Managing Power Saving Classes in IEEE 802.16 Wireless MANs: A Fold-and-Demultiplex Method

Yu-Chee Tseng, Senior Member, IEEE, Jen-Jee Chen, Member, IEEE, and Yen-Chih Yang

Abstract—In IEEE 802.16, power management at the Mobile Subscriber Station (MSS) side is always an important issue. The standard defines three types of power saving classes (PSCs). A PSC can bind one or multiple traffic flows. However, given multiple flows in an MSS, the standard does not define how to form PSCs, how to organize the cooperation of multiple PSCs to obtain better energy efficiency, and how to guarantee QoS of these flows. Given a set of flows and their QoS parameters, the objective of this paper is to define multiple PSCs and their listen-and-sleep-related parameters and packet-scheduling policy such that the unavailability intervals of the MSS can be maximized and the QoS of each flow can be guaranteed. To achieve this, we propose a novel fold-and-demultiplex method for an IEEE 802.16 network with PSCs of types I and II together with an earliest-next-bandwidth-first packet scheduler. Given a set of traffic flows in an MSS, the fold-and-demultiplex method first gives each flow a tentative PSC satisfying its bandwidth requirement. Then we fold them together into one long series so as to calculate the total bandwidth requirement. Finally, we demultiplex the series into multiple PSCs, each supporting one or multiple flows. It ends up with high energy efficiency of MSSs while meets flows’ bandwidth requirements. Furthermore, our packet scheduler ensures that real-time flows’ delay constraints can be met. To the best of our knowledge, this is the first result offering bounded packet delays under MSS’s sleep-and-listen behaviors.

Index Terms—IEEE 802.16, link protocol, MAC protocol, packet schedule, power management, WiMAX, wireless network,

I. INTRODUCTION

IEEE 802.16/WiMAX [1] has been considered as a promising approach for supporting mobile and broadband wireless access. Similar to most wireless systems, conserving energy is a critical issue for Mobile Subscriber Stations (MSSs). In IEEE 802.16, three types of Power Saving Classes (PSCs) are defined to meet different traffic characteristics. Each PSC consists of a sequence of interleaved listening and sleep windows, and can support one or multiple traffic flows in an MSS with similar characteristics. Type I is designed for non-real-time traffic flows; it has exponentially increasing sleep windows if no packet comes. Type II is designed for real-time traffic flows; it has a fixed size of sleep windows. Type III is designed for multicast connections or management operations. An MSS can turn off its radio interface when all its PSCs are in their sleep windows, but has to wake up when any PSC is in a listening window. From an MSS’s point of view, the standard allows each of its flows to be corresponded to a PSC. However, it does not define how multiple flows can cooperate in a PSC and how multiple PSCs can cooperate with each other for better energy efficiency. At the same time, it needs to answer how to determine the parameters of each PSC, such as start frame, listening window size, and sleep window size, and how to guarantee QoS of traffic flows when multiple PSCs coexist. These motivate us to study the power saving class management in IEEE 802.16 network. (Recently, the IEEE 802.16m [2], [3] task group is developing an amendment as an extension to 802.16-2009 to meet the 4G network requirements. IEEE 802.16m also define similar sleep modes and it is claimed that IEEE 802.16m will be backward-compatible with the original IEEE 802.16 [4].) Several work [5], [6], [7], [8], [9] have conducted performance modeling of 802.16’s power management. These results have provided a potential guidance for setting PSCs’ parameters. However, these schemes all assume non-real-time traffics and most of them consider the arrival patterns to be memoryless, which is not always true in the real world. For PSCs of type I, the authors of [10] propose a Longest-Virtual-Burst-First (LVBF) scheduling to improve the energy efficiency of MSSs, while the authors of [11] present an adaptive scheme to dynamically adjust the initial and the maximum sleep window sizes. The authors of [12] show that setting the initial sleep window to half of the last sleep window can achieve better energy efficiency, while the authors of [13] show how to decide the initial sleep window depending on both the last initial sleep window and the estimated packet inter-arrival time. Assuming that the probability distribution function of the response packet arrival time is known, [14] proposes that an MSS can stay asleep until the expected response packets may arrive to decrease unwanted listening time. For PSCs of type II, recent papers [15], [16] show how to find the Maximum Unavailability Interval (MUI) by only adjusting PSCs’ start frames. However, most of these papers assume that PSCs are already given. They do not discuss a more basic problem: “given a set of connections and their bandwidth and delay requirements, how to form PSCs and how to relate these requirements to PSCs’ listen-and-sleep parameters? Considering real-time connections, the authors in [17] and [18] propose a Periodic On-Off Scheme (PS) to form one PSC of type II for all connections, such that the lengths of sleep and listening windows are constrained by delay and resource requirements. Since only one PSC is used, it may suffer from either longer wake-up time or mismatch of bandwidth requirements. In [19], authors consider sleep scheduling for multiple MSSs. However, for each MSS, the approach is similar to PS. To enhance the work of [15], [16], the authors of [20] incorporate PSC of type I and type II by arranging PSCs of type I to wake up when any type II is awake. By this way, both energy consumption of the MSS and the delay of non-real-time packets can be reduced. However, this violates the exponential sleep behavior of type I defined in the standard [1]. To summarize, our work distinguishes from existing work by considering both real-time and non-real-time connections in an...
MSS to be served by multiple PSCs. Given each connection’s QoS requirements, we need to answer how to from PSCs and how to associate them with those PSCs to meet their QoS requirements.

In this work, we consider PSCs of types I and II between a pair of MSS and BS. Since the PSC of type III is mainly designed for multicast (which means that multiple MSSs need to be involved), it is out of the scope of this work. Given a set of connections and their bandwidth and delay requirements, managing PSCs needs to answer the following questions: 1) How to form PSCs and how to translate these requirements into the listen-and-sleep parameters of PSCs? 2) When two or more connections are associated to the same PSC, how they cooperate with each other to meet their requirements? 3) When there are two or more PSCs, how they cooperate with each other such that the sleep time of the MSS is maximized? In this paper, we propose a novel fold-and-demultiplex method together with an earliest-next-bandwidth-first packet scheduler to address these issues. Given a set of real-time traffic flows in an MSS and their traffic demands and delay constraints, we first try to give each flow a tentative PSC of type II satisfying its requirements. Each PSC actually corresponds to a long series of bandwidths available to the flow. Then we “fold” these tentative PSCs of type II together into one long series so as to calculate the overall bandwidth requirement of the MSS. The fold series is in fact an imaginary PSC. Then we “demultiplex” this imaginary PSC into multiple PSCs of type II, each supporting one or multiple flows. Finally, we include non-real-time flows by adding one PSC of type I. After this novel folding and demultiplexing operations, a set of PSCs of type I and type II are derived. During the sleep mode, new incoming real-time packets will be scheduled for transmission according to their associated PSCs and allocated bandwidths. We show that, under our fold-and-demultiplex method, such an earliest-next-bandwidth-first scheduling rule always ensures packets’ delay constraints. The major contributions of this paper are two-fold. First, the fold-and-demultiplex method trickily balances MSS’s duty cycle and bandwidth requirements by well-arranged PSCs. Second and to the best of our knowledge, it is the first result that offers bounded delays considering stations’ sleep-and-listen behaviors.

This paper is organized as follows. Backgrounds related to our work are presented in Section II. Section III presents our fold-and-demultiplex method. A packet scheduler to cooperate with our method for delay guarantee is presented in Section IV. Section V gives our simulation results. Conclusions are drawn in Section VI.

II. BACKGROUNDS

IEEE 802.16 defines three types of PSCs for an MSS. Type I is to support BE (Best Effort) and NRT-VR (Non-Real-Time Variable Rate) connections. Type II is to support UGS (Unsolicited Grant Service), RT-VR (Real-Time Variable Rate), and ERT-VR (Extended-Real-Time Variable Rate) connections. When an MSS activates its sleep operation, each PSC will switch between listening and sleep windows. Each connection is bound to a PSC. During a sleep window, the corresponding connections cannot send or receive packets. So, when all PSCs of an MSS are in their sleep windows, the MSS can turn off its air interface to save energy. This period is called an unavailability interval of the MSS.

A PSC of type I is denoted by $P_I$. (In the standard, each PSC is associated with a number of parameters. In the following we use programming language-like notations to represent these parameters.) Its sleep windows are interleaved with fixed-length listening windows, each of size $P_I T_S$. Its initial sleep window size is $P_I T_{S_{init}}$, and is doubled each time, until reaching the maximum size, $P_I T_{S_{max}}$, after which it remains the same. During a listening window, the MSS will check if there are incoming packets for $P_I$. If not, $P_I$ will enter another sleep window; otherwise, it will return to normal operation. A PSC of type II is denoted by $P_{II}$. Both its sleep windows and listening windows are of fixed lengths $P_{II} T_{S}$ and $P_{II} T_{L}$, respectively. However, if there is any transmission/reception during a listening window, it will not return to normal operation unless being instructed. A PSC of type III has only one sleep window, after which it will return to normal operation immediately. Fig. 1 illustrates these definitions. The unit of these window sizes is the frame length, which is normally 5 ms [21]. A frame can be divided into a downlink subframe and an uplink subframe.

The BS needs to allocate bandwidths to flows according to their service types. For BE and NRT-VR, this is relatively easier because they have no real-time constraints. For UGS, RT-VR, and ERT-VR, we review three uplink scheduling schemes in the standard. (Note that if PSCs are to be applied to such flows, their setting has to follow these characteristics.) UGS is designed for real-time services with periodical fixed-size packets, such as VoIP. So periodical fixed-size grants will be allocated to a UGS connection. For RT-VR, such as MPEG videos, rtPS (real-time Polling Service) can be used, where periodical fixed uplink resources are allocated to a connection. This allows the connection to ask for more resources when it has burst traffic. For ERT-VR, a connection may have periodical fixed-size packets interleaved by intermittent silence, such as VoIP connections with silence suppression. Then errPS (extended real-time Polling Service) can be used. Initially, a connection has periodical fixed-size grants. Once it has nothing to send, it will inform the BS to reduce each grant size to the minimal one. When the connection becomes active again, it can inform the BS to restore its original allocation. These are controlled by the QoS parameters: Minimum Reserved Traffic Rate (MRTR), Maximum Sustained Traffic Rate (MSTR), Maximum Latency, Unsolicited Grant Interval (UGI), and Unsolicited Polling Interval (UPI). Fig. 2 illustrates some examples.

III. FOLD-AND-DEMULTIPLEX METHOD

We consider an MSS with $U$ non-real-time connections, $M$ real-time uplink connections, $C_i$, $i = 1..M$, and $N$ real-time
to conserve energy. Our goal is to compute a set of PSCs to multiple PSCs of type II. The former may overlap with the latter one PSC of type I and real-time ones are assigned to one or Table I, we summarize all notations used in this paper.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a(t) )</td>
<td>the bandwidth to be allocated to the MSS in the ( t )-th frame</td>
</tr>
<tr>
<td>( C_i )</td>
<td>the ( i )-th real-time connection, ( i = 1..M + N )</td>
</tr>
<tr>
<td>( C_i.D_{\text{max}} )</td>
<td>the delay constraint in milliseconds for real-time connection ( C_i )</td>
</tr>
<tr>
<td>( C_i.L_{\text{d}} )</td>
<td>the expected packet inter-arrival time of real-time connection ( C_i )</td>
</tr>
<tr>
<td>( C_i.I_{\text{pol}, \text{grt}} )</td>
<td>the UPI (resp., UGI) of real-time connection ( C_i ) when ( C_i ) is an uplink RT-VR/ERT-VR (resp., UGS) connection. (This parameter is not used for downlink real-time connections.)</td>
</tr>
<tr>
<td>( C_i.MRTR )</td>
<td>the minimum reserved traffic rate of real-time connection ( C_i )</td>
</tr>
<tr>
<td>( F )</td>
<td>the length of a frame</td>
</tr>
<tr>
<td>( P^I )</td>
<td>the PSC of type I</td>
</tr>
<tr>
<td>( P^I.T_L )</td>
<td>the size of a listening window of a PSC ( P^I )</td>
</tr>
<tr>
<td>( P^I.T_{\text{S,init}} )</td>
<td>the size of the initial sleep window of a PSC ( P^I )</td>
</tr>
<tr>
<td>( P^I.T_{\text{S,max}} )</td>
<td>the size of the maximum sleep window of a PSC ( P^I )</td>
</tr>
<tr>
<td>( P^I.L )</td>
<td>the ( i )-th PSC of type II</td>
</tr>
<tr>
<td>( P^I_i.T_L )</td>
<td>the size of a listening window of a PSC ( P^II )</td>
</tr>
<tr>
<td>( P^I_i.T_S )</td>
<td>the size of a sleep window of a PSC ( P^II )</td>
</tr>
<tr>
<td>( p_{\text{lcm}} )</td>
<td>the least common multiplier of ( T_i ), ( i = 1..M + N ), i.e., ( p_{\text{lcm}} = \text{lcm} { T_i \mid i = 1..M + N } )</td>
</tr>
<tr>
<td>( S_i(t) )</td>
<td>the bandwidth requirement of connection ( C_i ) in frame ( t )</td>
</tr>
<tr>
<td>( \tilde{S}(t) )</td>
<td>the bandwidth requirement of the MSS in frame ( t )</td>
</tr>
<tr>
<td>( \hat{S}(t) )</td>
<td>the binary state of the MSS in frame ( t ), where ( 1 ) and ( 0 ) mean active and sleeping states, respectively</td>
</tr>
<tr>
<td>( T_i )</td>
<td>the wake-up period of real-time connection ( C_i )</td>
</tr>
</tbody>
</table>

![Fig. 2. Bandwidth allocation examples of UGS, rTPS, and eTPS.](image)

downlink connections, \( C_i \), \( i = M + 1..M + N \). Each real-time connection \( C_i \)'s QoS parameters, \( i = 1..M + N \), are already known to the MSS and are summarized as follows:

- \( C_i.D_{\text{max}} \): The delay constraint in milliseconds for real-time connection \( C_i \).
- \( C_i.MRTR \): The minimum reserved traffic rate (bits/sec).
- \( C_i.L_{\text{d}} \): The expected packet inter-arrival time.
- \( C_i.I_{\text{pol}, \text{grt}} \): For an uplink RT-VR/ERT-VR connection, it is the UPI; for an uplink UGS connection, it is the UGI. (This parameter is not used for downlink connections.)

Note that to set up a real-time connection, the BS and the corresponding MSS have to negotiate the bandwidth requirements for the connection, such as MRTR and MSTR (maximum sustained traffic rate). To guarantee the bandwidth requirement of a real-time connection, we choose MRTR as the main index for our scheme to calculate the resource to be reserved. (However, other bandwidth indices, such as MSTR can also be applied to our framework.) In this work, we do not concern about non-real-time connections’ QoS since such traffics have the lowest priority. In Table I, we summarize all notations used in this paper.

In our method, all non-real-time connections are assigned to one PSC of type I and real-time ones are assigned to one or multiple PSCs of type II. The former may overlap with the latter to conserve energy. Our goal is to compute a set of PSCs to maximize the unavailability intervals of the MSS while meet all connections’ QoS requirements. Specifically, for the PSC of type I, denoted by \( P^I \), the following parameters will be determined:

- \( P^I_i.T_L \): Size of a listening window.
- \( P^I_i.T_{\text{S,init}} \): Size of the initial sleep window.
- \( P^I_i.T_{\text{S,max}} \): Size of the maximum sleep window.

For the \( i \)-th PSC of type II, denoted by \( P^II_i \), the following parameters will be determined:

- \( P^II_i.T_L \): Size of a listening window.
- \( P^II_i.T_S \): Size of a sleep window.

Our scheme consists of four steps. First, only real-time connections are considered and one tentative PSC of type II per connection is created. For each \( P^II_i \), we call \( P^II_i.T_L + P^II_i.T_S \) the wake-up period of this PSC. We will pick one PSC with the shortest wake-up period and enforce all other PSCs’ wake-up periods to be integer multiples of the shortest one. Second, we fold all these PSCs into one so as to calculate the overall bandwidth requirement. Third, we demultiplex the above result into multiple real PSCs. The last step will include non-real-time connections.

### A. Creating Tentative PSCs

In this step, only real-time connections are considered. It will compute one tentative PSC for each connection and send the result to the BS via a MOB_SLP-REQ (sleep request) message, where a MOB_SLP-REQ message should contain the PSCs suggested by the MSS and each PSC’s sleep parameters and member connections. It includes three steps: 1) For each \( C_i \), \( i = 1..M + N \), create a tentative PSC of type II according to its QoS parameters. 2) To increase the overlapping of listening windows, adjust these PSCs such that their wake-up periods are integer multiples of the smallest one. 3) Adjust these connections’ QoS parameters to adapt to their sleep behaviors.
packet arrival of C:

\[
\begin{array}{cccccccccccccc}
\text{Frame no.} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17
\end{array}
\]

Fig. 3. A mismatch example between the packet arrival time of a flow and its wake-up period.

1) For each \( C_i \), \( i = 1 \cdots M+N \), we define a tentative PSC \( P_i^{II} \) as follows:

\[
P_i^{II}.T_L = \begin{cases} 
2, & \text{if } C_i \text{ is of type uplink } \text{RT-VR} \\
1, & \text{otherwise,}
\end{cases}
\]

\[
P_i^{II}.T_S = \left( \frac{C_i.D_{\text{max}}}{2 \times F} \right) - P_i^{II}.T_L.
\]

Here \( F \) is the length of a frame. (Normally, the length of an OFDM/OFDMA frame is 5 ms [21].) As mentioned in Section II, for an uplink RT-VR connection, since it requires an additional frame to send its bandwidth request to the BS, its listening window should be two frames. For other types, we assume that one frame is sufficient (later on, we will relax this). We regard \( T_L + T_S \) as the wake-up period of a PSC. Eq. (2) ensures that \( C_i \)'s wake-up period is no more than \( \frac{1}{2} \) of its delay constraint \( C_i.D_{\text{max}}. \) Such setting guarantees bounded delay for each packet of \( C_i \) (refer to Sec. IV).

2) We further adjust these PSCs to increase their overlapping. Let \( p_{\text{min}} = min \{ P_i^{II}.T_L + P_i^{II}.T_S \mid i = 1..M+N \} \), i.e., the smallest wake-up period. We adjust each \( C_i \)'s sleep window size as follows:

\[
P_i^{II}.T_S = \left( \frac{P_i^{II}.T_L + P_i^{II}.T_S}{p_{\text{min}}} \right) \times p_{\text{min}} - P_i^{II}.T_L.
\]

This makes \( C_i \)'s wake-up period an integer multiple of \( p_{\text{min}} \). For example, if there are three PSCs with \( P_i^{II}.T_L = 1, P_2^{II}.T_L = 3, P_3^{II}.T_L = 3 \), \( P_2^{II}.T_S = 1, P_1^{II}.T_S = 4, P_3^{II}.T_S = 4 \), and \( P_1^{II}.T_S = 8 \), then \( p_{\text{min}} = 4 \). After adjustment, \( P_2^{II}.T_S = 3 \) and \( P_3^{II}.T_S = 7 \). Before the adjustment, since the wake-up periods of the three PSCs are co-primes, the sleep behavior of the MSS repeats per 180 frames. In each cycle, there will be 84 listening frames. After the adjustment, there is 1 listening frame per 4 frames.

3) Because of the MSS's sleeping behavior, a flow may need to deliver more traffic during a listening window. So changing its QoS parameters may be needed. Consider the connection in Fig. 3, where the packet inter-arrival time is 4 frames but its wake-up period is 6 frames. A listening window needs to serve one or two packets each time. We need to reserve sufficient bandwidth in each listening window to serve the demand (in this example, it is the size of two packets). Therefore, we suggest to change the MRTR of each \( C_i \) as Eq. (4). Note that the new MRTR is larger than or equal to the original MRTR (represented by \( C_i.MRTR_{old} \)). That is, in our scheme, the MSS may be reserved more bandwidth than needed. Later, in Section V, we will evaluate the utilization of these reserved resources for the MSS by our scheme and compare to other previous schemes. Since \( P_i^{II}.T_S \) has changed, \( C_i \)'s grant interval/polling interval should be changed to:

\[
C_i.I_{\text{pol,grt}} = (P_i^{II}.T_L + P_i^{II}.T_S) \times F.
\]

These changes can be updated by DSC-REQ (dynamic service change request) messages, where a DSC-REQ message should be sent to dynamically change the parameters of an existing flow, such as MRTR, MSTR, maximum latency, grant interval/polling interval, etc. The effect is a slight increase of bandwidth demand with reduction of the buffering delay. In Fig. 3, we will allocate space to deliver \( \left\lceil \frac{g}{s} \right\rceil = 2 \) packets per listening window, so that the new \( MRTR = 2 \times \frac{C_i.MRTR \times 20}{6F} = \frac{4}{3} \times C_i.MRTR \) and the new \( I_{\text{pol,grt}} = 6F. \)

B. Folding PSCs into a State Series

Next, we will fold the above PSCs into an infinite periodical series of real numbers standing for bandwidth requirements. Then depending on the available bandwidth per frame, the real series is converted into a binary series, standing for the active or sleeping state of each frame. Note that the series does not meet the definition of PSC. Later on, we will demultiplex it into actual PSCs. For ease of presentation, we define \( T_i = P_i^{II}.T_L + P_i^{II}.T_S \) (\( C_i \)'s wake-up period) and \( p_{\text{cm}} = \text{lcm} \{ T_i \mid i = 1..M+N \} \) (i.e., the least common multiplier of \( T_i, i = 1..M+N \)).

We first define a series for each \( C_i \). For \( C_i \) of type UGS/ERT-VR, due to its sleeping behavior, the amount of data \( b_i \) that it has to transmit during a listening window is

\[
b_i = C_i.MRTR \times \left( P_i^{II}.T_L + P_i^{II}.T_S \right) \times F.
\]

Assuming frame 0 to be a listening window, \( C_i \)'s bandwidth requirement in each frame can be represented by a periodical series \( S_i^{\text{ert}}(t), t = 0..\infty \):

\[
S_i^{\text{ert}}(t) = \begin{cases} 
b_i, & \text{if } (t \mod T_i) = 0 \\
0, & \text{otherwise.}
\end{cases}
\]

For \( C_i \) of type RT-VR, in each listening window, it requires a fixed bandwidth \( b \) to submit its request in the first frame and, if needed, a bandwidth of \( b_i \) in the second frame. So, its bandwidth requirement can be represented by

\[
S_i^{\text{rt-err}}(t) = \begin{cases} 
b, & \text{if } (t \mod T_i) = 0 \\
b_i, & \text{if } (t \mod T_i) = 1 \\
0, & \text{otherwise.}
\end{cases}
\]

In the following, whenever the context is needed, we may use \( S_i^{\text{ert}}(t) \) and \( S_i^{\text{rt-err}}(t) \) to represent the bandwidth requirement of a connection of type UGS/ERT-VR and RT-VR, respectively. However, when these types are mixed, we may simply use \( S_i(t) \).

Putting all these together, the bandwidth requirement of the MSS in each frame can be written as a series:

\[
\hat{S}(t) = \sum_{t=0} S_i(t).
\]

Note that the above discussion does not distinguish uplink from downlink communications. In fact, \( \hat{S}(t) \) should be separated into
two series, one for uplink and one for downlink. With this understanding, we will still use \( \bar{S}(t) \) for simplicity. Since each \( S_i(t), i = 1..M + N \), has a period of \( T_i \), it also has a period of \( p_{\text{ICM}} \). It follows that the summation of them also has a period of \( p_{\text{ICM}} \).

Lemma 3.1: \( \bar{S}(t) \) is a periodical binary series with period \( p_{\text{ICM}} \).

Next, we convert the real series \( \bar{S}(t) \) to a binary state series \( \tilde{S}(t) \), where 1 and 0 mean active and sleeping states, respectively:

\[
\tilde{S}(t) = \begin{cases} 1, & \text{if } a(t) > 0 \\ 0, & \text{otherwise} \end{cases}
\]

where \( a(t) = \min \left\{ \sum_{k=0}^{t} \bar{S}(k) - \sum_{k=0}^{t-1} a(k), B \right\} \) is the bandwidth to be allocated to the MSS in the \( t \)-th frame and \( B \) is the maximum bandwidth that can be allocated to the MSS in a frame (this is decided by the BS). Intuitively, \( \sum_{k=0}^{t} \bar{S}(k) \) is the total bandwidth required up to the \( t \)-th frame and \( \sum_{k=0}^{t-1} a(k) \) is the actual bandwidth consumed by the MSS up to the \((t-1)\)-th frame. So the first term in \( \min() \) is the actual bandwidth required in the \( t \)-th frame; however, the actual allocation should be bounded by \( B \). If \( a(t) > 0 \), the MSS should be active in the \( t \)-th frame, enforcing \( \tilde{S}(t) = 1 \); otherwise, it can go to sleep, making \( \tilde{S}(t) = 0 \).

Fig. 4 is an example. Fig. 4(a) shows the actual resource requirement per frame from three connections (i.e., \( \bar{S}(t) \)). Fig. 4(b) shows the actual resource allocation in each frame (i.e., \( a(t) \)). Fig. 4(c) is the state series \( \tilde{S}(t) \) of the MSS.

Lemma 3.2: \( \bar{S}(t) \) is a periodical binary series with period \( p_{\text{ICM}} \).

Proof: \( \bar{S}(t) \) is a binary series which alternates between continuous 1s and continuous 0s infinitely. Consider each group of continuous 1s in \( \bar{S}(t) \); let it start from position \( t_{\text{start}} \) and end at position \( t_{\text{end}} \). Let us define \( C \) as the set of connections, each of which has a non-zero bandwidth requirement during the interval \([t_{\text{start}} : t_{\text{end}}]\), i.e., \( C = \left\{ C_i \mid \sum_{j=t_{\text{start}}}^{t_{\text{end}}} S_i(j) > 0 \right\} \). Intuitively, each connection in \( C \) contributes to the wake-up behavior (continuous 1s) during the interval \([t_{\text{start}} : t_{\text{end}}]\). Since each \( S_i(j) \) corresponding to \( C_i \in C \) is periodical, the same bandwidth requirements from \( C \) occurring during the interval \([t_{\text{start}} : t_{\text{end}}]\) must occur again in interval \([t_{\text{start}} + i \times p_{\text{ICM}} : t_{\text{end}} + i \times p_{\text{ICM}}]\), where \( i \in \mathbb{Z} \). Therefore, \( \bar{S}(t) \) must contain all 1s in interval \([t_{\text{start}} + i \times p_{\text{ICM}}, t_{\text{end}} + i \times p_{\text{ICM}}]\) for each \( i \in \mathbb{Z} \). It follows that the theorem is true.

C. Demultiplexing the State Series into PSCs

Although \( \bar{S}(t) \) is periodical (Lemma 3.2), if we look at any subsequence of length \( p_{\text{ICM}} \) in \( \bar{S}(t) \), it may contain multiple groups of continuous 1s interleaved by 0s and thus does not fit into the definition of PSC. Below, we show how to demultiplex \( \bar{S}(t) \) into multiple state series, each meeting the definition of PSC. By Lemma 3.2, a straightforward approach is to pick the first \( p_{\text{ICM}} \) bits of \( \bar{S}(t) \) and let each group of continuous 1s be a PSC (this is in fact how the proof of Lemma 3.2 works). However, this may generate too many PSCs. Still, using the first \( p_{\text{ICM}} \) bits of \( \bar{S}(t) \), we propose a scheme based on folding \( \bar{S}(t) \) to put duplicate continuous 1s together. This may result in less PSCs.

1) Denote the first \( p_{\text{ICM}} \) bits of \( \bar{S}(t) \) by \( \bar{S}[0:p_{\text{ICM}} - 1] \). Our goal is to construct a set \( C \) of PSCs. Initially, let \( C = \emptyset \).

2) For \( i = 1 \) to \( \frac{p_{\text{ICM}}}{\text{min}} \) do

   a) If \( p_{\text{ICM}} \) is not divisible by \( i \times \text{min} \), skip this \( i \) and go back step 2. Otherwise, cut \( \bar{S}[0:p_{\text{ICM}} - 1] \) into \( \frac{p_{\text{ICM}}}{i \times \text{min}} \) segments, each of length \( i \times \text{min} \) and then fold them together by the bitwise-AND operator into a \((i \times \text{min})\)-bit string \( T[0:i \times \text{min} - 1] \), i.e.,

   \[
   T[j] = \bar{S}(j) \land \bar{S}(j + i \times \text{min}) \land \bar{S}(j + 2i \times \text{min}) \land \cdots,
   \]

   for \( j = 0..i \times \text{min} - 1 \).

   b) Scan string \( T[0:i \times \text{min} - 1] \) from left to right and consider each group of 1s in the string. Suppose that there is a group of 1s starting from the \( x \)-th bit to the \( y \)-th bit. Let \( T'[0:i \times \text{min} - 1] \) be a binary string which has all 1s from the \( x \)-th bit to the \( y \)-th bit and all 0s in the other places. We try to form a PSC from \( T'[0:i \times \text{min} - 1] \) as follows.

\[
C_i.\text{MRTR} = \left\lceil \frac{\text{max. no. of arrivals per wake-up period}}{\left( P_{i}^{11} T_L + P_{i}^{11} T_S \right) \times F} \right\rceil \times \left( C_i.\text{MRTR}_{\text{std}} \times C_i.\text{Id} \right) \times \left( \frac{C_i.\text{MRTR}_{\text{std}} \times C_i.\text{Id}}{1000} \right) \times \left( \frac{1000}{P_{i}^{11} T_L + P_{i}^{11} T_S} \right) \times \left( \frac{1000}{P_{i}^{11} T_L + P_{i}^{11} T_S} \right) \times F^*.
\]
i) Check if $T' \mid [0 : i \times p_{min} - 1]$ is redundant by calling the procedure

$Check\_Redundancy(C, T' \mid [0 : i \times p_{min} - 1])$.

ii) If the response is negative (i.e., not redundant), then add the string $T' \mid [0 : i \times p_{min} - 1]$ to $C$.

3) For each string $str$ in $C$, we generate a PSC of type II. Let $str$ have 1s from the $x$-th bit to the $y$-th bit. The PSC can be defined by setting $T_L = y - x + 1$ and $T_S = |str| - T_L$.

Procedure $Check\_Redundancy()$ is shown below. The binary string $Tested\_Str$, which stands for a potential PSC, is redundant if each of its 1s already appears in a PSC in $C$. The PSCs in $C$ are converted to a string $W[0 : plen - 1]$ by “bitwise-OR” the strings of all PSCs. Similarly, for example, $Tested\_Str$ is converted to a string $W'[0 : plen - 1]$. Then we compare $W[\ ]$ and $W'[\ ]$ to see if $Tested\_Str$ is redundant.

Procedure $Check\_Redundancy(C, Tested\_Str)$:

1) Let $W[0 : plen - 1]$ and $W'[0 : plen - 1]$ be two binary arrays. For $j = 0$ to $plen - 1$, we define

$W[j] = \forall_{Str \in C} j \ mod \ |Str|$

$W'[j] = \begin{cases} 1, & \text{if } Tested\_Str[j \ mod \ |Tested\_Str|] = 1 \\ 0, & \text{otherwise.} \end{cases}$

2) If $W'[j] = 1$ implies $W[j] = 1$ for all $j = 0..plen - 1$, then a “positive” response is returned; otherwise, a “negative” response is returned.

The new definitions of PSCs can be notified to the MSS by MOB_SLP-RSPs (sleep response) message, where a MOB_SLP-RSP message should contain the definitions of PSCs for the MSS and each PSC’s member connections. Similar to the previous procedure, for example, Fig. 5(a) shows a state series $S(t)$ with $plen = 18$ and $p_{min} = 3$. In Fig. 5(b), $S(t)$ is cut into segments, each of 3 bits. By “bitwise-AND” these segments, we get a string $T[0 : 2] = 100$ and then retrieve a $T'$ string 100. Since $C = \emptyset$ (empty set), $Check\_Redundancy()$ returns a “negative” and a PSC of type II with $T_L = 1$ and $T_S = 2$ is added to $C$. In the next iteration, $S(t)$ is cut into segments, each of 6 bits. By “bitwise-AND” these segments, we have $T[0 : 5] = 100100$. Then we retrieve two $T'$ strings, 100000 and 000100, and call $Check\_Redundancy()$ twice. Both strings are redundant because their active patterns have already been covered by string 100. Next, $S(t)$ is cut into segments, each of 9 bits. A binary string $T[0 : 8] = 111111000$ is obtained, from which two $T'$ strings, 111110000 and 000000100, can be retrieved. By calling $Check\_Redundancy()$, we know the former is not redundant but the latter is, so one more PSC of type II with $T_L = 5$ and $T_S = 4$ is added to $C$. In the last iteration, the string $S[0 : plen - 1]$ does not add any new PSC. The two final PSCs are shown in Fig. 5(c).

D. Including a PSC of Type I for Non-real-time Connections

For non-real-time connections, we will define only one PSC of type I, $P^I$, for all of them. Whenever $P^I$ enters a listening window, the BS will send the MSS a broadcasting MOB_TRF-IND message containing a bitmap to indicate whether there are packets buffered at the BS. If there are packets for the MSS, $P^I$ will be deactivated until all packets are transmitted. Then $P^I$ will be re-activated from the initial sleep window.

Recall that we already construct a set $C$ of PSCs of type II with a basic wake-up period of $p_{min}$ frames. Assuming that $P^I$ is more likely to enter the maximum sleep window of $T_{S_{max}}$ (considering that these are non-real-time traffics), we will try to schedule the reactivation time of $P^I$ such that the periodical (maximum) listening windows of $P^I$ will align with the listening windows controlled by $p_{min}$. More specifically, we will tune the parameters of $P^I$ such that $P^I.T_{S_{max}} + P^I.T_L$ is an integer multiple of $p_{min}$.

The problem is formulated as follows: We are given the initial values of $T_L$, $T_{S_{init}}$, and $T_{S_{max}}$. We assume that $T_{S_{max}} \geq p_{min}$ (it is unlikely that $T_{S_{max}} < p_{min}$ considering that these are non-real-time traffics). Without loss of generality, assume that the PSCs in $C$ have common active frames appearing at integer multiples of $p_{min}$ and at frame $t$ the MSS intends to reactivate its $P^I$. Our goal is to find a waiting interval $\Delta t$ and modified parameters $T'_{S_{init}}$, $T'_{S_{max}}$, and $T'_L$ such that $P^I$ will actually enter its sleep window at frame $(t + \Delta t)$ and when $P^I$'s sleep window size reaches $T'_{S_{max}}$, its listening windows will appear at frame numbers that are integer multiples of $p_{min}$. To achieve this goal, we first set

$$T'_{S_{init}} = T_{S_{init}}.$$  

$$T'_L = 1.$$  

$$T'_{S_{max}} = \left\lfloor \frac{T_{S_{max}}}{p_{min}} \right\rfloor \times p_{min} - T'_L.$$  

So $T'_{S_{max}} + T'_L$ is divisible by $p_{min}$. Now, if there is no packet arrival, $P^I$ will enter its first sleep window at frame $(t + \Delta t)$, sleep for $T'_{S_{init}}$ frames, wake up for 1 frame, sleep for $2 \times T'_{S_{init}}$ frames, wake up for 1 frame, sleep for $4 \times T'_{S_{init}}$ frames, ···, until reaching the maximum sleep window size of $T'_{S_{max}}$. If so, the first listening window after the first maximum sleep window will appear at frame number

$$t_{flw} = t + P^I.T'_{S_{init}} + 1 + 2 \times P^I.T'_L + \cdots + P^I.T'_{S_{max}} + 1.$$  

So $\Delta t$ can be set to the smallest number such that $t_{flw} + \Delta t$ is divisible by $p_{min}$. The above concept is illustrated in Fig. 6.

An alternative is to not always start with $T'_{S_{init}}$, but instead start with a larger $2^j \times T'_{S_{init}}$ for some $j$. We can easily rewrite Eq.(11) to identify a new $\Delta t'$ with the same alignment property.

IV. QoS-Guaranteed Packet Scheduler

The above fold-and-demultiplex method can form a set $C$ of PSCs that reserve sufficient bandwidths for connections while achieving energy efficiency. However, when packets from multiple connections arrive at the same time, it is still unclear how to schedule their transmissions to ensure their delay constraints. In this section, we propose a packet scheduler that can guarantee bounded delays for packets under the fold-and-demultiplex method.

Recall that our method will create two tentative series $S(t)$ and $S(t)$ to represent the MSS’s bandwidth requirement and listening windows. We will prove our result through these two series. Consider the $S(t)$ in Fig. 7. Suppose a packet Data_i of connection $C_i$ arriving at frame $t_a$. To analyze the delay that Data_i may experience, consider $C_i$’s bandwidth requirement series $S(t)$. Let
Fig. 5. Example of the PSC demultiplexing procedure.

Fig. 6. Inserting a waiting interval $\Delta t$ such that after $P^I$ reaching $T'_{S_{\text{max}}}$, the listening windows of $P^I$ will align with the listening windows of PSCs in $C$.

$t_b$ be the first frame after $t_a$ such that $S_i(t_b) \neq 0$. Conceptually, an amount of $S_i(t_b)$ bandwidth will be allocated to $C_i$ at frame $t_b$ to deliver $Data_i$ (refer to Fig. 7). However, it is clear from our method that the aggregated bandwidth requirement $S(b)$ may exceed the capacity $B$, so it may take several frames to serve $S_i(t_b)$. Let $\Delta t$ be the number of frames that the request $S_i(t_b)$ is served. Then the total delay experienced by $Data_i$ is $((t_b - t_a) + \Delta t)$ frames. The first part $(t_b - t_a)$ can be bounded, while the second part $(\Delta t)$ actually depends on the scheduler. Below, we will propose a scheduler that ensures $((t_b - t_a) + \Delta t) \leq C_i.D_{\text{max}}$.

The proposed scheduler schedules data for transmission following an earliest-next-bandwidth-first policy. For any $Data_i$ of $C_i$ arriving at frame $t_a$, let $t_b$ be the first frame after $t_a$ such...
that \( S_i(t_b) > 0 \). We assign a priority \( pty(Data_i) = t_b + T_i \) to \( Data_i \). Recall that \( T_i \) is the period of the tentative PSC of type II of \( C_i \). Also note that \( t_b + T_i \) is the next frame such that \( S_i(t_b + T_i) > 0 \). Here a lower number means a higher priority.

Then the scheduler simply schedules data for transmission in each active frame according to their priorities. Such a scheduling guarantees that \( Data_i \) can be completely delivered before frame \( t_b + T_i \). Combining the following theorem and the fact that \( C_i, D_{\max} \geq 2T_i \times F \) (refer to Eq. (2)), our scheme ensures each packet’s delay bound.

**Theorem 4.1:** Under the fold-and-demultiplex method, the earliest-next-bandwidth-first policy guarantees that if data all arrives as planned, then each packet’s delay is bounded by \( 2T_i \) frames, where \( T_i \) is the wake-up period of the corresponding PSC \( P_i^{II} \).

**Proof:** We use the scenario in Fig. 7 to develop our proof. Since \( t_b \) is the first frame after \( t_a \) such that \( S_i(t_b) > 0 \), it is clear that \( t_b - t_a \leq T_i \). So we only need to prove that \( \Delta t \leq T_i \).

We develop an “imaginary” scheduling that guarantees \( \Delta t \leq T_i \) and then show that our policy cannot do worse than it, so our policy ensures \( \Delta t \leq T_i \). The imaginary scheduler assumes that data is infinitely divisible, so all data of \( C_i \) associated to \( S_i(t_b) \) can be evenly served by frames \( t_b, t_b + 1, \ldots, t_b + T_i - 1 \).

For example, Fig. 8(a) shows how it works to serve each \( C_i \)’s data by enforcing each frame to share the same amount of data \( S_i(t_b) \) for \( C_i \). Clearly, such a scheduling is feasible because the total load in each frame cannot exceed \( B \) (otherwise, the BS will be overloaded) and the delay of \( Data_i \) is exactly \( T_i \) frames.

Now, with our earliest-next-bandwidth-first policy, there are two possible changes: (i) data may be moved to the free space of an earlier frame but cannot be earlier than its associated bandwidth requirement \( S_i(t) \) and (ii) data with a higher priority may squeeze into the space of data with a lower priority. The former has no impact on delay bound. The latter has impact on those with lower priorities. However, if the space of lower-priority data is exchanged with the space of higher-priority data, the former can still make the required delay bound because the latter is already bounded by a tighter delay bound (this is what “earliest-next-bandwidth” means). Fig. 8(b) illustrates the ideas. The arrows on the left-hand side indicate the data movement directions (item (i)). Note that these data are moved toward their earlier free spaces, but cannot cross the locations of their associated requirements. Arrows on the right-hand side are the data exchanges due to their priorities (item (ii)). In this example, high-priority \( Data_{k} \) at frame \( t_b + 1 \) with \( pty(Data_k) = t_b + T_k \) is exchanged with low-priority \( Data_i \) at frame \( t_b \) with \( pty(Data_i) = t_b + T_i \). \( Data_i \) can still make its deadline because the space of \( Data_k \) already meets a deadline, which is earlier than \( Data_i \)’s. It follows that \( \Delta t \leq T_i \) for all \( C_i \)’s.

**V. Simulation Results**

We have developed a simulator in C to evaluate the performance of the proposed scheme. In our simulation, the frame length \( F = 5 \) ms and the resource \( B \) that can be allocated per frame is a fixed amount in each simulation round. Between the BS and the MSS, we define six real-time flow types with different QoS parameters, where the six types of real-time flows are shown in Table II (these parameters are set according to [17]). The directions of flows of types III-VI are randomly decided as down or up. Types I and II have the same QoS parameters and differ in their directions. For non-real-time flows, we assume their packet arrival following the Poisson distribution. We compare our (FD) scheme against the scheme in the standard [1] and the scheme in [18] (denoted as STD and PS, respectively). Note that PS uses one single PSC of type II to serve all real-time flows and the wake-up period of this PSC is enforced to be smaller than or equal to the minimum delay constraint of all flows to guarantee their delay constraints. Moreover, the length of the listening window of the PSC in PS is decided by the maximum possible arrival data during a wake-up period of the PSC. As to performance metrics, we use sleep ratio \( (r_s) \), resource utilization \( (U) \), i.e., the ratio of the amount of bandwidth consumed by the MSS to the total amount of bandwidth allocated to it), average delay, and jitter to make comparisons. For non-real-time traffic, only sleep ratio and
TABLE II

SIX TYPES OF REAL-TIME FLOWS USED IN THE SIMULATION

<table>
<thead>
<tr>
<th>Flow type</th>
<th>Service type</th>
<th>Direction</th>
<th>Delay constraint (ms)</th>
<th>Packet inter-arrival time (ms)</th>
<th>Packet size (byte/frame)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ERT-VR</td>
<td>DOWN</td>
<td>50</td>
<td>20</td>
<td>160</td>
</tr>
<tr>
<td>II</td>
<td>ERT-VR</td>
<td>UP</td>
<td>50</td>
<td>20</td>
<td>160</td>
</tr>
<tr>
<td>III</td>
<td>UGS</td>
<td>DOWN/UP</td>
<td>60</td>
<td>20</td>
<td>160</td>
</tr>
<tr>
<td>IV</td>
<td>UGS</td>
<td>DOWN/UP</td>
<td>300</td>
<td>80</td>
<td>500</td>
</tr>
<tr>
<td>V</td>
<td>RT-VR</td>
<td>DOWN/UP</td>
<td>300</td>
<td>35</td>
<td>440</td>
</tr>
<tr>
<td>VI</td>
<td>RT-VR</td>
<td>DOWN/UP</td>
<td>800</td>
<td>200</td>
<td>200</td>
</tr>
</tbody>
</table>

Fig. 9. sleep ratio \( r_s \) vs. \( B \) with (a) two real-time flow sets and (b) three real-time flow sets.

Fig. 10. sleep ratio \( r_s \) vs. number of real-time flow sets with (a) \( B = 1250 \) byte/frame and (b) \( B = 2000 \) byte/frame.

Fig. 11. resource utilization \( U \) vs. \( B \) with (a) two real-time flow sets and (b) three real-time flow sets.

A. Impact of B and Traffic Load

Fig.9(a) and Fig.9(b) plot sleep ratio \( r_s \) against \( B \) for STD, PS, and FD schemes with two and three real-time flow sets, respectively. As \( B \) increases, sleep ratio \( r_s \) increases for both PS and FD. FD always performs the best, followed by PS. This is because FD uses multiple PSCs to manage the sleeping behavior of the MSS while PS uses one PSC. This is particularly important when flows have different QoS characteristics. Since PS uses only one PSC, it has to reserve the maximum required resource in each cycle, thus wasting lots of resources sometimes. STD uses too many PSCs, which are not synchronized, leading to very low sleep ratio \( r_s \). By varying the number of real-time flow sets and fixing \( B = 1250 \) (resp., \( B = 2000 \) ) byte/frame, Fig.10(a) (resp., Fig.10(b)) shows the obtained sleep ratio \( r_s \). Naturally, sleep ratio \( r_s \) decreases as the number of real-time flow sets increases. FD can sustain up to 3 (resp., 5) sets when \( B = 1250 \) (resp., \( B = 2000 \)). On the contrary, with 3 (resp., 4) sets, PS will keep the MSS always awake when \( B = 1250 \) (resp., \( B = 2000 \)). This is because PS over-estimates the potential traffic in some frames. Such an effect is even more serious when there exist larger differences among flows’ delay constraints and inter-arrival times.

Fig.11(a) and Fig.11(b) illustrate the resource utilization \( U \) by varying \( B \) when there are 2 and 3 real-time flow sets, respectively. We see that FD always outperforms PS, due to PS’s over-estimation on resource demands, except at the point \( B = 750 \) in Fig.11(a) and at the point \( B = 1250 \) in Fig.11(b). Note that when \( B \leq 750 \) and \( B \leq 1250 \) in Fig.11(a) and Fig.11(b), respectively, PS will become invalid and will not allow the MSS to enter sleep mode (this fact has been reflected in Fig.9). Since we assume that an MSS in the normal mode can always request the exact amount of resources that it actually needs, the resource utilization \( U \) of the MSS will be 100% (note that this is at the cost of the MSS always staying awake). That’s why PS looks like showing a better resource utilization than FD at these two situations. By assuming a perfect network environment, Fig.12(a) and Fig.12(b) plot average delay and jitter against \( B \), respectively, where jitter is defined as the standard deviation of packet delivery delay. We see that FD causes higher average delay and jitter in all cases. This is the cost of our scheduling because it will buffer packets with larger delay constraints for future delivery. However, as has been proved in Theorem 4.1, this is acceptable because no packet will violate its specified delay constraint\(^1\). In both Fig.12(a) and Fig.12(b), we can see a jump as \( B \) is increased to 2000 bytes/frame. This is because the number of sleep frames of the MSS are suddenly increased as \( B \) reaches 2000 bytes/frame (which can be observed in Fig.9(a)) such that the arriving packets during the sleep frames of the MSS have to wait for more frames until the MSS is awake. After \( B \geq 2000 \), we can see that both the average delay and jitter of the MSS decrease as \( B \) increases. This is because, as \( B \) is larger, more buffered packets accumulated during the sleep window of the MSS can be delivered per active frame. So the delays of these buffered packets will be reduced.

B. Impact of Flow Arrival Pattern

The above discussion considers flow types of fixed combination. Below, we use the six types of real-time flows defined in Table II again, but different from the previous experiment, the flows of each type arrive at a Poisson process with rate \( \lambda_r \) (number of flows/sec) and the hold time of each flow is exponentially distributed with mean \( 1/\mu \) (secs). Letting \( \rho = \frac{\lambda_r}{\mu} \), where ‘\( \rho \)’ means the above 6 types of real-time flows under

\(^1\)For connections of type UGS, IEEE 802.16 includes tolerated jitter as one of its QoS parameters (this parameter is not included for other service types). To avoid jitter, using a playback buffer is a common solution to guarantee the perceived quality of applications at the user side [22]. In our experience, for voice applications, a playback buffer of size of 3 to 5 packets is a common setting [23], [24].
consideration, we will observe the impact of ρ on performance. Note that here a higher ρ means a higher traffic load to the MSS. With B = 1250, Fig.13(a) shows the impact of ρ on sleep ratio r_s. When ρ is small, PS performs slightly better than FD. After ρ ≥ 5, FD outperforms PS because a larger ρ causes multiple flows coexisting to show FD’s advantage. On the other hand, when ρ is small, there is usually 1 or very few flows between the MSS and the BS. Compared with PS’s wake-up period, which is close to or even equal to the most strict delay constraint of flows, FD’s wake-up period is smaller than or equal to half of the flow’s delay constraint. This makes FD awake longer than PS when ρ is small. Fig.13(b) plots resource utilization U against ρ. When ρ < 1, PS performs better than FD; after ρ > 1, FD becomes better; but after ρ ≥ 20, PS outperforms FD again. Under ρ < 1, there is less than 1 flow in average; since PS uses the delay constraint of the flow as the sleep cycle, it can get better utilization. As there are more flows, FD performs better than PS due to its multi-PSC capability. However, when there are too many flows (ρ ≥ 20), PS outperforms FD again because PS would disable the sleep mode more often than FD does (here we assume that the resource utilization is 1 when the sleep mode is disabled because resource can be accurately allocated). Fig.14 sets B = 2000 and shows similar results to Fig.13. In Fig.14(a), we can see that a larger B makes FD outperform PS after ρ ≥ 7 which is larger than the value of 5 in the case of B = 1250 (Fig.13(a)). This is because a larger B has a higher tolerance on PS’s over-estimation on resource demand.

C. Sleep Performance by Including Non-real-time Traffic

In this subsection, we consider one real-time flow set with a non-real-time downlink traffic flow with packet arrival rate λ_n. To verify the effectiveness of our scheme in handling non-real-time traffic, we simulate FD with and without including a PSC of type I, which are termed as “FD with type I” and “FD w/o type I” in Fig.15 and 16, respectively. In “FD with type I”, the procedure in Section III-D is executed to include a PSC of type I for non-real-time traffic, while in “FD w/o type I”, the initial values of T_{S_{init}}, T_I = 1, and T_{S_{max}} are directly used by the type I PSC to serve non-real-time traffic. Fig.15 shows sleep ratio r_s and the average response time experienced by non-real-time packets. A larger packet arrival rate λ_n will degrade sleep ratio r_s. When T_{S_{max}} is smaller, adding a PSC of type I can significantly save energy because it is easier to reach the maximum sleep window, after which the type I PSC can reuse the active frames of type II PSCs. In terms of response time, using a type I PSC is always beneficial because FD actually uses a smaller maximum sleep window size than the original given one for the type I PSC. Fig.16 sets B = 2000 and shows similar results to Fig.15.

VI. CONCLUSION

In this paper, we have proposed a novel sleep scheduling scheme called Fold-and-Demultiplex method, which conforms to the sleep mechanism and message formats defined in IEEE 802.16. The scheme considers the delay constraint, packet inter-arrival time, and data rate of connections to determine the parameters of PSCs. Multiple PSCs of type II are used to capture the sleep-active behavior contributed by real-time flows. One PSC of type I is used to handle non-real-time flows. We have also
proposed an earliest-next-bandwidth-first scheduler, which can guarantee the real-time flows’ delay constraints. Simulation results show that our scheme can save the MSS’s energy even when there are many real-time flows coexist while keeping bandwidth utilization high under real-time flows’ delay constraints. In our scheme, the computation of PSCs is based on a given set of flows. This implies that the set of PSCs may need to be recomputed whenever a new connection starts, an old connection terminates, or any change of an existing flow’s requirement. Therefore, one future direction would be how to dynamically adjust some PSCs to adapt to such changes, rather than re-executing the whole scheme again.

REFERENCES


Yu-Chee Tseng obtained his Ph.D. in Computer and Information Science from the Ohio State University in January of 1994. He is Professor (2000-present), Chairman (2005–2009), and Associate Dean (2007–present) at the Department of Computer Science, National Chiao Tung University, Taiwan. He is also Adjunct Chair Professor at the Chung Yuan Christian University (2006-present). Dr. Tseng received three times the Outstanding Research Award, NSC (2001, 2003, and 2009), the Best Paper Award (Int’l Conf. on Parallel Processing, 2003), the Elite I. T. Award in 2004, and the Distinguished Alumnus Award, by the Ohio State University, in 2005. His research interests include mobile computing, wireless communication, and parallel and distributed computing. Dr. Tseng serves on the editorial boards for Telecommunication Systems (2005-present), IEEE Trans. on Vehicular Technology (2005-2009), IEEE Trans. on Mobile Computing (2006-present), and IEEE Trans. on Parallel and Distributed Systems (2008-present).

Jen-Jee Chen received his B.S. and M.S. degrees in Computer Science and Information Engineering in 2001 and 2003, respectively, from National Chiao Tung University, Hsinchu, Taiwan, and his Ph.D. degree in Computer Science in 2009 from National Chiao Tung University, Hsinchu, Taiwan. He was a Visiting Scholar at the University of Illinois at Urbana-Champaign during the 2007-2008 academic year. Currently, he is a postdoctoral research fellow at the Department of Computer Science, National Chiao Tung University, Taiwan. His research interests include wireless communication, mobile computing, personal communication service, cross-layer design, and network performance modeling and analysis.

Yen-Chih Yang received his B.S. and M.S. degrees in Computer Science from National Chiao Tung University in 2006 and 2008, respectively. His research interests include wireless communication and mobile computing. He has been with MediaTek Inc., Taipei, Taiwan, as a software engineer since 2009.