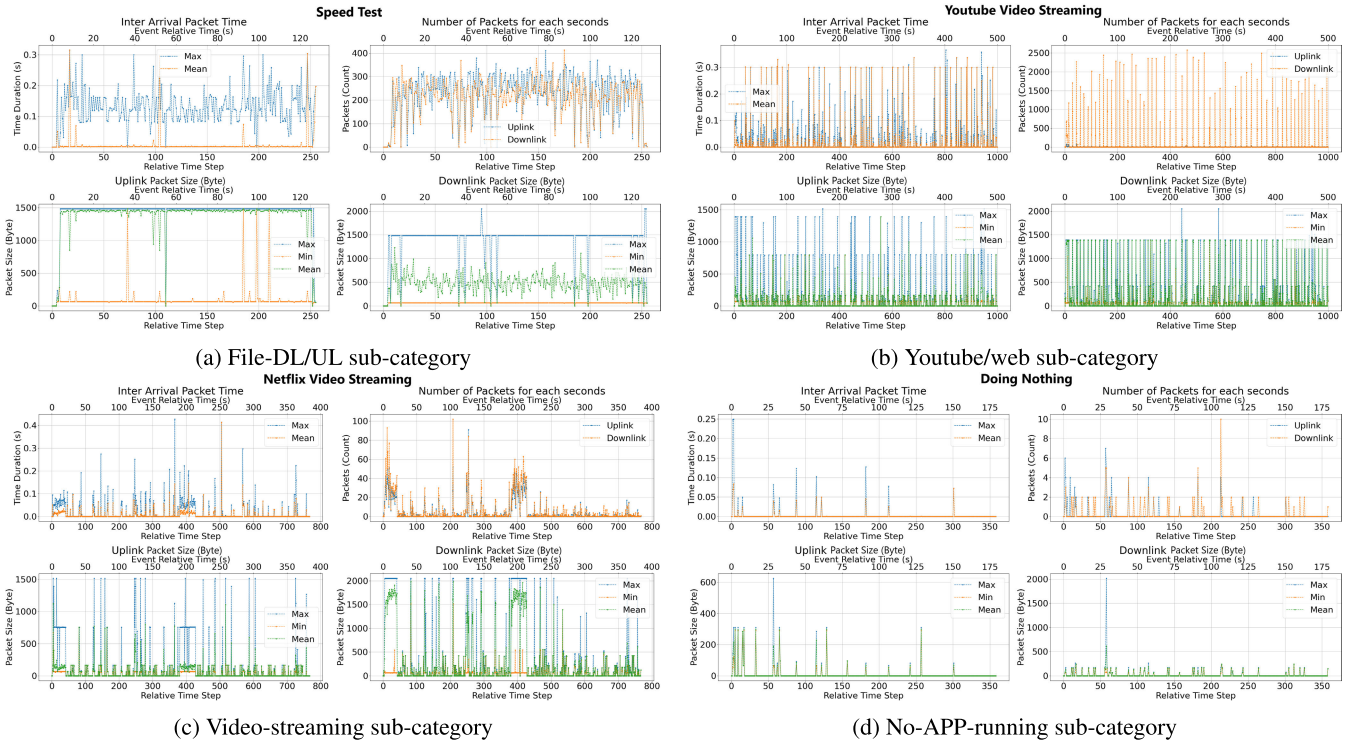


(a) Audio call sub-category (b) Mobile gaming sub-category (c) Video call sub-category

FIGURE 8. **Statistic features of some instances of layer 2 RT sub-categories including audio call, mobile gaming, audio call.**

over a fixed-length period. The last classifier, we utilize the XGBoost method [21], which is a type of gradient boosting method. The tree structure is used as the base-learner, and the same 10 network statistic features are used to train the XGBoost model. The learning rate, max depth of the tree and the number of boosting rounds are chosen to be 0.28, 6 and 36 respectively. Fig. 11d shows the offline testing results. Overall, the XGBoost has the best testing accuracy 97.26% in the first layer. All of the classifiers were trained on 155,702 samples and tested on 37,505 samples. We further examine the feature importance of the trained XGBoost model shown in Fig. 12. The 10 features along the x-axis of Fig. 12 are: maximum and average uplink inter-arrival time; uplink and downlink packet number; max, min and average uplink packet size; max, min and average downlink packet size. It is shown that the uplink maximum inter-arrival time and the downlink maximum packet size are the two most important features used by the model to differentiate each service types.

Subsequently, we used RNN one more time to implement the real-time sub-classifier. The input to the RNN is a vector that has a shape of [12, 10], for which 12 is the number of time-steps (6 seconds duration) and 10 is the number of features. It was trained on 52,223 samples and test on 13,091 samples. The offline testing result of RNN is shown in Fig. 11. For the non-real-time sub-classifier, we utilize the XGBoost again. For the offline training and testing, The XGBoost is trained on 42,100 samples and tested on 16,614 samples. The learning rate, max depth of the tree and the number of boosting rounds are chosen to be 0.3, 8 and 40 respectively. The offline testing result of XGBoost is shown in Fig. 13b.



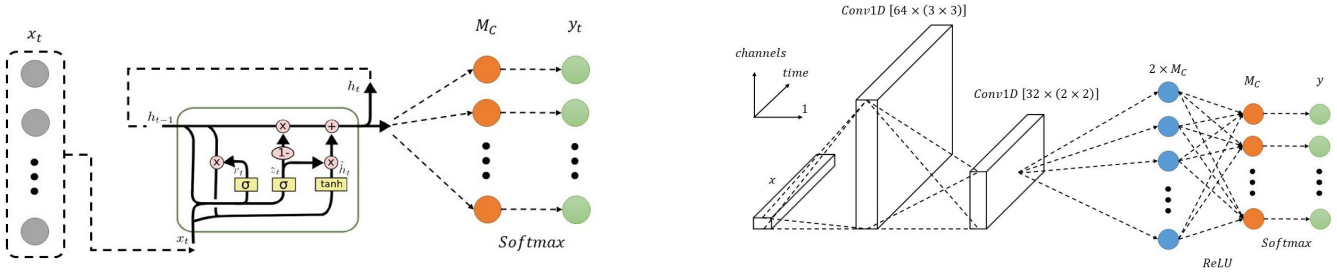
(a) File-DL/UL sub-category

(b) Youtube/web sub-category

(c) Video-streaming sub-category

(d) No-APP-running sub-category

FIGURE 9. Statistic features of some instances of the layer 2 NRT sub-categories including file-DL/UL, YT/W, video-streaming, and no-APP-running.



(a) Recurrent Neural Network.

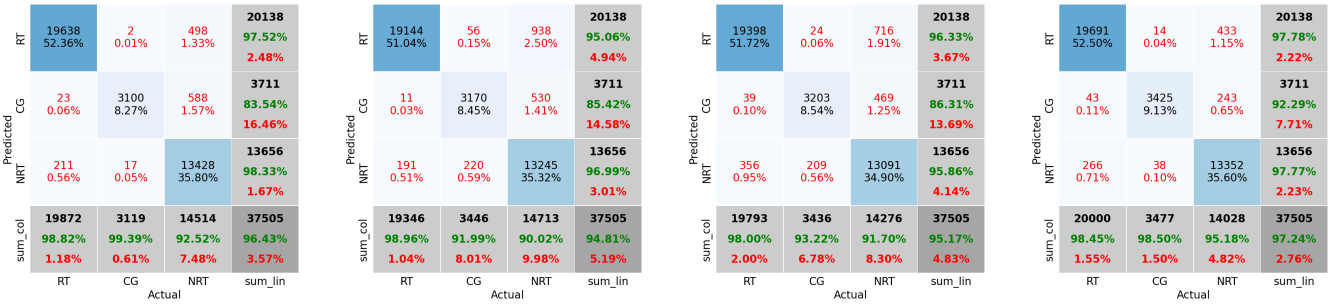
(b) 1D Convolutional Neural Network.

FIGURE 10. Our RNN and Conv1D architecture design. M_C is the number of real-time sub-categories which is 3 (audio-call service, mobile-gaming service, and video-call service). The Conv1D use 2 1D Convolutional layers. One has 64 feature maps and kernel size of 3. The other has 32 feature maps and kernel size of 2.

B. TRAFFIC TYPE DETECTION

For different applications, the variation of the throughput can show different patterns which we call traffic patterns. When considered jointly with T_{wd} customization strategy, we classify the traffic patterns into three categories: 1 - Random, 2 - Stable, 3 - Bursty, which are shown in Fig. 19. The bursty traffic features a periodic burst/large throughput as shown in Fig. 14. When the data in the local buffer is low or empty, a large throughput (shown as the burst in Fig. 14) can be observed to transmit the data to fill the local buffer. After the buffer is filled, relatively low and stable throughput (shown as the valley in Fig. 14) can be observed until the local buffer is close to empty for refilling again. The bursty traffic can often be seen in various streaming services such as Youtube live streaming and Youtube video. The stable traffic features a relative constant throughput with small variations, which can often be observed in some IM calling or teleconferencing applications such as WhatsApp audio call or zoom video conferencing. The random traffic often shows random variation of traffic throughput which is difficult to predict. The random traffic can be observed in the web browsing and other applications in which the user interaction with the phone is hard to predict.

For the three traffic types, we designed three different T_{wd} customization strategies which are described in Section V. The designed T_{wd} customization has different trade-off between TWT introduced latency and the power consumption. Among the three T_{wd} customization methods,



(a) Random Forest. (b) 1D Conv Neural Network. (c) Recurrent Neural Network. (d) XGBoost.

FIGURE 11. Offline result of the layer 1 classifiers. RT is real-time, CG is cloud-gaming, and NRT is non-real-time. The diagonal cells correspond to observations that are correctly classified. The off-diagonal cells correspond to incorrectly classified observations. Both the number of observations and the percentage of the total number of observations are shown in each cell. The column on the far right of the plot shows the precision (or positive predictive value) and false discovery rate, respectively. The row at the bottom of the plot shows the recall (or true positive rate) and false negative rate, respectively. The cell in the bottom right of the plot shows the overall accuracy.

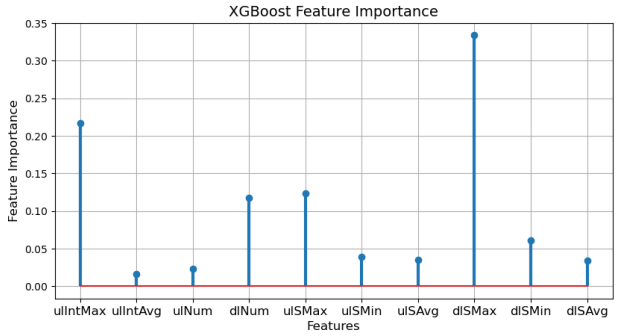
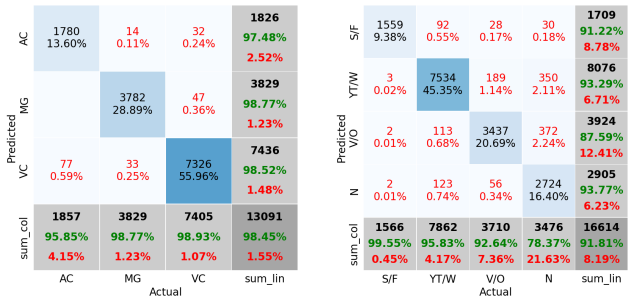


FIGURE 12. Layer 1 XGBoost model feature importance.



(a) Real-time. (b) Non-real-time.

FIGURE 13. Offline result of the layer 2 real-time RNN sub-classifier and XGBoost non-real-time sub-classifier. For real-time, AC is audio-call, MG is mobile-gaming, and VC is video-call. For non-real-time, S/F is speed test or file up/download, YT/W is Youtube or Web-browsing, V/O is video or other, and N is doing nothing.

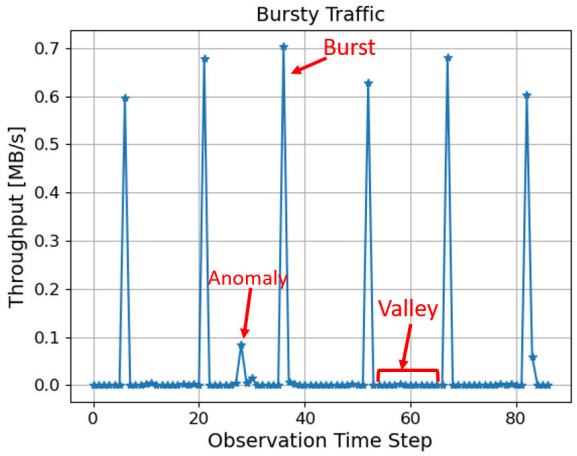


FIGURE 14. State-machine based traffic type detection method.

the one used for bursty traffic can have the largest power saving while it can also introduce the largest additional latency. As we prioritize the user experience, we would prefer that the false detection rate of the bursty is the lowest one, followed by stable traffic and then random traffic. To detect the three traffic types, we designed a State Machine (SM) based traffic type detection method as shown in Fig. 15. The SM is designed with the philosophy that the false detection rate of the bursty traffic should be the lowest one. As shown in Fig. 15, there are 5 internal state of the SM, and 9 conditions for the transitions among each state. For State 0 (initial and default state) and State 1, the output of the classification result will be random traffic. For State 2, the output will be stable traffic class. For State 3 and State 4, the output will depend on the value of $C_{V,B}$, which counts the number of valleys and bursts that have been observed. If the $C_{V,B}$ is larger than N, then we classify the current traffic as bursty traffic. Otherwise, the output of State 3 and State 4 will be random traffic. When the value of N is increased, we make the classification criteria of bursty traffic more strict and the false detection rate of bursty traffic can be further reduced. On the meantime, the detection time required to confirm a bursty traffic can also be increased. A typical value range of N is 0 to 3, and we use the value 0 in our system for a faster detection of bursty traffic.

To give a more clear explanation of the 9 conditions for state transition, we further divide them into three groups based on the traffic type detection results they may lead

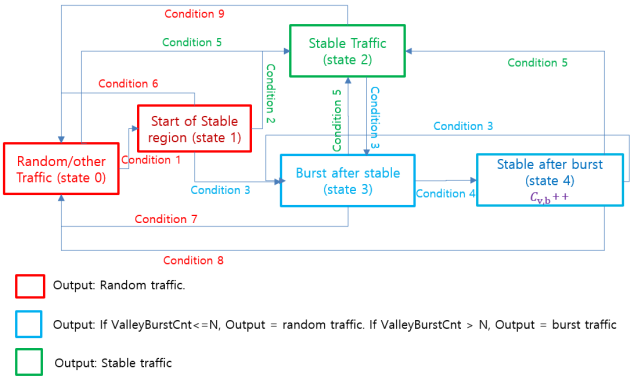


FIGURE 15. State-machine based traffic type detection method.

to. The three groups of state transition conditions are listed below.

Bursty Traffic Conditions: 1, 3, 4.

Stable Traffic Conditions: 1, 2, 5.

Random Traffic Conditions: 6, 7, 8, 9.

The **Bursty Traffic Conditions** are criteria to transit from random or stable traffic to bursty traffic, which will be discussed below.

Condition 1:

$$|\Delta P_i| <= Th_1 \quad (\text{for } i = N - k_1 + 1, N - k_1 + 2, \ldots, N)$$
$$\textbf{AND } \left| P_N - \text{MA}_i^{k_1}(P_i) \right| <= Th_2,$$

where $\text{MA}_i^{k_1}(P_i) = \frac{1}{k_1} \sum_{i=N-k_1+1}^{N} P_i$ is the moving average of P_i with a window size of k_1, and P_N is the throughput of the most recent observation. The values we use for k_1, Th_1 and Th_2 are 5, 0.1 Mbyte/s and 0.12 Mbyte/s, which are acquired empirically. The P_i is the throughput at the ith observation, and $\Delta P_i = P_i - P_{i-1}$ is the change of throughput between each observation. The first equation in condition 1 can ensure that variation of the throughput is small between each observation and the second equation ensures that the throughput remains stable and flat instead of keep increasing or decreasing. If condition 1 is satisfied, then the SM transit from state 0 to state 1. The condition 1 is used to find the initial stable part of the traffic characterized by state 1. If the traffic throughput continues to remain stable, then the SM would transit to state 2 for stable traffic classification. Otherwise, if a burst is observed, the SM will transit to state 3 and may lead to the classification as a bursty traffic.

Condition 3:

$$\Delta P_i > Th_\text{b} \textbf{ OR } \sum_{i=N-j}^{N} \Delta P_i > w \cdot Th_\text{b}$$

Condition 3 finds the time that a sudden increase of the throughput ΔP_i is larger than Th_b. Meanwhile, for bursty traffic, sometimes the throughput might not see a sudden steep jump between two observation time. Thus the condition that the sum of the throughput increase within j observation times being larger than $w \cdot Th_\text{b}$ can also be used. We set $j = 2$ and $w = 1.6$.

After entering state 3 (burst after stable state), we look for the next valley of the burst traffic by testing when the condition 6 will be satisfied. Condition 4 consists of condition 1 which ensures a stable traffic pattern and an additional condition that the moving average of the current valley's throughput must be similar to the moving average of the throughput of the previous valleys of the bursty traffic. This is the based on the fact that the throughput of the valley part of one bursty traffic should remain relatively similar.

Condition 4:

$$\text{Condition } 1 \textbf{ AND } \left| \text{MA}_i^{k_1}(P) - \text{MA}_j^{k_v}(P_{\text{v},j}) \right| <= Th_v$$

where $P_{\text{v},j}$ is the mean throughput of the jth previous valley within the current bursty traffic, and k_v is the number of previous valleys used to compute the moving average of the valleys' throughput. The value of k_v and Th_v are 5 and 0.12 Mbyte/s. This condition ensures that the mean throughput of the current valley is close to the moving average of the mean throughput of the previous valleys. After entering state 4 with condition 4, we have already identified a valley-burst-valley pattern of the traffic which indicates that the current traffic is highly likely to satisfy the bursty traffic pattern. We use a counter $C_{\text{V,B}}$ to record the number of times that State 4 is continuously reached from State 3, which represents the number of times that the valley-burst-valley traffic pattern is observed. We use the criteria that if $C_{\text{V,B}} > N$, then the traffic can be classified as bursty traffic. Currently, we set the number $N = 0$, while N can also be set to a larger number to further reduce the false detection rate of bursty traffic type.

The **Stable Traffic Conditions** are criteria to transit from random or bursty traffic to stable traffic, which will be discussed below.

Condition 2:

$$|\Delta P_i| <= Th_3 \text{ (for } i = N - k_2 + 1, N - k_2 + 2, \ldots N)$$
$$\textbf{AND } \left| P_N - \text{MA}_i^{k_2}(P_i) \right| <= Th_4,$$

Condition 2 is similar to condition 1 with the difference on the throughput variation threshold being set to Th_3 and Th_4, which are usually larger than Th_1 and Th_2. This is due to the fact that the throughput variation of the stable traffic, such as video conferencing, is often larger than the throughput variation of the valley part of bursty traffic. The value of Th_3 and Th_4 are set to be 0.2 Mbyte/s and 0.25 Mbyte/s. The variable k_2 represents the number of additional observations needed to confirm a stable traffic after observing the initial stable part of the traffic with state 1. The value of k_2 is set to 9.

Condition 5 shares the same form as condition 2 with the only difference in the number of consecutive observations (k_3 instead of k_2) needed to directly confirm a stable traffic without detecting the initial stable part of the traffic. A typical value of k_3 is 14 which can be the sum of k_1 and k_2.

The **Random Traffic Conditions** are criteria to transit from bursty or stable traffic to random traffic, which includes **Condition 7**, **8** and **9** as discussed below.

Condition 6: In state 1, when condition 1 is **NOT** satisfied, then transit to state 0. Condition 6 is used to check whether

the traffic pattern sees a large variation and is no longer stable, which indicates that the traffic becomes random.

Condition 7:

$$T_{\text{bur}} >= \text{MA}_i^{k_4}(T_{\text{val},i})$$

where T_{bur} is the time duration of the current burst in state 3, and $\text{MA}_i^{k_4}(T_{\text{val}}, i)$ is the moving average of the time duration of the valley part of the current bursty traffic. This condition is based on the fact that the time duration of the valley is always statistically longer than the duration of the burst.

Condition 8: In state 4, when condition 1 is **NOT** satisfied for N_1 consecutive observations or N_2 observations within a N_3 long observation window, then we will transit to state 0. A typical value of N_1, N_2 and N_3 is 2, 2 and 10.

Condition 8 is very similar condition 6 but with added tolerance to allow some anomalies (shown in Fig. 14) in the valley part of burst traffic. The tolerance is added based on the conditional probability that, when state 4 is reached, the traffic is highly possible to be bursty traffic and some abnormal pattern at the valley part is then tolerable.

Condition 9 is similar to the criteria in condition 8. The only difference is that ''when condition 1 is **NOT** satisfied'' is changed to ''when condition 2 is **NOT** satisfied''.

V. CATEGORY BASED TWT PARAMETER CUSTOMIZATION

A. LATENCY REQUIREMENT ANALYSIS

In this sub-section, we describe in detail how we experimentally analyze the latency requirement for various applications. As discussed previously, the maximal latency for a given TWT SP is a function of the TWT wake interval and the minimal TWT wake duration for the previous TWT SP (in order to control T_{SP}, we turned off early termination). Since the user experience is influenced by the maximal latency on the last hop, it is important to understand the last hop latency requirement for a given traffic stream.

Prior work largely addresses the round trip latency requirement for various applications. Since the round trip latencies are also influenced by the backbone delay values, they only provide a loose upper bound on the latency requirement on the last hop. However, a tighter bound is essential for an efficient interval design. To this end, we experimentally evaluate the last hop latency for various applications. Based on the last hop latency requirement and similarity in traffic stream, we then group applications together into various categories.

1) EXPERIMENTAL EVALUATION OF LATENCY REQUIREMENT

For the purpose of our experimental evaluation, we consider a number of popular applications (e.g., Facebook, WhatsApp, YouTube, etc.). Some of the applications support features that result in more than one type of data flow. For such applications, the last hop latency requirement is evaluated for each of the data flow type that it can generate. To achieve this, we following the following steps.

In the first step, we consider various functions supported in popular applications. For each function (e.g., audio call, video call, etc), we evaluated the latency requirement. For a given function or application, we keep a fixed duty cycle and then consider a number of T_{inv} values. Note that since the duty cycle is fixed, the T_{SP} varies with T_{inv}. For each pair of (T_{inv}, T_{SP}), we evaluate the performance of each function or application. As described in Eq. (1), T_{L} is approximately equal to the difference between T_{inv} and T_{SP}, when early termination is turned off. Therefore, for a given pair, the traffic gets subjected to a fixed last hop latency value as computed based on Eq. (1). Therefore, as we value the T_{inv} and T_{SP}, we evaluate the performance of the given function or application at a number of last hop latency values. The maximum last hop latency value at which the application does not show any visible degradation in performance gives us the maximum last hop latency the application can tolerate for the given function or application. This value serves as an upper bound for the TWT wake interval in our design. Note that the real impact factor of the user experience is the overall latency instead of last hop latency. The requirement on the last hop latency could change with the overall network condition. Considering T_{SP} normally cannot be close to zero for the case requiring low latency, thus when we set $T_{\text{inv}} = T_{\text{L,Req}}$, we already gain some margin by T_{SP}. This could protect the latency variation from backhaul network.

2) CATEGORY GROUPING BASED ON REQUIREMENT SIMILARITY

We group applications based on a number of criteria, aligning with the service type definition in Section IV-A. First, we group application based on function that they support, e.g., video call functions of different applications are grouped together. And if one application support multiple functions, those functions are grouped separately. For example, WhatsApp supports both audio call and video call. These two functions belong to two different groups. The list of classes are aligned with the layer 2 fine classification in Section IV-A. For each of these groups, the latency requirement is the minimum of the latency requirement for the applications that fall in that group.

On top the layer 2 service types, we further group them into layer 1 service type according to the criteria in Section IV-A. Overall, we have layer 1 service types as CG, RT and NRT. And we have layer 2 service types as audio-call, video-call and mobile gaming for RT, while streaming, web-browsing and file DL/UL as shown in Tab. 1. Note that YouTube is in the web-browsing type. This is because YouTube has both video streaming and video/comment browsing. And we found that the video/comment browsing requires lower latency than streaming. Thus, we count it in web-browsing type. Additionally, in our testing system, there is a constraint on the maixmal T_{inv}. There for, we cannot test with the latency more than 80ms. If the latency requirement is larger than 80ms, in Tab. 1, we marked as ''$>$ 80ms''.

TABLE 1. Application categories with latency requirment for each category.

Layer 1 Category	Layer 2 Category	Latency Requirement for Layer 2 Category	Applications	Latency Requirement for each application
Cloud Gaming	-	TWT OFF	-	-
Real Time Services	Gaming (HL)	40 ms	Brawl Stars	40ms
			Call of Duty	40ms
			PUBG	50ms
			Ludo King	50ms
			Clash of Clans	50ms
	Audio Call (HL)	40 ms	Facebook Messenger	50 ms
			Line	50 ms
			Zoom	50 ms
			WebEx	50 ms
			WhatsApp	50 ms
			Discord	50 ms
	Video Call (LL)	24 ms	Facebook Messenger	40 ms
			Line	50 ms
			Zoom	50 ms
			WebEx	25 ms
			WhatsApp	25 ms
			Discord	25 ms
Non-real time services	Streaming (HL)	48 ms	ESPN+	>80 ms
			Netflix	>80 ms
			Hulu	>80 ms
			Disney+	>80 ms
			Pandora	>80 ms
			Spotify	>80 ms
	Web browsing (HL)	48 ms	YouTube	50 ms
			CNN	64 ms
			Abc News	>80 ms
			New York Times	>80 ms
			Amazon	>80 ms
			Walmart	>80 ms
	File DL/UL (LL)	40 ms	Google Drive	>80 ms
			Drop Box	>80 ms
			Box	>80 ms
			Google Speed Test	40 ms

Based on the test result shown in Tab. 1, we can see that in RT category, audio-call and mobile gaming have very similar latency requirement. Considering some mobile gaming actually supporting voice chat in the game, sometimes there also could be some error on differentiating audio-call and mobile gaming. Therefore, we combine the audio-call and mobile gaming in the final service detection output as RT HL, and assign the same latency requirement ($T_{\mathrm{L,Req}}$) as 40ms. For video-call, it is treated as RT LL, assigning $T_{\mathrm{L,Req}} = 24$ms.

For CG, the latency requirement is very strict. Based on our test, even when we turn off TWT, sometimes it still shows some video quality problem. Thus, it is preferred to turn off TWT for CG.

In the NTR category, file DL/UL includes the WiFi speed test application as well. This is because these two applications have similar data pattern. Generally this type of applications do not require low latency. However, in some case, such as google speed test, if the latency is too high, the speed test result could be impacted. Due to this, the file DL/UL is treat as NRT LL type, assigning $T_{\mathrm{L,Req}} = 40$ms. For rest of NRT application, we treat it as NRT HL type and assign $T_{\mathrm{L,Req}} = 48$ms.

The summary of this grouping and assign $T_{\mathrm{L,\,Req}}$ are illustrated in Tab. 1. According to Eq. (4), $T_{\mathrm{L,Req}}$ will be the T_{inv} that we use in TWT parameter configuration for the corresponding service type.

B. MINIMAL WAKE DURATION CUSTOMIZATION

Once the latency requirement for a category determined a T_{inv}, we can predict the required T_{wd} based on the traffic flow. To predict the T_{wd}, we need to look at the traffic flow over time and make the prediction based on the estimation on required T_{wd} in current TWT SP and the traffic type of the data we are serving. The update to T_{wd} is done in two ways in this algorithm: overflow update and stable update. An overflow update is an event where the current T_{wd} is insufficient for the incoming traffic and we need to ramp up the value as fast as possible. A stable update is an event where the T_{wd} is updated based on the long term statistics of the traffic flow. Generally the stable update happens more frequent as it follows the traffic fluctuation. And overflow update happens less frequent, as it only happens when the traffic has a sudden large change.

As mentioned, TWT negotiation has unignorable overhead. Therefore, the frequency of re-negotiation cannot be too frequent. Due to this, we design to check if it needs to do stable update every several seconds (e.g., 3s), called as algorithm poll. And as the overflow update requires prompt response, it has to check if overflow update is needed with short cycle. The polling for the need of overflow update is called overflow poll. The overflow polling happens every T_{inv}. Considering the overflow update happens rarely, more frequent polling could be acceptable. Additionally, although

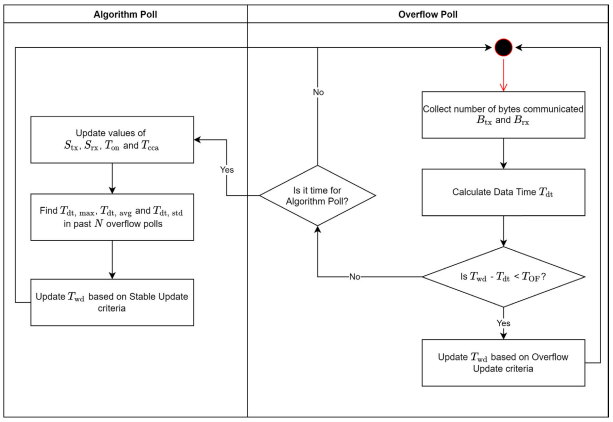


FIGURE 16. Polling cycles and updates.

the algorithm poll happens every several seconds, the data statistics need to be polled more frequently. In our design, the data statistics is polled together with overflow poll. The polling cycle and interaction of different modules is shown in Fig. 16. The data acquired includes:

1) The number of bytes transmitted/received (transport layer) for each T_{inv} (B_{tx}/B_{rx}), which is acquired at every overflow poll,
2) The transmit/receive linkspeed S_{tx}/S_{rx}, which is acquired at every algorithm poll,
3) The amount of time the radio was awake on the channel T_{on}, which is acquired at every algorithm poll, and
4) The amount of time the Clear Channel Assessment (CCA) was held busy on the channel T_{cca}, which is acquired at every algorithm poll.

Based on the above mentioned data, we map the amount of TX/RX data into the required time for data TX/RX, called as data time T_{dt}, which we use to predict the T_{wd} for the incoming traffic. T_{dt} is calculated at every overflow poll and it is evaluated if an overflow update is required or not. If at the end of an algorithm poll, an overflow has not occured, then we do a stable update. The details of data time estimation, overflow update and stable update is discussed as follow.

1) DATA TIME ESTIMATION

T_{wd} that is needed to finish transceiving data between the AP and STA during a T_{inv} is related to the amount of data, TX/RX data rate, and the congestion level. The traffic statistics that we can obtain at the STA side are the transport layer total TX data size (B_{tx}) and RX data size (B_{rx}), the PHY layer TX/RX data rate (R_{tx} and R_{rx}), the CCA time (T_{cca}) and the radio-on time (T_{on}). In 802.11ax, there is a throughput estimation parameters, which estimates the MAC layer throughput. But it cannot estimate the throughput for less than 1 beacon interval, which is not very suitable for our solution. Thus, a new model needs to be developed which can utilize the information above to estimate the required T_{wd} given a TWT interval T_{inv}.

First, we develop a preliminary data time estimation model in the scenario that there is no congestion. In this case, only B_{tx}, B_{rx}, R_{tx} and R_{rx} are needed for the modeling. The proposed model to calculate the total time T_{data} needed to transmit and receive $B_{tx} + B_{rx}$ bytes of transport layer packets can be formulated as:

$$T_{data} = \frac{B_{tx}}{S_{tx}} + \frac{B_{rx}}{S_{rx}} \tag{6}$$

where the S_{tx} and the S_{rx} are the effective transport layer transmitting and receiving data rate. The S_{tx} and the S_{rx} can be computed with the PHY data rate R_{tx} and R_{rx} and mapping factors as below:

$$\begin{aligned} S_{tx} &= \alpha_{tx,n} * R_{tx} \\ S_{rx} &= \alpha_{rx,n} * R_{rx} \end{aligned} \tag{7}$$

where $\alpha_{tx,n}$ and $\alpha_{rx,n}$ are the transmitting and receiving mapping factors at the nth PHY data rate region acquired empirically. Here we divide the PHY rate into four speed regions, and a segmented linear mapping is adopted to map the PHY rate to transport layer data rate. This is due to the reason that the MAC payloads(transport layer packets) are transmitted with the link speed while the PHY/MAC overheads take almost same time regardless of the different link speeds. The values of $\alpha_{tx,n}$ and $\alpha_{rx,n}$ for different speed regions are shown below.

$$(\alpha_{tx}, \alpha_{rx}) = \begin{cases} (0.37, 0.42), & (R_{tx}, R_{rx}) > 800 \\ (0.50, 0.56), & 300 < (R_{tx}, R_{rx}) <= 800 \\ (0.63, 0.63), & 70 < (R_{tx}, R_{rx}) <= 300 \\ (0.65, 0.66), & (R_{tx}, R_{rx}) <= 70 \end{cases} \tag{8}$$

The unit for R_{tx} and R_{rx} is Mbps. Theoretically, different packet size could lead to very different $\alpha_{tx,n}$ and $\alpha_{rx,n}$. During our test, we found that for most of the applications the dominated packet size is 1,000-1,500 bytes. So practically, we do not need to introduce packet size as another variable in the model.

The model provides us the time required to transmit or receive packets but doesn't take into account the additional time caused by channel contention. To compensate this part, we add a factor which uses T_{on} and T_{cca}. The T_{cca}/T_{on} can reflect the congestion level. Using T_{cca}/T_{on}, we derive an inverse CCA-over-Radio-On model which is represented as:

$$\begin{aligned} T_{dt} &= T_{data} * A_{amp} \\ &= T_{data} * \left(\frac{T_{on}}{T_{on} - T_{cca}} * \alpha + 1 - \alpha \right) \\ &= T_{data} * \left(\frac{\alpha}{1 - \frac{T_{cca}}{T_{on}}} + 1 - \alpha \right) \end{aligned} \tag{9}$$

where A_{amp} is the data time amplification factor due to congestion, and α is the tunable parameter to determine the scale of amplification which can be acquired empirically. The $1-\alpha$ term ensures that the A_{amp} will equal to 1 when there is no congestion. To empirically acquire the value of α, we measure the minimum required total data time T_{dt} at different congestion levels (different T_{cca}/T_{on} value), and then find the A_{amp}

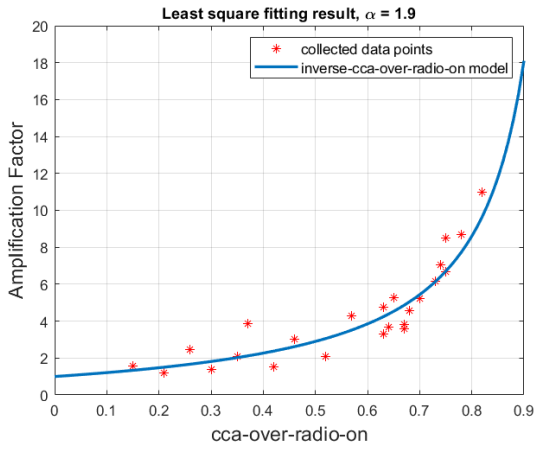


FIGURE 17. Inverse-CCA-over-Radio-On model.

by dividing the T_{dt} over T_{data}. Then a least square fitting of the data on the CCA-over-Radio-On model is performed to find the value of α which is shown in Fig. 17. As shown in our result, the selected α value is 1.9.

2) OVERFLOW UPDATE

An overflow update refers to updating the T_{wd} when it is estimated to be insufficient for the incoming traffic. The insufficiency criteria is based upon the following formula:

$$T_{\text{wd}} - T_{\text{dt}} < T_{\text{OF}} \tag{10}$$

where T_{OF} is the overflow threshold. The above condition is checked for each overflow poll. Overflow protection provided by the overflow update is key to accommodate random bursts of data that can arise due to any conditions that cannot be accounted for by stable update. The overflow threshold, T_{OF}, is updated based on the following criteria:

$$T_{\text{OF}} = \max(\rho_{\text{OF}} * T_{\text{wd}}, 1500\mu\text{s}) \tag{11}$$

where ρ_{OF} is the overflow threshold percentage. The ρ_{OF} is adjusted based on the traffic type as shown in Table 2.

On an overflow trigger, we update the T_{wd} for random and stable traffic types by the following formula:

$$\begin{aligned} T_{\text{wd}} &= T_{\text{dt,max}} + \delta_{\text{OF}} \\ \delta_{\text{OF}} &= 0.2 * T_{\text{inv}} \end{aligned}$$

where δ_{OF} is the overflow guard time, and $T_{\text{dt,max}}$ is the maximal T_{dt} from last algorithm poll. As we can see in the formula, when a burst change of traffic arrives at the STA, it corresponds to a large T_{dt}. In this case, the T_{dt} in the current overflow poll corresponds to the $T_{\text{dt, max}}$. We then scale the T_{wd} by adding 20% of the T_{inv}, which is the overflow guard to the $T_{\text{dt,max}}$. The overflow guard, δ_{OF} is added in order to accommodate for additional traffic that may come in the next TWT SP or provide a safe guard time for the stable traffic that may have ramped up in the previous TWT SP.

A special case of overflow update is the bursty traffic type with NRT service type. In this case the bursts are short intervals of traffic which arrive pseudo-periodic with long intervals of silence interleaved between the bursts. We customize the overflow update for the bursty traffic type by reducing the amount we update the T_{wd} by. On encountering an overflow trigger during the burst traffic type, the update formula is as follows:

$$T_{\text{wd,i+1}} = T_{\text{wd,i}} + \delta_{\text{OF}} \tag{12}$$

where i is the index of the ith TWT SP. As we can see here, instead of updating on the $T_{\text{dt,max}}$ we update on the previous T_{wd}. This reduces the amount by which the T_{wd} scales. As bursty traffic is usually observed in NRT type which is without strict latency requirement. This customization saves power by scaling smaller than the regular overflow update.

The ρ_{OF} is kept as 0.2 in the case of Random traffic because the traffic type is non-deterministic and we may need to adapt more quickly. An additional case which requires quick ramping up is speedtest and file downloading in which case we need to increase T_{wd} as much as possible because the throughput requirement is higher for these applications. In the case of deterministic traffic like stable or bursty, ρ_{OF} is kept as 0.1. This reduces the probability of having unnecessary overflows and wasting of power.

3) STABLE UPDATE

A stable update refers to updating the T_{wd} subject to long-term statistics. The long-term statistics are calculated by T_{dt} from the past N polls, where N is determined by the algorithm polling cycle subject to the detected traffic type. For example, in random type, the algorithm polling cycle is 3000ms (shown in Tab. 2), $N = 3000\text{ms}/T_{\text{inv}}$. For long-term statistics, we look at $T_{\text{dt, max}}$ and standard deviation of T_{dt} ($T_{\text{dt, std}}$) in the previous N overflow polls. The stable update the T_{wd} is done by:

$$T_{\text{wd}} = T_{\text{dt,max}} + \delta_{\text{stable}} \tag{13}$$

where δ_{stable} is the stable guard. The stable guard is added to accommodate for random or non-deterministic traffic that may arrive in addition to the stable traffic. δ_{stable} is formulated as:

$$\delta_{\text{stable}} = \begin{cases} \max(T_{\text{dt,std}}, \epsilon), & \text{if Traffic Type is stable} \\ \max(0.1 * T_{\text{inv}}, \epsilon), & \text{otherwise} \end{cases} \tag{14}$$

where ϵ is the time (calculated by Eq. 9) that is required to receive a Mac Protocol Data Unit (MPDU) or Mac frame of max size 11,454 bytes. This is estimated in the same way we estimate the traffic using the data time estimation discussed above. As we note here, for stable traffic, δ_{stable} we use the $T_{\text{dt, std}}$ instead of 10% of the T_{inv}. This is because in stable traffic we just accommodate for the variation in traffic which is deterministic. Additionally, for the bursty traffic, the stable update is performed by taking the sum of the $T_{\text{dt, avg}}$ and the δ_{stable}. The intuition behind this is that the bursty traffic is not very sensitive to latency and the communication can be spread over time without impact to the user experience.

TABLE 2. Traffic type based wake duration update.

Parameters	Traffic Type		
	Random	Stable	Burst
Overflow Threshold Percentage, ρ_{OF}	0.2	0.1	0.1
Overflow Threshold, T_{OF}	$\max(\rho_{OF} * T_{wd}, 1500\mu s)$		
Overflow Guard, δ_{OF}	$0.2 * T_{inv}$		
Overflow Update	$T_{dt,max} + \delta_{OF}$	$T_{dt,max} + \delta_{OF}$	$T_{wd} + \delta_{OF}$
Stable Guard, δ_{stable}	$\max(0.1 * T_{inv}, \epsilon)$	$\max(T_{dt,std}, \epsilon)$	$\max(0.1 * T_{inv}, \epsilon)$
Stable Update	$T_{dt,max} + \delta_{stable}$	$T_{dt,max} + \delta_{stable}$	$T_{dt,avg} + \delta_{stable}$
Algorithm Poll Duration	3000 ms	6000 ms	2000 ms

VI. RESULT

A. TEST SETUP

The test platform is as shown in Fig. 18. In the test system, we use ASUS ROG GT-AX11000 Tri-Band Wi-Fi router, which supports TWT function, as our testing router. We implement the proposed algorithm in a Samsung Galaxy S21 (SM-G998U), which is as testing phone with TWT enabled. There are additional 4 phones, which are associated to the same AP. As shown in Fig. 18, three phones are used to emulate the congested network environment. One of these 3 phone runs the iperf server, while the other 2 phones are running as iperf client. When we test the congested network condition, these 3 phones will run the iperf speed test to create congested network condition. And there is one additional non-TWT phone that runs same application as the testing phone, which is to perform as reference of the user experience for the test phone. In the test, all the devices use the same 5G band.

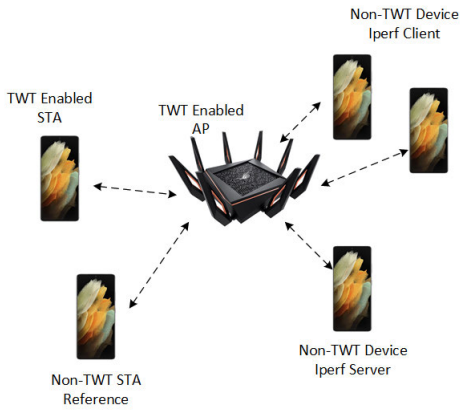


FIGURE 18. Test setup.

As the test setup are using commercial devices, there are some constraints from the device. For testing phone, there are 2 constraints on the parameter setting. 1) The minimal supported T_{wd} is as 5ms. 2) T_{wd} cannot be more than 75% of T_{inv}. Otherwise, TWT will be disabled. For ASUS router, it only supports the TWT parameters T_{inv} and T_{wd} as multiple of 8192μs, and T_{wd} must be greater than T_{inv}. Thus, the minimal supported T_{wd} is 8192μs, while the minimal supported T_{inv} is 16384μs. Different devices may have different constraints. As a system, the TWT agreement between the AP and STA needs to satisfy the constrains on both side.

During the test, we evaluate our solutions under different network conditions. For the signal strength, first we did the test in typical region where the received signal strength indication (RSSI) is around -60dBm. Then we also did the test in the edge region, where RSSI goes below -70dBm. Moreover, we evaluated our solution on normal network condition, where all 3 non-TWT devices are turned off. Then, we evaluated our solution on congested network condition, where all 3 non-TWT devices are running iperf speed test in the same band to create congestion. Note that those tests are not done in the anechoic chamber. Therefore, the Wi-Fi traffic generated by other uncontrolled devices are more or less in the environment.

B. TRAFFIC TYPE DETECTION EVALUATION

In this sub-section, we discuss the traffic type detection results for different traffic patterns. We run three applications: YouTube video, Webex video conference and web browsing which corresponds to bursty, stable and random traffic patterns, respectively, as shown in Fig. 19. In the second row of Fig. 19, we show how our SM's internal state (corresponds to Fig. 15) transits when seeing different traffic patterns. The third row of Fig. 19 shows the final classification result of the SM. As shown in Fig. 19, for bursty traffic, the SM's internal state will first transit to state 1 when it detects the first valley of the bursty traffic. Then it will transit to state 3 when the burst part of the bursty traffic is observed. Then it will transit to state 4 after observing another valley of the bursty traffic. At this stage, the $C_{v,b}$ equals to 1, and we then classify the current traffic as bursty traffic. We can also see in the figure that there is one anomaly in the valley part of the burst traffic, whose throughput is obviously higher throughput than the valley, but is also significantly lower than the burst. As our transition condition in state 4 allows such anomaly, the state does not immediately transit to state 0 to classify it as random traffic. For the stable traffic in Fig. 19, we can see that the SM first finds the initial stable region (state 1), and then transits to state 2 after it continues to observe that the traffic is stable without burst or anomalies. Then the traffic is classified as stable. For the random traffic of web browsing in the figure, the SM is not able to find a sufficiently long stable region as well as the pattern of a bursty traffic. Thus, the classification result is continuously to be random. We test the traffic type detection on the three traffic types generated by six different applications for ten minutes each. Overall,

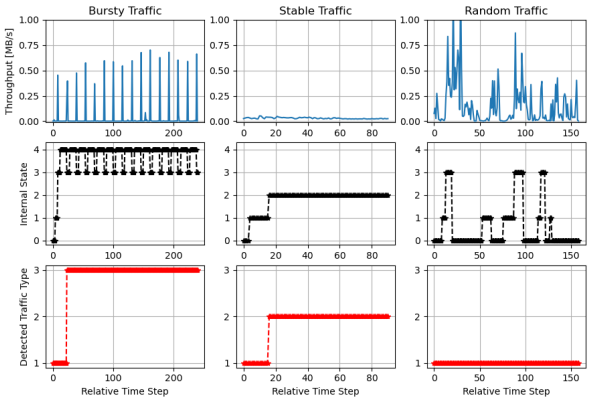


FIGURE 19. Three traffic types and the state-machine detection results. The three columns correspond to the Bursty (YouTube), Stable (Webex), Random (Web browsing) traffic type. The three rows correspond to the throughput, interval state of the SM, and the traffic type detection result. In the traffic detection results, for the y axis value, 1 represents random traffic, 2 represents stable traffic, 3 represents bursty traffic.

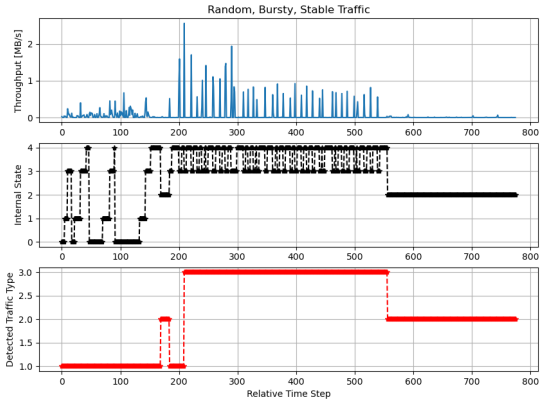


FIGURE 20. Transition between three traffic types. Start with web browsing (random traffic), and switch to YouTubelive video (bursty traffic), and then Pandora audio streaming (stable traffic).

the detection accuracy for bursty, stable and random traffic are 94.2%, 97.5%, and 99.1% respectively.

In Fig. 20, we demonstrate how the SM transits among the three traffic types. We first perform web browsing (random traffic), then wait several seconds and start to play YouTube live video (burst traffic). After that, we play the Pandora audio streaming, which can produce a stable traffic. As can be seen in Fig. 20, after finishing the web browsing and before starting the YouTube live video, the SM observes a period of time that the throughput is stable and then switches to the internal state 2 and classifies the traffic as stable. As the YouTube live video starts, a burst is observed which breaks the stable traffic condition, and the SM immediately switches to the random traffic type. Then, after observing the valley-burst pattern, the traffic is classified as bursty. After the YouTube live video is stopped and the Pandora audio stream starts, the SM observes a continuously stable throughput, and then classifies the traffic as stable.

C. SERVICE TYPE DETECTION EVALUATION

In this sub-section, we discuss about the service type detection results for different categories and in different scenarios. Here in the normal network condition, the $T_{\text{cca}}/T_{\text{on}}$ is below 35%, while in congested network condition, the ratio is in the range of [55%, 75%]. Both T_{cca} and T_{on} value are obtained from Android class (com.android.server.wifi.WifiLinkLayerStats.ChannelStats). All the test results are real time results from the test phone.

First, let us have a look at the test result for layer 1 classification. Four different classifiers are evaluated simultaneously while we are running the applications. They are RF, Conv 1D, RNN and XGBoost. The layer 1 service type detection result in typical region and normal network conditions is as show in Tab. 3. For CG class, we only have one testing application, which is Xbox Game Pass. We test 5 different games playing, each 10 mins. We can see that all the classifiers are able to recognize the CG with 100% accuracy. For RT class, we evaluate 3 mobile gaming APPs (each game 3 mins playing), 3 conference call APPs and 1 instant messenger (IM) APPs. Note that we do not have the data from Teams and AmoungUs for training. For the conference call APPs, audio call (3 mins) and video call (including single-side video, multiple-side video and screen sharing, each case 3 mins) are evaluated separately. For IM, audio and video call are evaluated separately as well, with each 3 mins. Generally speaking, the detection accuracy are high on RT class. The best accuracy (on average 99.7% accuracy) was provided by XGBoost classifier, while RF classifier provides very similar result. Note that the mobile gaming is with slightly higher error. This is because the variation in mobile gaming is relative larger. For NRT class, we evaluated music streaming APPs (Pandora, Spotify), video streaming APPs (Amazon Vidoe, Disney+, TikTok and youtube), file DL and UL, google speed test, web browsing and no App running cases. On averaging, RF classifier provided the best accuracy (98.8%), while XGBoost provided similar accuracy. In typical region but congested network, the test result is similar as typical region normal network condition. From Tab. 4, we can see that the XGBoost classifiers and RF classifier still outperform the other two classifiers. XGBoost classifier reached the best result as 100% accuracy for CG, 99.2% for RT and 97.9% for NRT. Note that RNN showed obvious performance degradation for NRT class. In edge region normal condition, XGBoost classifier and RF classifier still provide good accuracy. And MLP shows significant performance degradation in NRT case. By Tab. 3, Tab. 4 and Tab. 5, we can see that the two ensemble classifiers are quite robust to different network condition, while the two deep learning based classifiers are relative more sensitive to the network condition. Overall, XGBoost classifier provided the best performance with 99.2% average accuracy over 3 classes and different network conditions. CG application is the most sensitive to the latency and XGBoost could reach 100% accuracy in our test. For RT application, the accuracy is above 99%.

TABLE 3. Service detection layer 1 classification test result on typical region, normal network condition.

Layer 1 Class	Application	RF % (RT/CG/NRT)	Conv 1D % (RT/CG/NRT)	RNN % (RT/CG/NRT)	XGBoost % (RT/CG/NRT)	D_r %	D_a %
CG	**Xbox Game Pass**	**0 / 100 / 0**	**0 / 100 / 0**	**0 / 100 / 0**	**0 / 100 / 0**	**100**	**100**
RT	AmoungUs	99.0 / 0 /1.0	44.7 / 0 /55.3	85.1 / 0 / 14.9	100 / 0 / 0	19	12
RT	BrawlStars	97.8 / 0 / 2.2	98.4 / 0 /1.6	97.8 / 0 / 2.2	98.4 / 0 / 1.6	19	12
RT	PUBG	98.7 / 0 / 2.2	97.4 / 1.8 / 0.8	100 / 0 / 0	99.2 / 0 / 0.8	20	13
RT	Teams Audio	100 / 0 / 0	85.5 / 1.6 / 12.9	98.9 / 0 / 1.1	100 / 0 / 0	24	15
RT	Teams Video	99.6 / 0 / 0.4	99.2 / 0 /0.8	99.4 / 0 / 0.6	99.4 / 0 / 0.6	33	22
RT	Webex Audio	100 / 0 / 0	100 / 0 /0	100 / 0 / 0	100 / 0 / 0	20	13
RT	Webex Video	100 / 0 / 0	99.5 / 0 /0.5	99.6 / 0 / 0.4	100 / 0 / 0	31	20
RT	Zoom Audio	100 / 0 / 0	100 / 0 / 0	100 / 0 / 0	100 / 0 / 0	20	13
RT	Zoom Video	100 / 0 / 0	100 / 0 / 0	100 / 0 / 0	100 / 0 / 0	31	20
RT	WhatsApp Audio	100 / 0 / 0	100 / 0 / 0	100 / 0 / 0	100 / 0 / 0	19	12
RT	WhatsApp Video	98.9 / 0 / 1.1	96.5 / 0 / 0.5	100 / 0 / 0	100 / 0 / 0	32	20
RT	**RT Average**	**99.5**	**92.8**	**98.3**	**99.7**	**24.4**	**15.6**
NRT	Pandora	0 / 1.8 / 98.2	0 / 6.9 / 93.1	0 / 8.1 / 91.9	0.6 / 0.9 / 98.5	21	15
NRT	Spotify	0 / 0 / 100	0 / 1.7 / 98.3	1.1 / 0 / 98.9	0 / 0 / 100	20	14
NRT	AmazonVideo	0 / 0 / 100	0 / 3.4 / 96.6	0.5 / 1.9 / 97.6	0 / 1.0 / 99.0	41	34
NRT	Disney+	0 / 0 / 100	0 / 2.4 / 97.6	0 / 1.0 / 99.0	0 / 1.0 / 99.0	27	21
NRT	TikTok	9.5 / 0 / 90.5	0 / 4.7 / 95.4	3.8 / 0 / 96.2	9.3 / 0 / 90.7	28	21
NRT	youtube	0 / 0 / 100	0 / 2.1 / 97.9	0 / 1.1 / 98.9	0 / 0 / 100	24	19
NRT	file DL	0 / 0 / 100	0 / 0 / 100	6.3 / 0 / 93.7	0 / 0 / 100	95	95
NRT	file UL	0 / 0 / 100	0 / 0 / 100	0 / 0 / 100	0 / 0 / 100	37	26
NRT	google Speed test	0 / 0 / 100	0 / 0 / 100	0 / 0 / 100	0 / 0 / 100	58	55
NRT	Web Browsing	1.6 / 0 / 98.4	0.3 / 8.4 / 91.3	4.5 / 4.5 / 91.0	0.8 / 0.6 / 98.6	18	13
NRT	No App	0 / 0 / 100	0 / 0 / 100	0 / 0 / 100	0 / 0 / 100	10	3
NRT	**NRT Average**	**98.8**	**97.2**	**97.0**	**98.7**	**34.5**	**28.6**

TABLE 4. Service detection layer 1 classification test result on typical Region, congested network condition.

Layer 1 Class	Application	RF %	Conv 1D %	RNN %	XGBoost %	D_r %	D_a %
CG	**Xbox Game Pass**	100	100	100	100	100	100
RT	**RT Average**	98.8	95.5	98.5	99.2	31	22
NRT	**NRT Average**	97.4	94.2	87.9	97.9	52	48

TABLE 5. Service detection layer 1 classification test result on edge region, normal network condition.

Layer 1 Class	Application	RF %	Conv 1D %	RNN %	XGBoost %	D_r %	D_a %
CG	**Xbox Game Pass**	100	100	100	100	100	100
RT	**RT Average**	99.1	96.4	98.9	99.4	27.4	18
NRT	**NRT Average**	98.3	89.5	93.8	98.2	42.6	37.8

A rare error on the detection should not impact on the user experience on this type of application. Therefore, we think the layer 1 service raw classification should be able to satisfy the use case of TWT.

If the output of layer 1 classification is RT or NRT, it will go through a layer 2 fine classification. In the layer 2 fine service type detection, there are two individual classification. For RT, we further classify the service types output from classifier into high latency (HL) service (including audio call and mobile gaming) and low latency (LL) service (including video call). We evaluated RNN classifier and RF classifier for layer 2 RT fine classification. The result is as shown in Tab. 6. We can see that RNN provided better accuracy, which is able to classify the HL and LL classes with relative high accuracy (HL 91.8% and LL 98%). Only Teams audio has high probability to be recognized as video call service. This kind of error would cause the system to assign shorter T_{inv}, which only wastes some power, but does not impact the user experience.

TABLE 6. Service detection layer 2 realtime fine classification test result on typical region, normal network condition.

Layer 2 Class	Application	RNN % (HL / LL)	RF % (HL / LL)
HL	AmongUs	100 / 0	100 / 0
HL	BrawlStars	100 / 0	100 / 0
HL	PUBG	100 / 0	92.1 / 7.9
HL	Teams Audio	57.5 / 42.5	40.5 / 59.5
HL	Webex Audio	97.8 / 2.2	63.1 / 36.9
HL	WhatsApp Audio	97.7 / 2.3	71.1 / 28.9
HL	Zoom Audio	95.3 / 4.7	54.1 / 45.9
HL	**HL Average**	**91.8 / 8.2**	**86.8 / 13.2**
LL	Teams Videos	0.3 / 99.7	0.3 / 99.7
LL	Webex Videos	5.8 / 94.2	0 / 96
LL	WhatsApp Videos	0 / 100	0 / 100
LL	Zoom Videos	1.8 / 98.2	0.3 / 99.7
LL	**LL Average**	**2.0 / 98.0**	**1.1 / 98.9**

For NRT, although the offline test result in Fig. 13b was not bad, the errors among YT/W, streaming and no-APP-running were not sufficiently good in online tests. The reason is

TABLE 7. Service detection layer 2 non-realtime fine classification test result on typical region, normal network condition.

Layer 2 Class	Application	XGBoost % (HL / LL)
HL	Amazon Video	100 / 0
HL	Disney+	100 / 0
HL	Netflix	100 / 0
HL	web browsing	99.1 / 0.9
HL	youtube	100 / 0.0
HL	Nothing	100 / 0.0
HL	**HL Average**	**99.8 / 0.2**
LL	File DL	0.0 / 100
LL	File UL	0.2 / 97.8
LL	Google Speed Test	10.3 / 89.7
LL	**LL Average**	**3.5 / 96.5**

we only look at 6s of data, while some traffic gap in the streaming data or web data could be more than 6s. Therefore, we combine these 3 classes into one service type as NRT HL. And the file DL/UL category are treated as NRT LL type. The online test result for NRT layer 2 classification is shown in Tab. 7. With XGBoost classifier, NRT HL reaches 99.8% accuracy, while NRT LL reaches 96.5% accuracy.

Overall both RT and NRT layer 2 classifiers achieved good accuracy (96.5% accuracy on average), which satisfies TWT T_{inv} configuration requirement.

D. CATEGORY BASED TWT SOLUTION

For all the tests we launched, the testers observed same level of quality for both TWT test phone and non-TWT reference phone. The comparison is made by the tester observing the test phone and reference phone side-by-side. This validates that our solution would not cause user experience or QoS degradation.

In Fig. 21, an example is given to show how the proposed solution works. This application in this example is Webex with screen sharing. As shown in Fig. 21, T_{inv} is set to 24ms, which means the application is recognized as RT LL as expected. We also can see that T_{dt} could follow the traffic fluctuation. Ideally, T_{wd} generated by our algorithm should also follow the traffic fluctuation. However, due to the parameter constraint by AP, most of the time T_{wd} equals 8192μs. And when T_{dt} goes up and trigger overflow protection, then the T_{wd} would pump up to prevent further overflow.

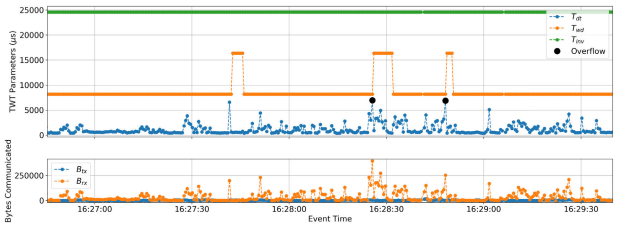


FIGURE 21. An example of how proposed solution works.

The duty cycle for different application in different scenarios are shown in Tab. 3-5. There are two duty cycles

TABLE 8. Burst traffic type customization duty cycle.

Solution	Application	Avg $T_{\text{cca}}/T_{\text{on}}$	Avg RSSI	D_r %
Burst	Youtube Live 1080p	10	-32	13
Random	Youtube Live 1080p	9	-32	14
Burst	Youtube Live 4k	10	-32	20
Random	Youtube Live 4k	11	-32	42
Burst	Youtube Live 1080p	11	-52	17
Random	Youtube Live 1080p	12	-54	19
Burst	Youtube Live 4k	11	-54	24
Random	Youtube Live 4k	11	-53	49
Burst	Youtube Live 1080p	68	-65	20
Random	Youtube Live 1080p	68	-64	37

which are recorded. First duty cycle (D_{a}) is the calculated duty cycle based on our algorithm output. The other duty cycle (D_{r}) is the real duty cycle measured from the test. As the negotiation result might not always be the same as the requested value from STA (e.g., AP rounds up T_{inv} and T_{wd} to multiple of 8192us), D_{r} is always greater than D_{a}. In Tab. 3, the duty cycles for all applications in typical region normal network condition are provided. For CG, it has TWT off. Thus, the duty cycle is 100%. For RT, the average of D_{r} and D_{a} are 24.6% and 15.6% respectively. For NRT, the average of D_{r} and D_{a} are 34.5% and 28.6%, respectively. Therefore, in typical region and normal network condition, which is the most common use case, our solution can reach 29.6% duty cycle averaged cross all RT and NRT applications. For the typical region with high congestion scenario (Tab. 4) and the edge region scenario(Tab. 5), the average duty cycle goes a bit higher as expected. In these two scenarios, other than NRT applications at typical region plus congestion scenario (in which $D_{\text{r}} = 52\%$), all other cases have an average duty cycle smaller than 50%. Overall, our solution could significantly save power in all different scenarios.

In Tab. 8 we can see the duty cycle when compared between the customized solution for bursty traffic and random traffic solution. In this experiment, we let a YouTube live video run for 5 minutes and record the duty cycle for the time the YouTube live video is running. For comparison, we also run a non-TWT device running the same YouTube live video to subjectively evaluate the QoS of the application. From the table we can see that the burst customization performs equal to or better than the random traffic customization for the same Avg RSSI and Avg $T_{\text{cca}}/T_{\text{on}}$. For higher resolutions or low RSSI or high congestion, we can see that the burst customization has almost obvious gain (in best case, the gain is more than 50%) on duty cycle over the random traffic customization.

VII. CONCLUSION

In this paper, we present a category based fully adaptive TWT parameter configuration solution. For the TWT wake interval, we develop a ML based NSD method, which can accurately identify the running network service type. The network service type is defined according to the latency requirement and the traffic characteristics. Subsequently, the TWT wake interval is assigned the value corresponding to the

required latency of the NSD detected service type. For TWT wake duration, a data time estimation model is developed, which can estimate the required TWT wake duration based on the observed throughput, linkspeed and contention level. In addition, a SM based method is developed to detect the three traffic types (random, stable, bursty) for the further optimization of the wake duration based on these traffic types. Finally, our solution is validated in a commercial system. The test results show that the NSD can reach 99.2% accuracy for the layer 1 classification, and 96.5% accuracy for the layer 2 classification. By maintaining the same level of QoS and user experience, our solution can significantly reduce the duty cycle to 29.6% on average in normal network conditions and to less than 50% for most of the other network conditions. In our future research, we plan to further improve the network service detection module and the data time estimation model. Specifically, we will further investigate the distinctiveness among different network services and then try to enable the classification of more service categories in NSD. We shall also try to further improve the NSD's classification accuracy by introducing new feature engineering on the packet statistics. For the data time estimation model, we will further develop better models to provide a continuous mapping of the physical layer data rate to the transport layer throughput under various network conditions, which would replace the current linear segmented mapping model.

REFERENCES

[1] *The 802.11 Work Group*, Standard IEEE 802.11ax D8.0, 2020.
[2] Q. Chen, Z. Weng, X. Xu, and G. Chen, "A target wake time scheduling scheme for uplink multiuser transmission in IEEE 802.11ax-based next generation WLANs," *IEEE Access*, vol. 7, pp. 158207–158222, 2019.
[3] Q. Chen and Y.-H. Zhu, "Scheduling channel access based on target wake time mechanism in 802.11ax WLANs," *IEEE Trans. Wireless Commun.*, vol. 20, no. 3, pp. 1529–1543, Mar. 2021.
[4] H. Yang, D.-J. Deng, and K.-C. Chen, "On energy saving in IEEE 802.11ax," *IEEE Access*, vol. 6, pp. 47546–47556, 2018.
[5] C. S. Bontu and E. Illidge, "DRX mechanism for power saving in LTE," *IEEE Commun. Mag.*, vol. 47, no. 6, pp. 48–55, Jun. 2009.
[6] M. K. Maheshwari, A. Roy, and N. Saxena, "DRX over LAA-LTE—A new design and analysis based on semi-Markov model," *IEEE Trans. Mobile Comput.*, vol. 18, no. 2, pp. 276–289, Feb. 2019.
[7] H. Ramazanali and A. Vinel, "Performance evaluation of LTE/LTE—A DRX: A Markovian approach," *IEEE Internet Things J.*, vol. 3, no. 3, pp. 386–397, Jun. 2016.
[8] A. T. Koc, S. C. Jha, R. Vannithamby, and M. Torlak, "Device power saving and latency optimization in LTE—A networks through DRX configuration," *IEEE Trans. Wireless Commun.*, vol. 13, no. 5, pp. 2614–2625, May 2014.
[9] C.-C. Tseng, H.-C. Wang, F.-C. Kuo, K.-C. Ting, H.-H. Chen, and G.-Y. Chen, "Delay and power consumption in LTE/LTE—A DRX mechanism with mixed short and long cycles," *IEEE Trans. Veh. Technol.*, vol. 65, no. 3, pp. 1721–1734, Mar. 2016.
[10] Y.-P. Yu and K.-T. Feng, "Traffic-based DRX cycles adjustment scheme for 3GPP LTE systems," in *Proc. IEEE 75th Veh. Technol. Conf. (VTC Spring)*, Yokohama, Japan, May 2012, pp. 1–5.
[11] S. C. Jha, A. T. Koç, and R. Vannithamby, "Optimization of discontinuous reception (DRX) for mobile internet applications over LTE," in *Proc. IEEE Veh. Technol. Conf. (VTC Fall)*, Quebec City, QC, Canada, Sep. 2012, pp. 1–5.
[12] S. Herrería-Alonso, M. Rodríguez-Pérez, M. Fernández-Veiga, and C. López-García, "Adaptive DRX scheme to improve energy efficiency in LTE networks with bounded delay," *IEEE J. Sel. Areas Commun.*, vol. 33, no. 12, pp. 2963–2973, Dec. 2015.
[13] M. Polignano, D. Vinella, D. Laselva, J. Wigard, and T. B. Sorensens, "Power savings and QoS impact for VoIP application with DRX/DTX feature in LTE," in *Proc. IEEE 73rd Veh. Technol. Conf. (VTC Spring)*, Budapest, Hungary, May 2011, pp. 1–5.
[14] Y. Fu, H. Xiong, X. Lu, J. Yang, and C. Chen, "Service usage classification with encrypted internet traffic in mobile messaging apps," *IEEE Trans. Mobile Comput.*, vol. 15, no. 11, pp. 2851–2864, Nov. 2016.
[15] A. Arora and S. K. Peddoju, "Minimizing network traffic features for Android mobile malware detection," in *Proc. 18th Int. Conf. Distrib. Comput. Netw.*, Jan. 2017, pp. 1–10.
[16] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé, "Multi-classification approaches for classifying mobile app traffic," *J. Netw. Comput. Appl.*, vol. 103, no. 1, pp. 131–145, Feb. 2018.
[17] S. Zhao, S. Chen, Y. Sun, Z. Cai, and J. Su, "Identifying known and unknown mobile application traffic using a multilevel classifier," *Secur. Commun. Netw.*, vol. 2019, pp. 1–11, Jan. 2019.
[18] M. Nurchis and B. Bellalta, "Target wake time: Scheduled access in IEEE 802.11ax WLANs," *IEEE Wireless Commun.*, vol. 26, no. 2, pp. 142–150, Apr. 2019.
[19] X. Perez-Costa and D. Camps-Mur, "IEEE 802.11E QoS and power saving features overview and analysis of combined performance [accepted from open call]," *IEEE Wireless Commun.*, vol. 17, no. 4, pp. 88–96, Aug. 2010.
[20] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," 2014, *arXiv:1406.1078*. [Online]. Available: https://arxiv.org/abs/1406.1078
[21] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2016.
[22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, and J. Vanderplas, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Oct. 2011.

WENXUN QIU (Member, IEEE) received the B.E. and M.S. degrees in electrical engineering from the Harbin Institute of Technology, in 2005 and 2008, respectively, and the Ph.D. degree in electrical engineering from The University of Texas at Dallas, in 2012. From 2012 to 2017, he was a System Engineer with Texas Instruments Inc. He is currently a Senior Staff Engineer with the Standards and Mobility Innovation Laboratory, Samsung Research America, Plano, TX, USA. His research interests include wireless communication and sensing, network optimization, and machine learning technology.

GUANBO CHEN (Member, IEEE) received the B.E. degree in electrical engineering from the Beijing Institute of Technology, Beijing, China, in 2010, the M.S. degree in electrical engineering from the University of Michigan, Ann Arbor, MI, USA, in 2011, and the Ph.D. degree in electrical engineering from the University of Southern California, Los Angeles, CA, USA, in 2017. From 2017 to 2019, he was a Postdoctoral Research Associate with the Ming Hsieh Department of Electrical Engineering, University of Southern California. He is currently a Senior Research Engineer at Samsung Research America, Plano, TX, USA. His research interests include wireless communication and sensing, inverse problems, and computational electromagnetics.

KHUONG N. NGUYEN (Member, IEEE) received the B.S. degree in computer engineering from The University of Texas at Arlington, TX, USA, in 2014, and the Ph.D. degree in computer science from Texas A&M University, College Station, TX, USA, in 2019. He is currently a Senior Research Engineer with the Standards and Mobility Innovation Laboratory, Samsung Research America, Plano, TX, USA. His research interests include artificial neural networks, reinforcement learning, general artificial intelligence involving cognitive science, and applied machine learning in telecommunication.

ABHISHEK SEHGAL (Member, IEEE) received the B.E. degree in instrumentation technology from Visvesvaraya Technological University, Belgaum, India, in 2012, and the M.S. and Ph.D. degrees in electrical engineering from The University of Texas at Dallas, Richardson, TX, USA, in 2015 and 2019, respectively. He is currently a Senior Research Engineer with the Standards and Mobility Innovation (SMI) Laboratory, Samsung Research America. His research interests include real-time signal processing, pattern recognition, machine learning, and wireless communication. His awards and honors include the Second Place Award for the Hearables Challenge organized by the National Science Foundation (NSF), in 2017.

PESHAL NAYAK (Member, IEEE) received the B.Tech. degree in electrical engineering from IIT Patna, in 2014, and the M.S. and Ph.D. degrees in electrical and computer engineering from Rice University, Houston, TX, USA, in 2016 and 2019, respectively. He currently works as a Senior Engineer in research with Samsung Research America, TX, USA. His research interests include experimental wireless networking, testbed design, and cross-layer protocol design, with a focus on analytical modeling and performance evaluation. He was a recipient of the Texas Instruments Distinguished Student Fellowship. He received the Best-in-Session Presentation Award at the IEEE INFOCOM 2017.

JUNSU CHOI (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical engineering from Seoul National University (SNU), South Korea, in 2010, 2017, and 2017, respectively, through combined master's and doctorate program. In 2017, he joined Samsung, where he is currently a Staff Engineer with the IT and Mobile Business Division. His current research interests include wireless communications, networks, sensing, and deep learning. In 2016, he won the Outstanding Graduate Student Award from the Department of Electrical and Computer Engineering, Seoul National University.