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.

1 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 networks. (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 interarrival 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*

- *Y.-C. Tseng is with the Department of Computer Science, National Chiao Tung University, Hsinchu 30010, Taiwan, and also with the Research Center for Information Technology Innovation, Academia Sinica, Taipei 11529, Taiwan. E-mail: yctseng@cs.nctu.edu.tw.*
- *J.-J. Chen is with the Department of Electrical Engineering, National University of Tainan, Tainan 70005, Taiwan. E-mail: jjchen@mail.nutn.edu.tw.*
- *Y.-C. Yang is with the Department of Computer Science, National Chiao Tung University, Hsinchu 30010, Taiwan. E-mail: yangyc@cs.nctu.edu.tw.*

Manuscript received 4 June 2009; revised 16 Jan. 2010; accepted 12 Aug. 2010; published online 27 Oct. 2010.
For information on obtaining reprints of this article, please send e-mail to: tmc@computer.org, and reference IEEECS Log Number TMC-2009-06-0206. Digital Object Identifier no. 10.1109/TMC.2010.215.

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 coming 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.

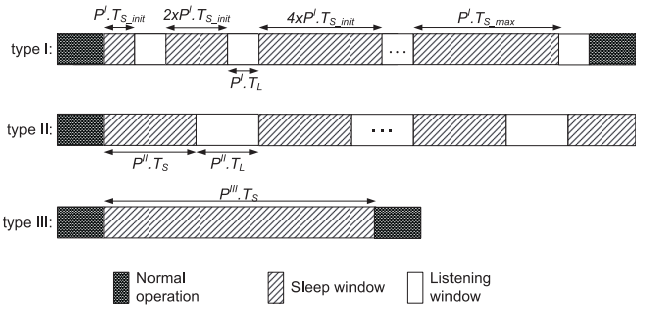


Fig. 1. Definitions of PSCs.

This paper is organized as follows: Backgrounds related to our work are presented in Section 2. Section 3 presents our fold-and-demultiplex method. A packet scheduler to cooperate with our method for delay guarantee is presented in Section 4. Section 5 gives our simulation results. Conclusions are drawn in Section 6.

2 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 $P^I.T_L$. 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

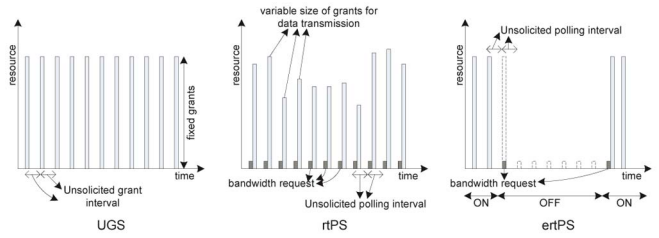


Fig. 2. Bandwidth allocation examples of UGS, rtPS, and ertPS.

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 traffics. For ERT-VR, a connection may have periodical fixed-size packets interleaved by intermittent silence, such as VoIP connections with silence suppression. Then *ertPS (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.

3 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 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:

- $C_i.D_{max}$: The delay constraint in milliseconds for real-time connection C_i.
- $C_i.MRTR$: The minimum reserved traffic rate (bits/sec).
- $C_i.I_d$: The expected packet interarrival time.
- $C_i.I_{pol\_grt}$: 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 1, 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:

- $P^I.T_L$: Size of a listening window.
- $P^I.T_{S\_init}$: Size of the initial sleep window.
- $P^I.T_{S\_max}$: Size of the maximum sleep window.

TABLE 1
Summary of Notations

Notation	Definition
$a(t)$	the bandwidth to be allocated to the MSS in the t-th frame
C_i	the ith real-time connection, $i = 1..M+N$
$C_i.D_{max}$	the delay constraint in milliseconds for real-time connection C_i
$C_i.I_d$	the expected packet inter-arrival time of real-time connection C_i
$C_i.Ipol\_grt$	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.)
$C_i.MRTR$	the minimum reserved traffic rate of real-time connection C_i
F	the length of a frame
P^I	the PSC of type I
$P^I.T_L$	the size of a listening window of a PSC P^I
$P^I.T_{S\_init}$	the size of the initial sleep window of a PSC P^I
$P^I.T_{S\_max}$	the size of the maximum sleep window of a PSC P^I
P_i^{II}	the ith PSC of type II
$P_i^{II}.T_L$	the size of a listening window of a PSC P_i^{II}
$P_i^{II}.T_S$	the size of a sleep window of a PSC P_i^{II}
p_{lcm}	the least common multiplier of T_i, $i = 1..M+N$, i.e., $p_{lcm} = lcm\{T_i \mid i = 1..M+N\}$
$S_i(t)$	the bandwidth requirement of connection C_i in frame t
$\hat{S}(t)$	the bandwidth requirement of the MSS in frame t
$\tilde{S}(t)$	the binary state of the MSS in frame t, where 1 and 0 mean active and sleeping states, respectively
T_i	the wake-up period of real-time connection C_i

For the ith PSC of type II, denoted by P_i^{II}, the following parameters will be determined:

- $P_i^{II}.T_L$: Size of a listening window.
- $P_i^{II}.T_S$: 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 PSC P_i^{II}, we call $P_i^{II}.T_L + P_i^{II}.T_S$ 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.

3.1 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.

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

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

$$P_i^{II}.T_S = \left\lfloor \frac{C_i.D_{max}}{2 \times F} \right\rfloor - P_i^{II}.T_L. \quad (2)$$

   Here F is the length of a frame. (Normally, the length of an OFDM/OFDMA frame is 5 ms [21].) As mentioned in Section 2, 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. Equation (2) ensures that C_i's wake-up period is no more than $\frac{1}{2}$ of its delay constraint $C_i.D_{max}$. Such setting guarantees bounded delay for each packet of C_i (refer to Section 4).
2. We further adjust these PSCs to increase their overlapping. Let $p_{min} = min\{P_i^{II}.T_L + P_i^{II}.T_S | \ i = 1..M+N\}$, i.e., the smallest wake-up period. We adjust each C_i's sleep window size as follows:

$$P_i^{II}.T_S = \left\lfloor \frac{P_i^{II}.T_L + P_i^{II}.T_S}{p_{min}} \right\rfloor \times p_{min} - P_i^{II}.T_L. \quad (3)$$

   This makes C_i's wake-up period an integer multiple of p_{min}. For example, if there are three PSCs with $P_1^{II}.T_L = 1$, $P_1^{II}.T_S = 3$, $P_2^{II}.T_L = 1$, $P_2^{II}.T_S = 4$, $P_3^{II}.T_L = 1$, and $P_3^{II}.T_S = 8$, then $p_{min} = 4$. 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 coprimes, the sleep behavior of the MSS repeats per 180 frames. In each cycle, there will be 84 listening frames. After the adjustment, there is one listening frame per four 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 interarrival time is four frames but its wake-up period is six 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:

$$\begin{aligned} & C_i.MRTR \\ & = [(\text{max. no. of arrivals per wake-up period}) \\ & \quad \times (\text{expected size per arrival})] \\ & \quad \times (\text{no. of wake-up periods per second}) \\ & = \left\lceil \left\lceil \frac{\left(P_i^{II}.T_L + P_i^{II}.T_S\right) \times F}{C_i.I_d} \right\rceil \right. \\ & \quad \left. \times \left(C_i.MRTR_{old} \times \frac{C_i.I_d}{1000} \right) \right\rceil \\ & \quad \times \left(\frac{1000}{(P_i^{II}.T_L + P_i^{II}.T_S) \times F} \right) \\ & = \left\lceil \frac{(P_i^{II}.T_L + P_i^{II}.T_S) \times F}{C_i.I_d} \right\rceil \\ & \quad \times \frac{C_i.MRTR_{old} \times C_i.I_d}{\left(P_i^{II}.T_L + P_i^{II}.T_S\right) \times F}. \end{aligned} \quad (4)$$

   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 5, 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:

$$C_i.I_{pol\_grt} = \left(P_i^{II}.T_L + P_i^{II}.T_S\right) \times F. \quad (5)$$

   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 $\lceil \frac{6}{4} \rceil = 2$ 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 $I_{pol\_grt} = 6F$.

3.2 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.

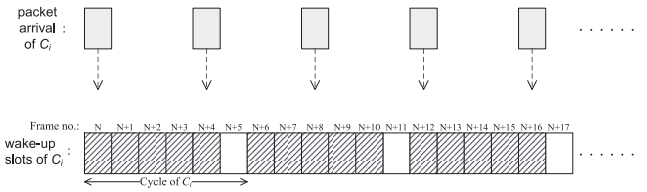


Fig. 3. A mismatch example between the packet arrival time of a flow and its wake-up period.

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 $p_{lcm} = lcm\{T_i |\ i = 1..M+N\}$ (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

$$b_i = C_i.MRTR \times \left(P_i^{II}.T_L + P_i^{II}.T_S\right) \times F. \tag{6}$$

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

$$S_i^{uert}(t) = \begin{cases} b_i, & \text{if } (t \bmod T_i) = 0, \\ 0, & \text{otherwise.} \end{cases}$$

For C_i of type RT-VR, in each listening window, it requires a fixed bandwidth ϱ 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

$$S_i^{rt-vr}(t) = \begin{cases} \varrho, & \text{if } (t \bmod T_i) = 0, \\ b_i, & \text{if } (t \bmod T_i) = 1, \\ 0, & \text{otherwise.} \end{cases}$$

In the following, whenever the context is needed, we may use $S_i^{uert}(t)$ and $S_i^{rt-vr}(t)$ 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:

$$\hat{S}(t) = \sum_{t \geq 0} S_i(t).$$

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 $\hat{S}(t)$ for simplicity. Since each $S_i(t), i = 1..M+N$, has a period of T_i, it also has a period of p_{lcm}. It follows that the summation of them also has a period of p_{lcm}.

Lemma 3.1. *$\hat{S}(t)$ is a periodical series with period p_{lcm}.*

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

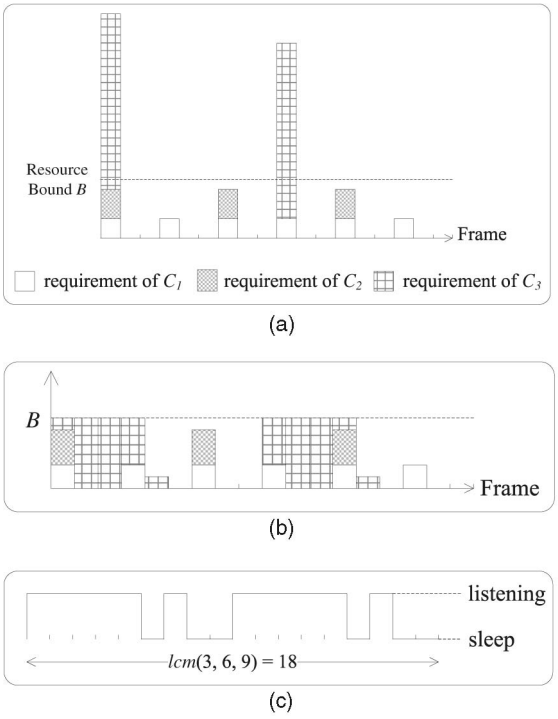


Fig. 4. Example of the state series construction. (a) Resource requirement series $\hat{S}(t)$. (b) Resource allocation series $a(t)$. (c) State series $\tilde{S}(t)$.

$$\tilde{S}(t) = \begin{cases} 1, & \text{if } a(t) > 0, \\ 0, & \text{otherwise,} \end{cases} \tag{7}$$

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

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

Lemma 3.2. *$\tilde{S}(t)$ is a periodical binary series with period p_{lcm}.*

Proof. $\tilde{S}(t)$ is a binary series which alternates between continuous 1s and continuous 0s infinitely. Consider each group of continuous 1s in $\tilde{S}(t)$; let it start from position t_{st} and end at position t_{end}. Let us define C as the set of connections, each of which has a nonzero bandwidth requirement during the interval $[t_{st} : t_{end}]$, i.e., $C = \{C_i |\ \sum_{j=t_{st}}^{t_{end}} S_i(j) > 0\}$. Intuitively, each connection in C contributes to the wake-up behavior (continuous 1s) during the interval $[t_{st} : t_{end}]$. Since each $S_i(j)$ corresponding to $C_i \in S_c$ is periodical, the same bandwidth requirements from C occurring during the interval $[t_{st} : t_{end}]$ must occur again in interval $[t_{st} + i \times p_{lcm} : t_{end} + i \times p_{lcm}]$, where $i \in Z$. Therefore, $\tilde{S}(t)$ must contain

all 1s in interval $[t_{st} + i \times p_{lcm}, t_{end} + i \times p_{lcm}]$ for each $i \in Z$. It follows that the theorem is true. □

3.3 Demultiplexing the State Series into PSCs

Although $\tilde{S}(t)$ is periodical (Lemma 3.2), if we look at any subsequence of length p_{lcm} in $\tilde{S}(t)$, 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 $\tilde{S}(t)$ into multiple state series, each meeting the definition of PSC. By Lemma 3.2, a straightforward approach is to pick the first p_{lcm} bits of $\tilde{S}(t)$ 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_{lcm} bits of $\tilde{S}(t)$, we propose a scheme based on folding $\tilde{S}(t)$ to put duplicate continuous 1s together. This may result in less PSCs.

1. Denote the first p_{lcm} bits of $\tilde{S}(t)$ by $\tilde{S}[0 : p_{lcm} - 1]$. Our goal is to construct a set C of PSCs. Initially, let $C = \emptyset$.
2. For $i = 1$ to $\frac{p_{lcm}}{p_{min}}$ do
   a. If p_{lcm} is not divisible by $i \times p_{min}$, skip this i and go back to step 2. Otherwise, cut $\tilde{S}[0 : p_{lcm} - 1]$ into $\frac{p_{lcm}}{i \times p_{min}}$ segments, each of length $i \times p_{min}$ and then fold them together by the bitwise-AND operator into a $(i \times p_{min})$-bit string $T[0 : i \times p_{min} - 1]$, i.e.,
   $$T[j] = \tilde{S}(j) \wedge \tilde{S}(j + i \times p_{min}) \wedge \tilde{S}(j + 2i \times p_{min}) \wedge \ldots,$$
   for $j = 0..i \times p_{min} - 1$.
   b. Scan string $T[0 : i \times p_{min} - 1]$ from left to right and consider each group of 1s in the string. Suppose that there is a group of 1s starting from the xth bit to the yth bit. Let $T'[0 : i \times p_{min} - 1]$ be a binary string which has all 1s from the xth bit to the yth bit and all 0s in the other places. We try to form a PSC from $T'[0 : i \times p_{min} - 1]$ as follows:
      i. Check if $T'[0 : i \times p_{min} - 1]$ is redundant by calling the procedure $Check\_Redundancy(C, T'[0 : i \times p_{min} - 1])$.
      ii. If the response is negative (i.e., not redundant), then add the string $T'[0 : i \times p_{min} - 1]$ to C.
3. For each string str in C, we generate a PSC of type II. Let str have 1s from the xth bit to the yth 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 $W[0 : p_{lcm} - 1]$ by "bitwise-OR" the strings of all PSCs. Similarly, string $Tested\_Str$ is converted to a string $W'[0 : p_{lcm} - 1]$. Then we compare $W[\,]$ and $W'[\,]$ to see if $Tested\_Str$ is redundant.

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

1. Let $W[0 : p_{lcm} - 1]$ and $W'[0 : p_{lcm} - 1]$ be two binary arrays. For $j = 0$ to $p_{lcm} - 1$, we define
   $$W[j] = \vee_{Str \in C} Str[j \bmod |Str|],$$
   $$W'[j] = \begin{cases} 1, & \text{if } Tested\_Str[j \bmod |Tested\_Str|] = 1, \\ 0, & \text{otherwise.} \end{cases}$$
2. If $W'[j] = 1$ implies $W[j] = 1$ for all $j = 0..p_{lcm} - 1$, then a "positive" response is returned; otherwise, a "negative" response is returned.

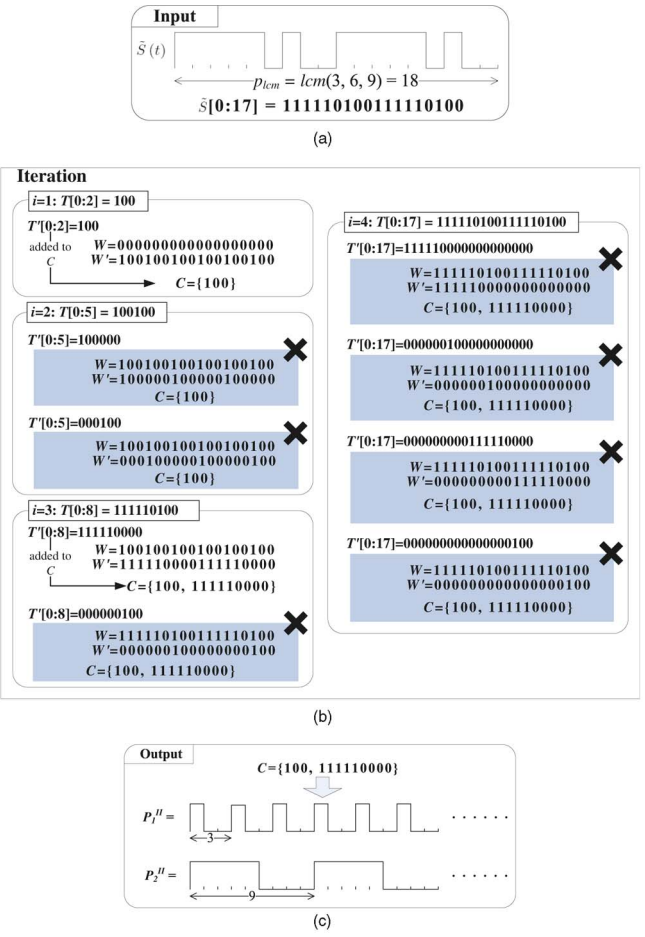


Fig. 5. Example of the PSC demultiplexing procedure.

2. If $W'[j] = 1$ implies $W[j] = 1$ for all $j = 0..p_{lcm} - 1$, 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. For example, Fig. 5a shows a state series $\tilde{S}(t)$ with $p_{lcm} = 18$ and $p_{min} = 3$. In Fig. 5b, $\tilde{S}(t)$ 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, $\tilde{S}(t)$ 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, $\tilde{S}(t)$ is cut into segments, each of 9 bits. A binary string $T[0 : 8] = 111110100$ 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

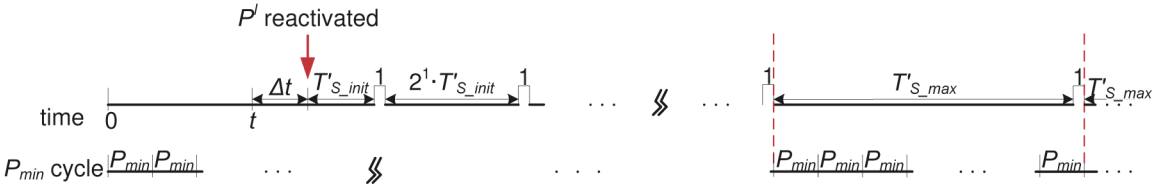


Fig. 6. Inserting a waiting interval Δt such that, after P^I reaching T'_{S_max}, the listening windows of P^I will align with the listening windows of PSCs in C.

$T_L = 5$ and $T_S = 4$ is added to C. In the last iteration, the string $\tilde{S}[0 : p_{lcm} - 1]$ does not add any new PSC. The two final PSCs are shown in Fig. 5c.

3.4 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 reactivated 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 Δt 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

$$T'_{S\_init} = T_{S\_init}, \tag{8}$$

$$T'_L = 1, \tag{9}$$

$$T'_{S\_max} = \left\lfloor \frac{T_{S\_max}}{p_{min}} \right\rfloor \times p_{min} - T'_L. \tag{10}$$

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 one frame, sleep for $2 \times T'_{S\_init}$ frames, wake up for one frame, sleep for $4 \times T'_{S\_init}$ frames, $\cdots$, 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

$$\begin{aligned} t_{flw} = t + P^I.T'_{S\_init} + 1 + 2 \times P^I.T'_{S\_init} \\ + 1 + \cdots + P^I.T'_{S\_max} + 1. \end{aligned} \tag{11}$$

So Δt 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 (11) to identify a new $\Delta t'$ with the same alignment property.

4 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-demultipex method.

Recall that our method will create two tentative series $\hat{S}(t)$ and $\tilde{S}(t)$ to represent the MSS's bandwidth requirement and listening windows. We will prove our result through these two series. Consider the $\hat{S}(t)$ 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 $S_i(t)$. Let 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 $\hat{S}(t_b)$ may exceed the capacity B, so it may take several frames to serve $S_i(t_b)$. Let Δt 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 (Δt) actually depends on

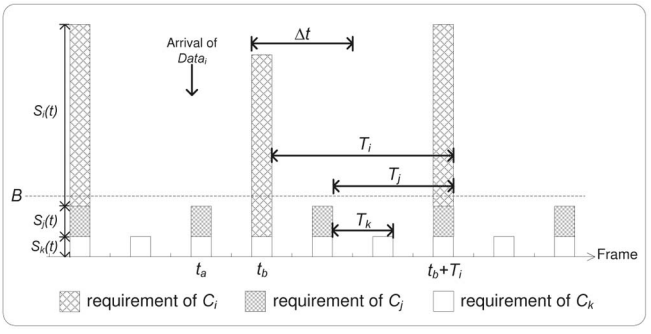


Fig. 7. Illustration of packet delay analysis.

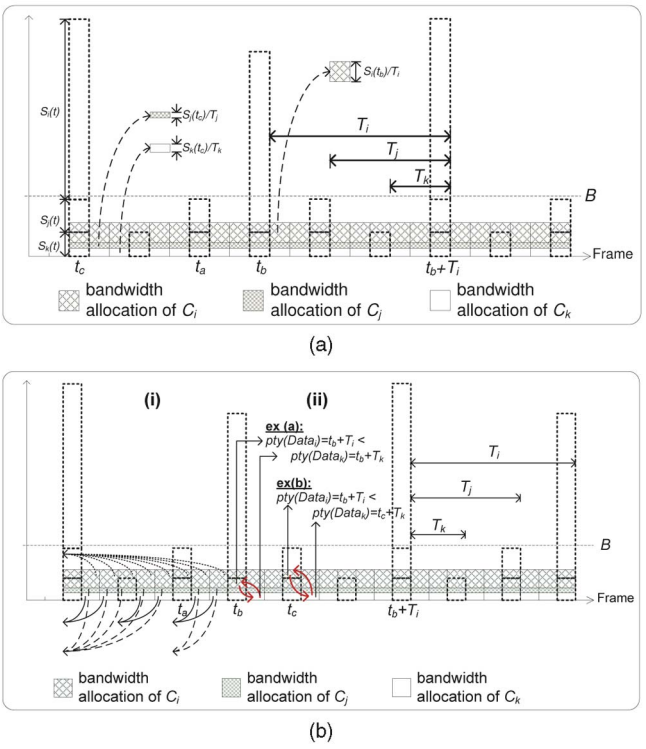


Fig. 8. Proof of Theorem 4.1. (a) The imaginary scheduling and (b) two changes to the imaginary scheduling by our earliest-next-bandwidth-first scheduling.

the scheduler. Below, we will propose a scheduler that ensures $((t_b - t_a) + \Delta t)F \leq C_i.D_{max}$.

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 (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 also 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. 8a shows how it works to serve each C_i's data by enforcing each frame to share the same amount of data $\frac{S_i(t_b)}{T_i}$ 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: 1) 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 2) 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 requested delay bound because the latter is already bounded by a tighter delay bound (this is what "earliest-next-bandwidth" means). Fig. 8b 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_is. □

TABLE 2
Six Types of Real-Time Flows Used in the Simulation

Flow type	Service type	Direction	Delay constraint (ms)	Packet inter-arrival time (ms)	Packet size (byte)
I	ERT-VR	DOWN	50	20	160
II	ERT-VR	UP	50	20	160
III	UGS	DOWN/UP	60	20	160
IV	UGS	DOWN/UP	300	80	500
V	RT-VR	DOWN/UP	300	35	440
VI	RT-VR	DOWN/UP	800	200	2000

5 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 2 (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

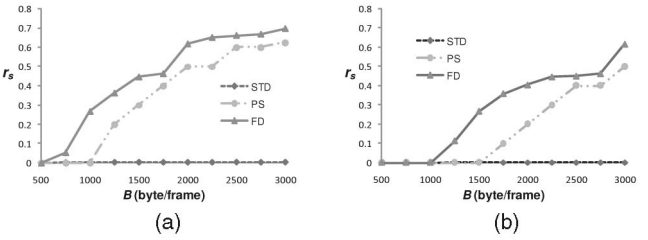


Fig. 9. sleep ratio r_s versus B with (a) two real-time flow sets and (b) three real-time flow sets.

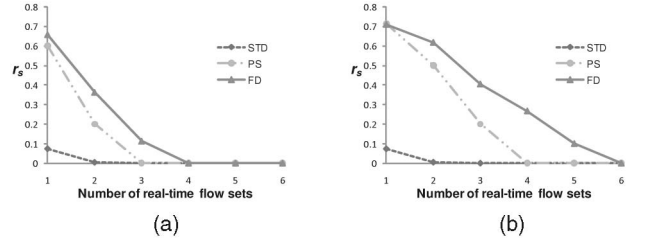


Fig. 10. sleep ratio r_s versus number of real-time flow sets with (a) $B = 1{,}250$ byte/frame and (b) $B = 2{,}000$ byte/frame.

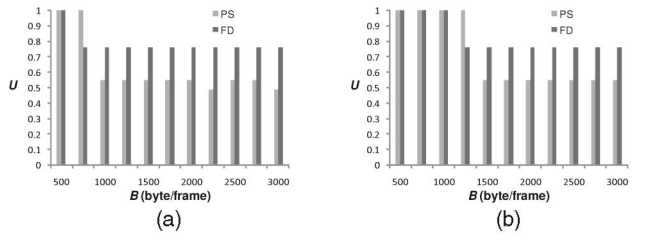


Fig. 11. resource utilization U versus B with (a) two real-time flow sets and (b) three real-time flow sets.

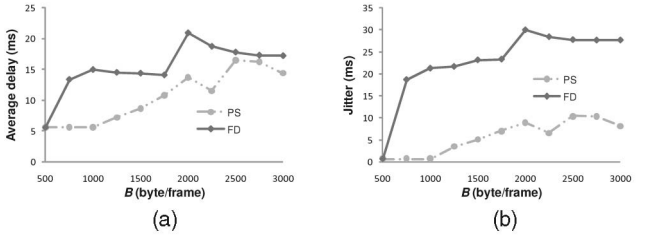


Fig. 12. (a) Average delay versus B and (b) jitter versus B (two real-time flow sets).

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 response time are considered. Below, we define one *real-time flow set* as six flows containing one from each of types I-VI.

5.1 Impact of B and Traffic Load

Fig. 9a and Fig. 9b 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 $B = 1{,}250$ (respectively, $B = 2{,}000$) byte/frame, Fig. 10a (respectively, Fig. 10b) 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 (respectively, 5) sets when $B = 1{,}250$ (respectively, $B = 2{,}000$). On the contrary, with 3 (respectively, 4) sets, PS will keep the MSS always awake when $B = 1{,}250$ (respectively, $B = 2{,}000$). This is because PS overestimates the potential traffic in some frames. Such an effect is even more serious when there exist larger differences among flows' delay constraints and interarrival times.

Fig. 11a and Fig. 11b 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 overestimation on resource demands, except at the point $B = 750$ in Fig. 11a and at the point $B = 1{,}250$ in Fig. 11b. Note that when $B \leq 750$ and $B \leq 1{,}250$ in Fig. 11a and Fig. 11b, 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 percent (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. 12a and Fig. 12b 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. 12a and Fig. 12b, we can see a jump as B is increased to 2,000 bytes/frame. This is because the number of sleep frames of the MSS are suddenly increased as B reaches 2,000 bytes/frame (which can be observed in Fig. 9a) such that the arriving packets during the sleep frames of the MSS have to wait for more frames until the MSS is awake. After $B \geq 2{,}000$, 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.

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].

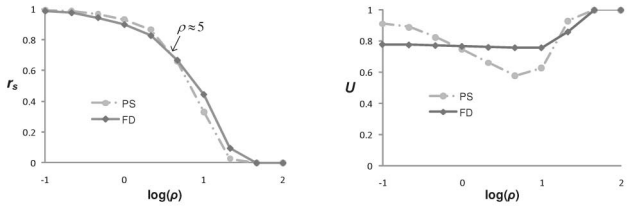


Fig. 13. Effect of ρ on sleep ratio r_s and resource utilization U under random flow arrival ($B = 1{,}250$ byte/frame).

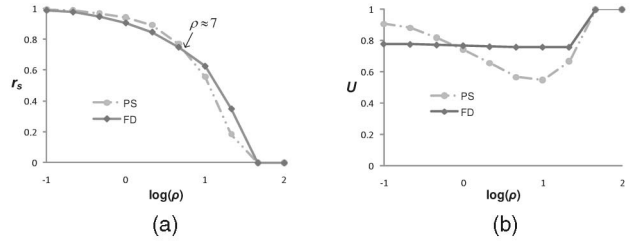


Fig. 14. Effect of ρ on sleep ratio r_s and resource utilization U under random flow arrival ($B = 2{,}000$ byte/frame).

5.2 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 2 again, but different from the previous experiment, the flows of each type arrive at a Poisson process with rate λ_r (number of flows/sec) and the hold time of each flow is exponentially distributed with mean $1/\mu$ (secs). Letting $\rho = \frac{6\lambda_r}{\mu}$, where "6" means the above six types of real-time flows under 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 $B = 1{,}250$, Fig. 13a shows the impact of ρ on sleep ratio r_s. When ρ is small, PS performs slightly better than FD. After $\rho \geq 5$, 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 one or very few flows between the MSS and the BS. Therefore, only a single PSC is enough for the sleep operation. 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. 13b plots resource utilization U against ρ. When $\rho < 1$, PS performs better than FD; after $\rho > 1$, FD becomes better; but after $\rho \geq 20$, PS outperforms FD again. Under $\rho < 1$, 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 ($\rho \geq 20$), PS outperforms FD again because PS would disable the sleep mode more often than FD does (here we assume that the resource utilization is one when the sleep mode is disabled because resource can be accurately allocated). Fig. 14 sets $B = 2{,}000$ and shows similar results to Fig. 13. In Fig. 14a, we can see that a larger B makes FD outperform PS after $\rho \geq 7$ which is larger than the value of five in the case of $B = 1{,}250$ (Fig. 13a). This is because a larger B has a higher tolerance on PS's over-estimation on resource demand.

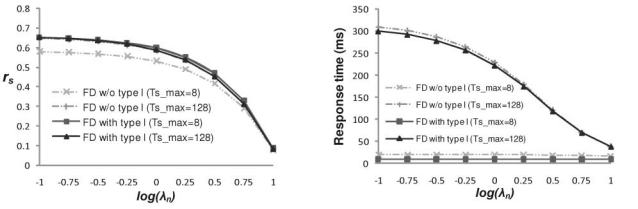


Fig. 15. Effect of packet arrival rate λ_n on (a) sleep ratio r_s and (b) response time with one real-time flow set and a non-real-time downlink flow of rate λ_n ($B = 1{,}250$ byte/frame).

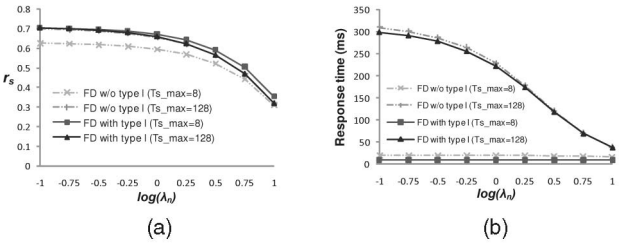


Fig. 16. Effect of packet arrival rate λ_n on (a) sleep ratio r_s and (b) response time with one real-time flow set and a non-real-time downlink flow of rate λ_n ($B = 2{,}000$ byte/frame).

5.3 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 3.4 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 $T_{S\_init}, T_L = 1$, 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 $B = 2{,}000$ and shows similar results to Fig. 15.

6 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 interarrival 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 reexecuting the whole scheme again.

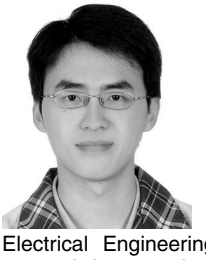
ACKNOWLEDGMENTS

Y.-C. Tseng's research was cosponsored by the MoE ATU Plan; NSC grants 97-3114-E-009-001, 97-2221-E-009-142-MY3, 98-2219-E-009-019, 98-2219-E-009-005, and 99-2218-E-009-005; ITRI, Taiwan; III, Taiwan; D-Link; and Intel.

REFERENCES

[1] *IEEE 802.16-2009, IEEE Standard for Local and Metropolitan Area Networks Part 16: Air Interface for Broadband Wireless Access Systems,* IEEE, May 2009.
[2] *IEEE 802.16m/D4, Part 16: Air Interface for Broadband Wireless Access Systems: Advanced Air Interface,* IEEE, Feb. 2010.
[3] S. Jin, M. Choi, and S. Choi, "Performance Analysis of IEEE 802.16m Sleep Mode for Heterogeneous Traffic," *IEEE Comm. Letters,* vol. 14, no. 5, pp. 405-407, May 2010.
[4] R. Kalle, M. Raj, and D. Das, "A Novel Architecture for IEEE 802.16m Subscriber Station for Joint Power Saving Class Management," *Proc. IEEE Int'l Conf. Comm. Systems and Networks (COMSNETS '09),* pp. 286-295, Jan. 2009.
[5] Y. Xiao, "Energy Saving Mechanism in the IEEE 802.16e Wireless MAN," *IEEE Comm. Letters,* vol. 9, no. 7, pp. 595-597, July 2005.
[6] Y. Zhang and M. Fujise, "Energy Management in the IEEE 802.16e MAC," *IEEE Comm. Letters,* vol. 10, no. 4, pp. 311-313, Apr. 2006.
[7] L. Kong and H.-K. Tsang, "Performance Study of Power Saving Classes of Type I and II in IEEE 802.16e," *Proc. IEEE Conf. Local Computer Networks (LCN '06),* pp. 20-27, Nov. 2006.
[8] Y. Zhang, "Performance Modeling of Energy Management Mechanism in IEEE 802.16e Mobile WiMAX," *Proc. IEEE Wireless Comm. and Networking Conf. (WCNC '07),* pp. 3205-3209, Mar. 2007.
[9] K. Han and S. Choi, "Performance Analysis of Sleep Mode Operation in IEEE 802.16e Mobile Broadband Wireless Access Systems," *Proc. IEEE Vehicular Technology Conf. (VTC '06),* vol. 3, pp. 1141-1145, May 2006.
[10] J. Shi, G. Fang, Y. Sun, J. Zhou, Z. Li, and E. Dutkiewicz, "Improving Mobile Station Energy Efficiency in IEEE 802.16e WMAN by Burst Scheduling," *Proc. IEEE Global Telecomm. Conf.,* pp. 1-5, Nov. 2006.
[11] F. Xu, W. Zhong, and Z. Zhou, "A Novel Adaptive Energy Saving Mode in IEEE 802.16e System," *Proc. IEEE Military Comm. Conf. (MILCOM '06),* pp. 1-6, Oct. 2006.
[12] J. Xiao, S. Zou, B. Ren, and S. Cheng, "An Enhanced Energy Saving Mechanism in IEEE 802.16e," *Proc. IEEE Global Telecomm. Conf.,* pp. 1-5, Nov. 2006.
[13] S. Cho and Y. Kim, "Improving Power Savings by Using Adaptive Initial-Sleep Window in IEEE802.16e," *Proc. IEEE Vehicular Technology Conf. (VTC '07),* pp. 1321-1325, Apr. 2007.
[14] J.-R. Lee and D.-H. Cho, "Performance Evaluation of Energy-Saving Mechanism Based on Probabilistic Sleep Interval Decision Algorithm in IEEE 802.16e," *IEEE Trans. Vehicular Technology,* vol. 56, no. 4, pp. 1773-1780, July 2007.
[15] T.-C. Chen, Y.-Y. Chen, and J.-C. Chen, "An Efficient Energy Saving Mechanism for IEEE 802.16e Wireless MANs," *IEEE Trans. Wireless Comm.,* vol. 7, no. 10, pp. 3708-3712, Oct. 2008.
[16] T.-C. Chen, J.-C. Chen, and Y.-Y. Chen, "Maximizing Unavailability Interval for Energy Saving in IEEE 802.16e Wireless MANs," *IEEE Trans. Mobile Computing,* vol. 8, no. 4, pp. 475-487, Apr. 2009.
[17] Y.-L. Chen and S.-L. Tsao, "Energy-Efficient Sleep-Mode Operations for Broadband Wireless Access Systems," *Proc. IEEE Vehicular Technology Conf. (VTC '06),* pp. 1-5, Sept. 2006.
[18] S.-L. Tsao and Y.-L. Chen, "Energy-Efficient Packet Scheduling Algorithms for Real-Time Communications in a Mobile WiMAX System," *Computer Comm.,* vol. 31, no. 10, pp. 2350-2359, June 2008.
[19] S.-C. Huang, R.-H. Jan, and C. Chen, "Energy Efficient Scheduling with QoS Guarantee for IEEE 802.16e Broadband Wireless Access Networks," *Proc. Int'l Conf. Wireless Comm. and Mobile Computing (IWCMC '07),* pp. 547-552, Aug. 2007.
[20] T.-C. Chen and J.-C. Chen, "Extended Maximizing Unavailability Interval (eMUI): Maximizing Energy Saving in IEEE 802.16e for Mixing Type I and Type II PSCs," *IEEE Comm. Letters,* vol. 13, no. 2, pp. 151-153, Feb. 2009.
[21] H.-S. Kim and S. Yang, "Tiny MAP: An Efficient MAP in IEEE 802.16/WiMAX Broadband Wireless Access Systems," *Computer Comm.,* vol. 30, no. 9, pp. 2122-2128, June 2007.
[22] A. Markopoulou, F. Tobagi, and M. Karam, "Assessment of VoIP Quality over Internet Backbones," *Proc. IEEE INFOCOM,* vol. 1, pp. 150-159, June 2002.
[23] Y.-C. Tseng, J.-J. Chen, and Y.-L. Cheng, "Design and Implementation of a SIP-Based Mobile and Vehicular Wireless Network with Push Mechanism," *IEEE Trans. Vehicular Technology,* vol. 56, no. 6, pp. 3408-3420, Nov. 2007.
[24] S.-F. Huang, E.H.-K. Wu, and P.-C. Chang, "Adaptive Voice Smoothing with Optimal Playback Delay Based on the ITU-T E-Model," *Embedded and Ubiquitous Computing,* pp. 805-815, Springer, 2005.

Yu-Chee Tseng received the PhD degree in computer and information science from the Ohio State University in January 1994. He is a professor (2000-present), chairman (2005-2009), and associate dean (2007-present) in the Department of Computer Science, National Chiao Tung University, Taiwan. He is also the adjunct chair professor at the Chung Yuan Christian University (2006-present). He received the Outstanding Research Award three times from the National Science Council of China (2001, 2003, and 2009), the Best Paper Award from the International Conference on Parallel Processing in 2003, the Elite I.T. Award in 2004, and the Distinguished Alumnus Award from Ohio State University in 2005. His research interests include mobile computing, wireless communication, and parallel and distributed computing. He serves on the editorial boards of *Telecommunication Systems* (2005-present), the *IEEE Transactions on Vehicular Technology* (2005-2009), the *IEEE Transactions on Mobile Computing* (2006-present), and the *IEEE Transactions on Parallel and Distributed Systems* (2008-present). He is a senior member of the IEEE.



Jen-Jee Chen received the BS and MS degrees in computer science and information engineering in 2001 and 2003, respectively, from the National Chiao Tung University, Hsinchu, Taiwan, and the PhD 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 an assistant professor at the Department of Electrical Engineering, National University of Tainan, Taiwan. His research interests include wireless communication, mobile computing, personal communication service, cross-layer design, and network performance modeling and analysis. He is a member of the IEEE.

Yen-Chih Yang received the BS and MS degrees in computer science from the National Chiao Tung University in 2006 and 2008, respectively. His research interests include wireless communications and mobile computing. He has been with MediaTek Inc., Taipei, Taiwan, as a software engineer since 2009.