Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computer Communications 27 (2004) 1236–1247
computer communications
www.elsevier.com/locate/comcom

A bandwidth-based polling scheme for QoS support in Bluetooth

Chun-Chuan Yang*, Chin-Fu Liu

Multimedia and Communications Laboratory, Department of Computer Science and Information Engineering, National Chi Nan University, 1, University Rd, Puli Nantou County 545, Taiwan, ROC

Received 18 March 2003; revised 29 March 2004; accepted 13 April 2004

Available online 8 May 2004

Abstract

A flexible bandwidth management scheme namely Bandwidth-based Polling (BBP) for Bluetooth is proposed. A framing structure of time is defined in BBP, and the master allocates proper number of slots for each active slave in a frame. BBP allows the master to poll a slave more than once to achieve high flexibility for bandwidth allocation. The calculation of the polling time as well as the payload type for a slave is according to the bandwidth requirement of the slave and the limit of the frame size controlled by the master. Extension of BBP for supporting slaves with the SCO link and slaves without bandwidth requirement (best-effort slaves) is also proposed. Simulation results have shown the efficiency and fairness of BBP in bandwidth management. The flexibility of bandwidth allocation depends on the maximum polling time preset by BBP. A larger maximum polling time makes the higher flexibility of bandwidth allocation at the expense of a longer latency before reaching the equilibrium state. Analysis of the impact of the maximum polling time on bandwidth allocation is also presented in the paper.

Keywords: Bluetooth; QoS; Bandwidth management; Polling scheme

1. Introduction

Bluetooth [1–5] is an emerging technology of ad hoc networking that provides low power, low cost and low complexity communications for electronic devices in a small area. Bluetooth devices sharing a wireless channel form a piconet. In a Bluetooth piconet, time slot is defined for accessing the channel and the medium access scheme is based on polling algorithm controlled by the master in a Time Division Duplex fashion. More specifically, the master sends packets to slaves in even-numbered slots triggering a transmission from slaves in subsequent slot. Slaves are allowed to send packets only in response to a master packet. Most of the previous work of research in Bluetooth focused on performance improvement in terms of channel utilization. Different polling and scheduling schemes as well as SAR (Segmentation and Reassembly) policies for improving utilization in Bluetooth had been proposed [6–12]. QoS support for Bluetooth [13,14] has attracted less attention in the literature. There are some research results for QoS support by polling strategy in the wireless environment [15–19], but none of them can be applied to Bluetooth because of the special MAC protocol and different payload types of Bluetooth. In this paper, we focus on QoS support for Bluetooth.

Bandwidth allocation is direct but important in supporting QoS to some extent for Bluetooth. A good bandwidth management scheme should meet various bandwidth requirements of slaves and maintain fairness when the channel is saturated (over-requested). As the bandwidth is over-requested, fairness means that (1) for those slaves with a bandwidth requirement lower than the equal share of channel capacity, they can get the amount of bandwidth they have requested, and (2) the rest of the channel capacity is equally allocated to the other slaves (i.e. slaves with a bandwidth requirement higher than the equal share).

Thus, goals of bandwidth management should include (1) *bandwidth satisfaction* and (2) *fairness*. A pure Round Robin polling scheme with three payload types (1, 3, 5 slots) is not enough to meet the first goal mentioned above, since only three levels of bandwidth are provided. In this paper, a flexible bandwidth management scheme namely Bandwidth-based Polling (BBP) is proposed for bandwidth allocation for slaves. A slave with bandwidth requirement is called a *QoS-slave* in the paper. A slave requesting a *SCO* connection (*Synchronous Connection-Oriented link*) is

* Corresponding author. Tel.: +886-49-291-0960; fax: +886-49-291-5226.
E-mail address: ccyang@csie.ncnu.edu.tw (C.-C. Yang).

0140-3664/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2004.04.007

called a *SCO-slave*. Slaves that are neither QoS-slaves nor SCO-slaves are called best effort slaves (denoted by BE-slave). We assume that a slave can only be one of the three types of slaves at the same time. Two versions of BBP are proposed in this paper. The basic version of BBP namely *BBP-bas* is for QoS-slaves only. The extension of basic BBP namely *BBP-ext* can support all types of slaves in a Bluetooth piconet. Moreover, with the scheduling algorithm associated with BBP-ext, an upper bound of access delay for a SCO-slave can be provided.

The rest of the paper is organized as follows. The basic version of BBP as well as the actions of QoS-slaves and the master in BBP-bas are presented in Section 2. The extension of basic BBP (BBP-ext) for supporting SCO-slaves and BE-slaves is presented in Section 3. The scheduling algorithm of BBP-ext is presented in Section 4. Performance evaluation of the proposed bandwidth management scheme is presented in Section 5. Finally Section 6 concludes this paper.

2. Basic version of bandwidth-based polling (BBP-bas)

2.1. Basic idea and slot allocation for QoS-slaves

In order to allocate proper bandwidth, a framing structure of time is defined. The master of the piconet allocates proper number of slots in a frame for each active slave. The length of the frame should not be static but dynamic for flexible bandwidth allocation. Since only three payload types (1, 3, 5 slots) can be used for slaves, if the master equally polls each active slave in a time frame, at most three levels of bandwidth can be allocated, which greatly reduce the flexibility of bandwidth allocation. Therefore, the proposed BBP scheme allows the master to poll a slave more than once in a time frame to achieve high flexibility.

Multiple polling for a slave implies that the slave can transmit data by any combination of 1-, 3-, and 5-slot payload in a frame. However, since a larger payload (e.g. DH5) has higher utilization than a smaller one (e.g. DH1), it is better for a slave to properly choose a larger payload for each poll. Types of payload used in BBP are shown in Table 1. The number of bytes (*ByteCount*) and the polling time for each payload type are also shown in the table. Note that the payload type higher than DH5 represents a combination of DH5, DH3, and DH1. For example, DH8 means DH5 + DH3 (polling twice), and DH11 means DH5 + DH5 + DH1 (polling three times). Practically, BBP should set an upper bound for the polling time, which is denoted by K in the paper. The maximum polling time K determines the maximum payload type (i.e. DH5* K) for each slave as well as the maximum frame length.

Calculation of the number of slots and the polling time in a frame for a given bandwidth requirement is explained in the following. Given that the master restricts the frame size of the piconet within a limit value namely *PicoFrameLimit* (in *slots*) and the bandwidth requirement of $QoS\text{-}slave_i$ is $BwRQ_i$ (in *bps*), the maximum number of bytes ($\#Bytes_i$) that needs to be transmitted in a frame for $QoS\text{-}slave_i$ is:

$$\#Bytes_i = (BwRQ_i * PicoFrameLimit * 625\mu s)/8.$$

Thus, the payload type for $QoS\text{-}slave_i$ in a frame is the smallest one in Table 1 whose $ByteCount >= \#Bytes_i$ or the maximum payload type DH5* K, where K is the maximum polling time predefined by BBP.

BBP adopts a progressive and distributed approach for bandwidth allocation, which may cross several frames to finish. The master and slaves in a piconet exchange information in each frame for bandwidth management. Initially, the payload type for an active slave is set as the smallest one, i.e. DH1. During the bandwidth allocation (negotiation) process, each QoS-slave tries to upgrade its payload type to have a larger share of channel capacity to fulfill its bandwidth requirement. On the other hand, the master controls the bandwidth allocation by properly changing (either enlarging or shrinking) the upper bound of frame size (*PicoFrameLimit*). Moreover, BBP adopts the soft-state bandwidth reservation, which means a QoS-slave needs to issue its bandwidth request in each frame to maintain its bandwidth share. Bandwidth requests that are granted in a time frame are served in the next frame. In other words, during a time frame, a QoS-slave is served the bandwidth it has requested in the previous frame and refreshes its bandwidth requirement for the following frame. Details of the actions at the slave and the master of BBP-bas are explained, respectively, in the following sections.

2.2. QoS-slave's action in BBP-bas

When the master polls a slave, current frame size (denoted by *PicoFrameSize*) and the upper bound of the frame size (*PicoFrameLimit*) are passed to the slave. The slave tries to upgrade its payload type from the previous payload type (initially, DH1) allocated in the last frame. New payload type (denoted by $RequestSlot_i$) is calculated according to current *PicoFrameLimit* and the bandwidth requirement of the slave ($BwRQ_i$) as mentioned above. Since upgrading the payload type will increase the frame size of the piconet (*PicoFrameSize*), upgrade of the payload

Table 1
Bytes and polling time for payload type in BBP

Payload type	DH1	DH3	DH5	DH6	DH8	DH10	DH11	DH13	DH15	···
Polling time	1	1	1	2	2	2	3	3	3	···
ByteCount	27	183	339	366	522	678	705	861	1017	···
Remarks				5 + 1	5 + 3	5 + 5	5 + 5 + 1	5 + 5 + 3	5 + 5 + 5	···

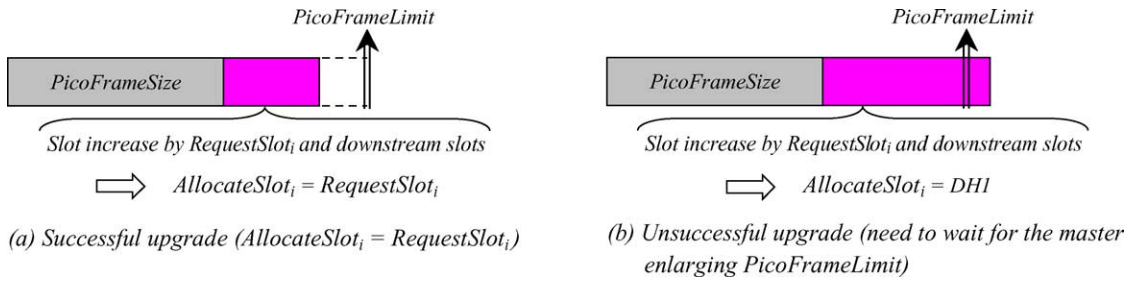


Fig. 1. Upgrading the payload type.

type is successful only when the resulted frame size is still smaller than current *PicoFrameLimit*. The idea is shown in Fig. 1. Increase of the slots for the upgrade is easily computed from the old payload type, the new payload type (*RequestSlot*$_i$) as well as the change of the downstream slots from the master to the slave.

If the upgrade is successful, number of slots allocated to the slave in a frame (denoted by *AllocateSlot*$_i$) is *RequestSlot*$_i$, if not, *AllocateSlot*$_i$ = DH1. Note that for unsuccessful upgrade, the new payload type for the slave is not set as the old one but the smallest payload type (DH1) so that the following QoS-slaves may have more chance to upgrade their payload types. Moreover, the slave computes its expected frame limit (in *slots*, denoted by *FrameLimit*$_i$) according to the calculated payload type and its bandwidth requirement. Calculation of *FrameLimit*$_i$ is similar to the reverse of calculation of the payload type described in Section 2.1:

$$FrameLimit_i = (ByteCount \text{ in } RequestSlot_i) * 8/(BwRQ_i * 625\mu s).$$

RequestSlot$_i$ and *FrameLimit*$_i$ are both passed to the master for updating *PicoFrameSize* and *PicoFrameLimit*. The overall algorithm for QoS-slaves in BBP-bas is displayed in Fig. 2.

2.3. Master's action in BBP-bas

As mentioned above, the master passes *PicoFrameSize* and *PicoFrameLimit* to a slave and collects/records *RequestSlot*$_i$ and *FrameLimit*$_i$ returned by the slave.

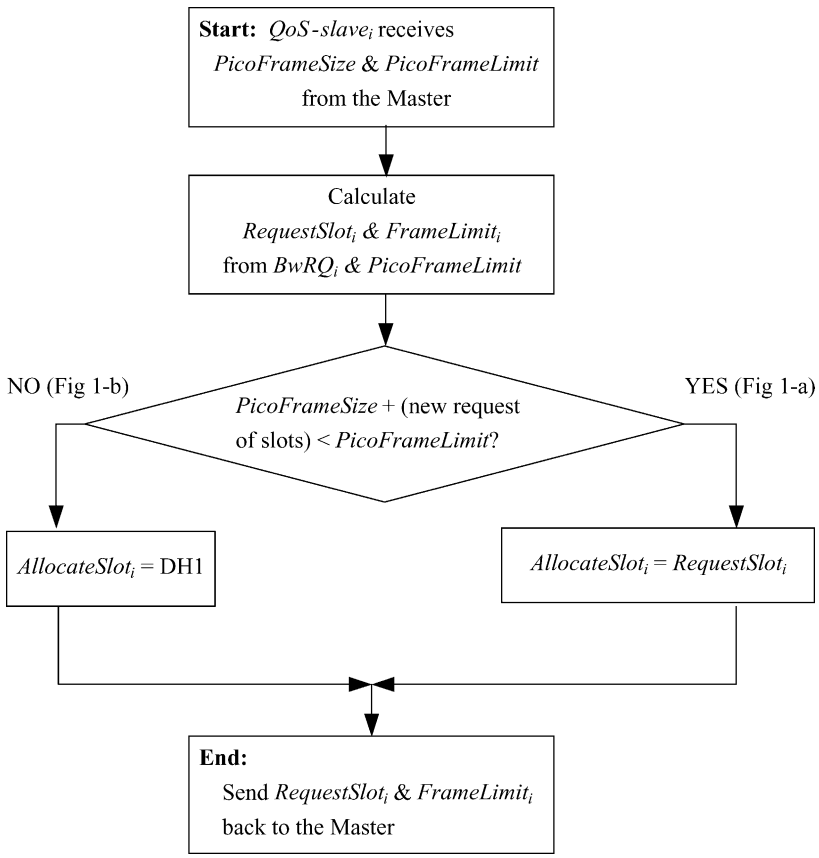
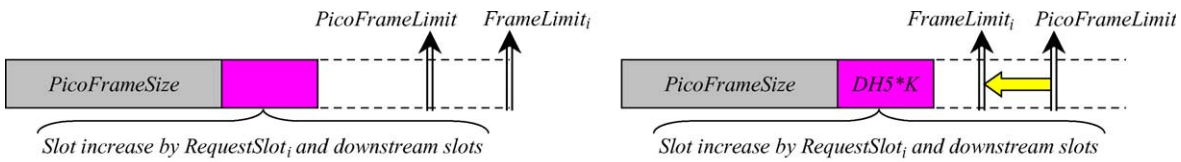


Fig. 2. The algorithm for QoS-slaves in BBP-bas.



(a) $BwRQ_i$ is satisfied; *PicoFrameLimit* unchanged (b) $BwRQ_i$ is not satisfied; Master shrinks *PicoFrameLimit*

Fig. 3. *PicoFrameLimit* vs. $FrameLimit_i$.

$RequestSlot_i$ and $FrameLimit_i$ represent the bandwidth request of the slave, and the master performs the same check as the slave (i.e. whether the total number of slots is under *PicoFrameLimit* or not) to grant the request or not. If the request is granted (in this case, the slave's $AllocateSlot_i = RequestSlot_i$), the increase of slots by the upgrade is added to *PicoFrameSize*. Moreover, for a successful upgrade, if received $FrameLimit_i <$ *PicoFrameLimit*, which means the upgrade should be granted, but it does not satisfy the bandwidth requirement of the slave. In other words, the slave has requested the maximum payload type ($AllocateSlot_i = DH5*K$) but still cannot satisfy its $BwRQ_i$ under current *PicoFrameLimit*. In such case, the master reduces *PicoFrameLimit* to $FrameLimit_i$ to allocate enough bandwidth for the slave. *PicoFrameLimit* remains unchanged if received $FrameLimit_i >$ *PicoFrameLimit* (i.e. $BwRQ_i$ is satisfied). The relationship of *PicoFrameLimit* and $FrameLimit_i$ for successful upgrade is displayed in Fig. 3.

On the other hand, if the request is rejected (in this case, the slave's $AllocateSlot_i =$ DH1), the master knows the upgrade is unsuccessful and enlarges *PicoFrameLimit* to a proper value so that the slave may have a chance to upgrade in the next frame. The master either sets the new value of *PicoFrameLimit* as the smallest $FrameLimit_i$ that is larger than the old *PicoFrameLimit* or just adds a proper number of slots to the old *PicoFrameLimit*. Again, new values of *PicoFrameSize* and *PicoFrameLimit* are passed to next slave.

If the bandwidth requirements of all QoS-slaves in the piconet have not changed, BBP-bas process reaches the equilibrium state when *PicoFrameSize* and *PicoFrameLimit* remain unchanged for two consecutive frames.

The master reduces the value of *PicoFrameLimit* and restarts the bandwidth negotiation process when an active QoS-slave becoming inactive or leaving the piconet. The algorithm for master's actions in BBP-bas is shown in Fig. 4.

3. Supporting SCO-slaves and BE-slaves in BBP (BBP-ext)

3.1. Supporting SCO-slaves

In the specification of Bluetooth [4], SCO links (SCO-slaves) are served in a periodic manner to support isochronous service. A parameter T_{sco} is defined to set the length of the period. Given that the specification has defined $T_{sco} = 6$, a maximum number of three SCO-slaves can be supported at the same time in a piconet. In order to achieve high flexibility of synchronous bandwidth allocation, the concept of *adaptive* T_{sco} [13] is adopted in BBP. Adaptive T_{sco} allows a SCO-slave to select a different value of T_{sco} from the original one ($T_{sco} = 6$) and each SCO-slave may have its own T_{sco} value. As we will explain in Section 4, the scheduling algorithm of BBP cannot guarantee isochronous access for SCO-slaves that match individual T_{sco} value. However, we will prove that the offset of the access point for SCO-slaves is bounded by a small number of slots. That is, a small upper bound of the access delay for SCO-slaves is provided in BBP-ext.

Since SCO-slaves should be served periodically regardless of the size of the frame (*PicoFrameSize*), the master has to calculate and reserve proper slots for SCO-slaves when the size of the frame is increased (due to the successful upgrade of a QoS-slave as presented in Section 2.3) in the bandwidth negotiation process of BBP-bas. As the frame size is increased, the master checks if the new frame size crosses the service time slots of a SCO-slave according to T_{sco} of the SCO-slave. If so (i.e. the SCO-slave should be served one more time for the new frame size), *PicoFrameSize* is added by two more slots (reserved for the SCO-slave) and passed to the next QoS-slave. The idea is shown in Fig. 5. If the resulted frame size is larger than *PicoFrameLimit*, the master enlarges the value of *PicoFrameLimit*, which may result in renegotiation of QoS-slaves in the following frames.

3.2. Supporting best effort slaves

Since BE-slaves have no bandwidth requirement, the payload type for a BE-slave can always be DH1 (one slot) regardless of the size of the frame, *PicoFrameSize*. Therefore, as the size of the time is getting longer, the bandwidth share of a BE-slave is getting smaller. However, if all of the QoS-slaves are satisfied and there is still some channel capacity left for BE-slaves, BBP-ext will give each BE-slave the equal share of the rest of channel capacity. The master is in charge of the allocation of the rest of capacity, since it keeps track of the information of each slave in the piconet.

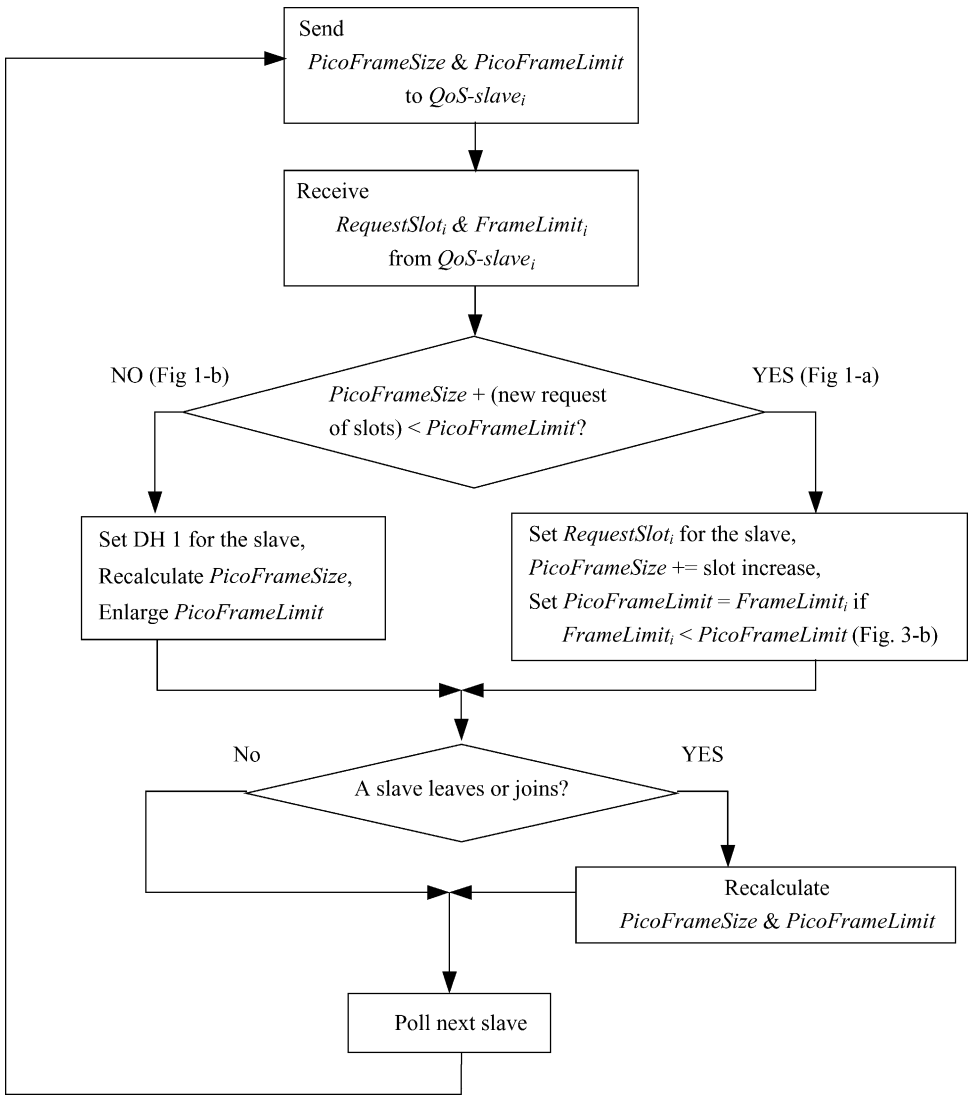


Fig. 4. The algorithm for Master in BBP-bas.

If any of the QoS-slaves is not satisfied with the bandwidth allocation in a frame, each BE-slave can only get DH1 in the frame. On the other hand, if all of the QoS-slaves are satisfied, the master calculates the rest of the channel capacity that can be allocated to BE-slaves. The difference between *PicoFrameLimit* and *PicoFrameSize* (i.e. *PicoFrameLimit* − *PicoFrameSize*) represents the capacity that can be allocated to BE-slaves. Thus, master gives each BE-slave the same share of the rest of the capacity, i.e. *(PicoFrameLimit* − *PicoFrameSize)/# BE-slaves*. Note that the resulted payload type for BE-slaves must properly fit 1-, 3-, and 5-slot boundaries.

3.3. Packet format for BBP

The original packet format (Fig. 6(a)) of Bluetooth needs to be modified to support operations of BBP. Two 8-bit fields are added in the packet header (in-band signaling) for

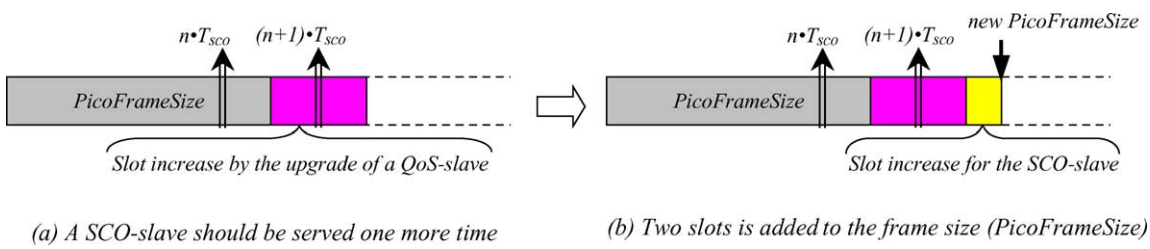


Fig. 5. Reserving slots for SCO-slaves.

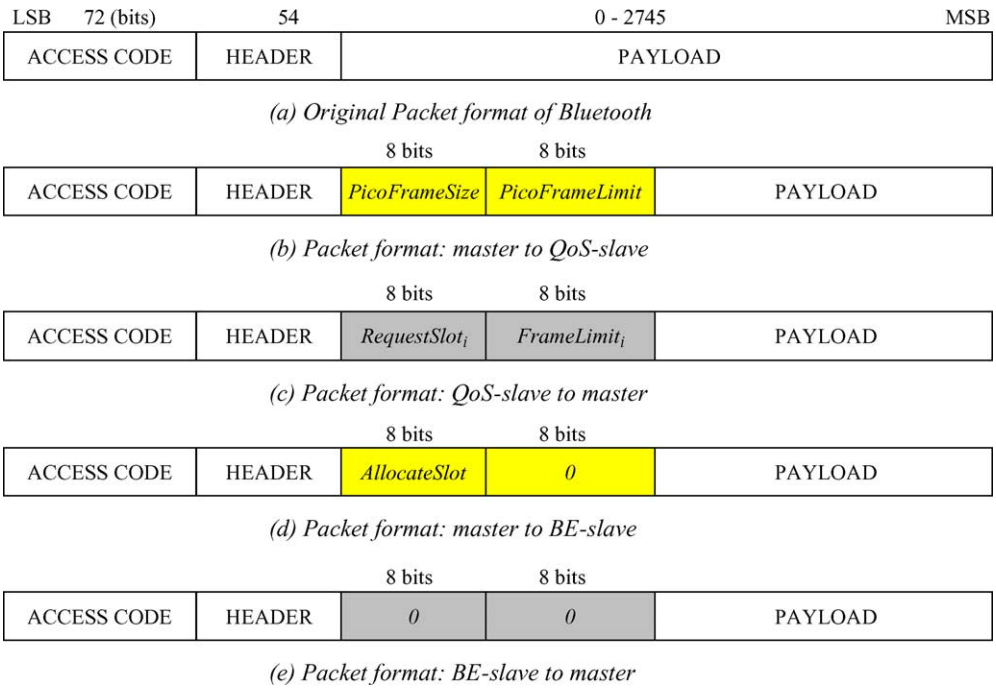


Fig. 6. Packet format for BBP.

exchanging information between the master and slaves. Interpretation of the two fields depends on the direction (uplink or downlink) of the packet as well as the type of slaves. For packets transmitted from the master to a QoS-slave, *PicoFrameSize* and *PicoFrameLimit* are filled in the two fields as shown in Fig. 6(b). A QoS-slave returns its $RequestSlot_i$ and $FrameLimit_i$ in the fields to the master as shown in Fig. 6(c). No change has been made for SCO packet format, but the SCO-slave must inform the master of its T_{sco} value during link setup phase. As shown in Fig. 6(e), the two fields of the packets from a BE-slave to the master are both set to *zero* to inform the master of the presence of the BE-slave. On the other hand, the master informs the BE-slave of its payload type by the first field as shown in Fig. 6(d).

Note that 8-bit *PicoFrameLimit* (*PicoFrameSize*) limits the frame size no more than 255 slots. The reason why it is appropriate for BBP is explained in the following. As will be shown in the simulation results (Section 5.2), the maximum polling time $K = 4$ is quite enough for flexible bandwidth allocation in a piconet. The maximum frame size for $K = 4$ and 7 active QoS-slaves is $(4*5)*2*7 = 280$ slots. In order to reduce the overhead introduced by new fields in a packet as well as make a proper boundary for each field, 8-bit is chosen.

4. Scheduling algorithm for BBP

At the end of a time frame, the master has already collected enough information for bandwidth allocation in the new frame. The information includes (1) *PicoFrameSize* and *PicoFrameLimit* of the new frame, (2) payload type for each QoS-slave, (3) T_{sco} value for each SCO-slave, and (4) payload type for each BE-slave. The scheduling algorithm in BBP decides the polling order of slaves in the new frame. The main objective of the scheduling algorithm is to reduce as much the jitter of the access delay as possible. Therefore, the scheduling algorithm tries to evenly distribute the polls for each slave in the frame. The first step of the scheduling algorithm is to put all QoS-slaves in a round-robin manner. Second, since SCO-slaves require isochronous access of the channel, they are inserted in proper slots according to T_{sco} value. If the insertion point of a SCO-slave locates in the middle of a payload type (e.g. DH5), the insertion point should be shifted to the boundary of the payload type. Lastly, polls for BE-slaves are appended to the end of the polling sequence in a round-robin manner. An example of determining the polling order is shown in Fig. 7. Although a QoS-slave is polled several times in a frame, information exchange between the master and slaves for bandwidth allocation in the next frame only happens in the first poll.

Because of the shift of the polling slot for SCO-slaves, isochronous service cannot be guaranteed in BBP. However, there is an upper bound for the shift of SCO-slaves as explained in the following. First, T_{sco} must be an even number of slots, and second, there are two reasons for a SCO-slave to be shifted: (1) the SCO-slave should be polled within the payload type of a QoS-slave, and (2) the SCO-slave is shifted by other SCO-slaves (i.e. the access time slots of two or more SCO-slaves hit the same slot). The maximum shift caused by reason (1) is 4 slots, and the maximum shift caused by reason (2) is $(N - 1)*2$, where N is the number of SCO-slaves in the piconet. Thus, the upper bound of the shift is $4 + (N - 1)*2$.

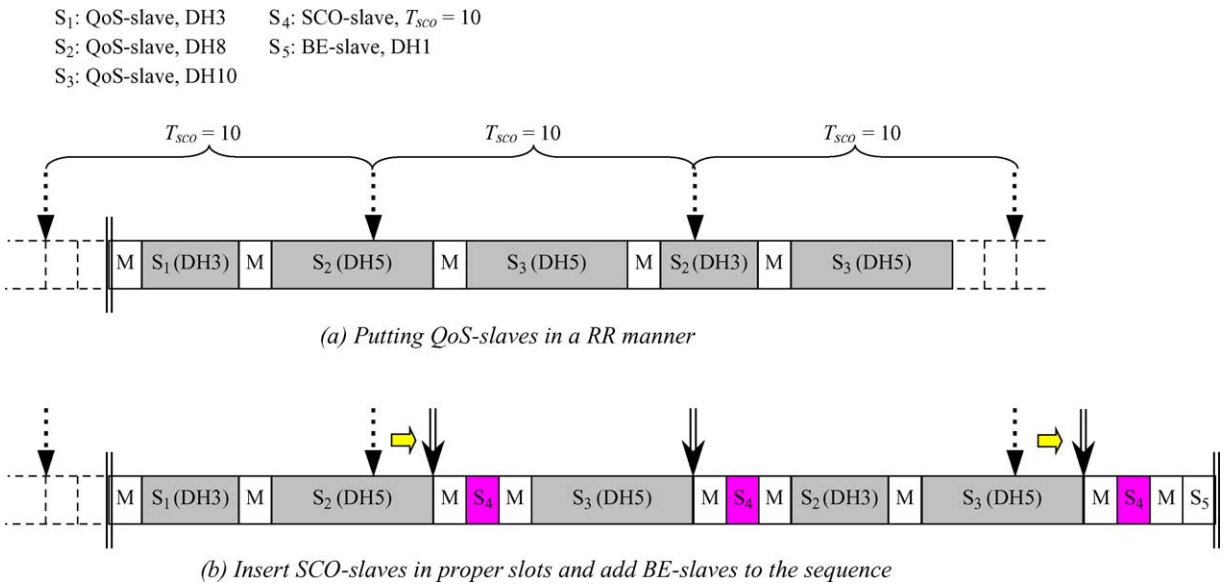


Fig. 7. E.g. determining the polling sequence.

It is worth mentioning that the upper bound of the shift for SCO-slaves provides a good way for power optimization by adopting the *sniff* mode at SCO-slaves [11]. More specifically, the SCO-slave can set the parameters of sniff mode as follows to reduce its power consumption: $T_{\text{sniff}} = T_{\text{sco}}$ and $N_{\text{sniff-attempt}} = 4 + (N - 1)*2$.

5. Performance evaluation

5.1. Analysis of the impact of K

We investigate the impact of K on bandwidth allocation by analyzing two performance criteria: (1) total number of bandwidth combinations for QoS-slaves, and (2) the longest time (in the worst case) needed to reach the equilibrium state for a given K. Total number of bandwidth combinations for a given K is denoted by $N_{\text{BW}}(K)$, and the worst-case time to reach the equilibrium state is denoted by $T_{\text{E}}(K)$. Apparently, a larger K results in a larger $N_{\text{BW}}(K)$ and a larger $T_{\text{E}}(K)$. Thus, there is a trade-off between $N_{\text{BW}}(K)$ and $T_{\text{E}}(K)$ in deciding the value of K. We assume only QoS-slaves are present in the piconet and the downstream payload from the master to each slave is always DH1 in the analysis.

In order to calculate $N_{\text{BW}}(K)$, we first consider possible combinations of payload type. We denote the number of payload combination by $N_{\text{PT}}(K)$. Considering the case of $K = 1$, there is only three choices for each master-slave pair: (DH1, DH1), (DH1, DH3), and (DH1, DH5). Thus, the number of payload combination for $K = 1$ and S QoS-slaves (S is the number of active QoS-slaves in the piconet) is $C(3 + S - 1, S)$;

we define

$$C(m, n) = \binom{m}{n} = \frac{m!}{n!(m-n)!},$$

which is actually the same as the number of ways to place S non-distinct objects into three distinct cells where a cell can hold more than one object. For a general K, there are $3K$ distinct cells (as shown in Table 2) for S non-distinct objects. Thus, $N_{\text{PT}}(K) = C(3K + S - 1, S)$.

Table 2
Pairs of payload for distinct cells

K=3 … K=2

Poll once	Twice	3 times	4 times	…
(1, 1)	(2, 6)	(3, 11)	(4, 16)	…
(1, 3)	(2, 8)	(3, 13)	(4, 18)	
(1, 5)	(2, 10)	(3, 15)	(4, 20)	

(1, 1) represents (master = DH1, slave = DH1) and
(2, 6) represents (master = DH1*2, slave = DH5+DH1)

Table 3
$N_{\text{PT}}(K)$ vs. $N_{\text{BW}}(K)$ for $S = 7$

	K					
	1	2	3	4	5	6
$N_{\text{PT}}(K)$	36	792	6435	31824	116280	346104
$N_{\text{BW}}(K)$	36	784	6426	31703	116158	345304

Table 4
Some test cases in the simulation

(Kbps)	Slave 1	Slave 2	Slave 3	Slave 4	Slave 5	Slave 6	Slave 7	Remarks
Test case 1	50	75	100	125	150	175	N/A	6 QoS
Test case 2	50	75	100	125	150	$T_{sco} = 10$	N/A	5 QoS, 1 SCO
Test case 3	32	64	96	128	160	$T_{sco} = 16$	$T_{sco} = 24$	5 QoS, 2 SCO
Test case 4	80	120	160	200	$T_{sco} = 16$	BE	N/A	4 QoS, 1 SCO, 1 BE
Test case 5	80	120	160	200	BE	BE	N/A	4 QoS, 2 BE
Test case 6	32 (0–2 s)	64 (0–2 s)	96 (0–2 s)	128 (0–2 s)	160 (0–2 s)	192 (0–1 s)	N/A	Slave 6 leaves early

Since there are cases that two different combinations of payload result in the same bandwidth allocation, $N_{BW}(K)$ is not equal to $N_{PT}(K)$. For instance, bandwidth allocation of QoS-slaves is actually the same for {(1, 5) (1, 5) (1, 5)} and {(2, 10) (2, 10) (2, 10)} for $S = 3$. We calculate the exact number of bandwidth combinations $N_{BW}(K)$ by a generator program. Values of $N_{PT}(K)$ and $N_{BW}(K)$ for $S = 7$ are listed in Table 3, which indicates that $N_{PT}(K)$ is pretty close to $N_{BW}(K)$.

The longest time $T_E(K)$ for reaching the equilibrium state is calculated as follows. The initial frame size is 2*S (i.e. 2 slots for each master-slave pair). The longest final frame size is $6*S*K$ (i.e. all slaves are served DH5* K in a frame). The master enlarges *PicoFrameLimit* if there is a QoS-slave not satisfied. The worst case happens when there is always a QoS-slave not satisfied with the current bandwidth allocation and the master enlarges *PicoFrameLimit* in each frame until the longest frame size is reached. Since the most conservative way to enlarge *PicoFrameLimit* is to add 2 slots to *PicoFrameLimit* in each frame, the total number of slots before reaching the equilibrium state in the worst case is:

$$T_E(K) = \frac{(6*S*K + 2*S)*\dfrac{6*S*K - 2*S}{2}}{2} = S^2(9K^2 - 1)$$

(in slots).

For example, for $K = 4$ and $S = 6, T_E(K) = 5148$ slots = 3.2175 s (1600 slots = 1 s).

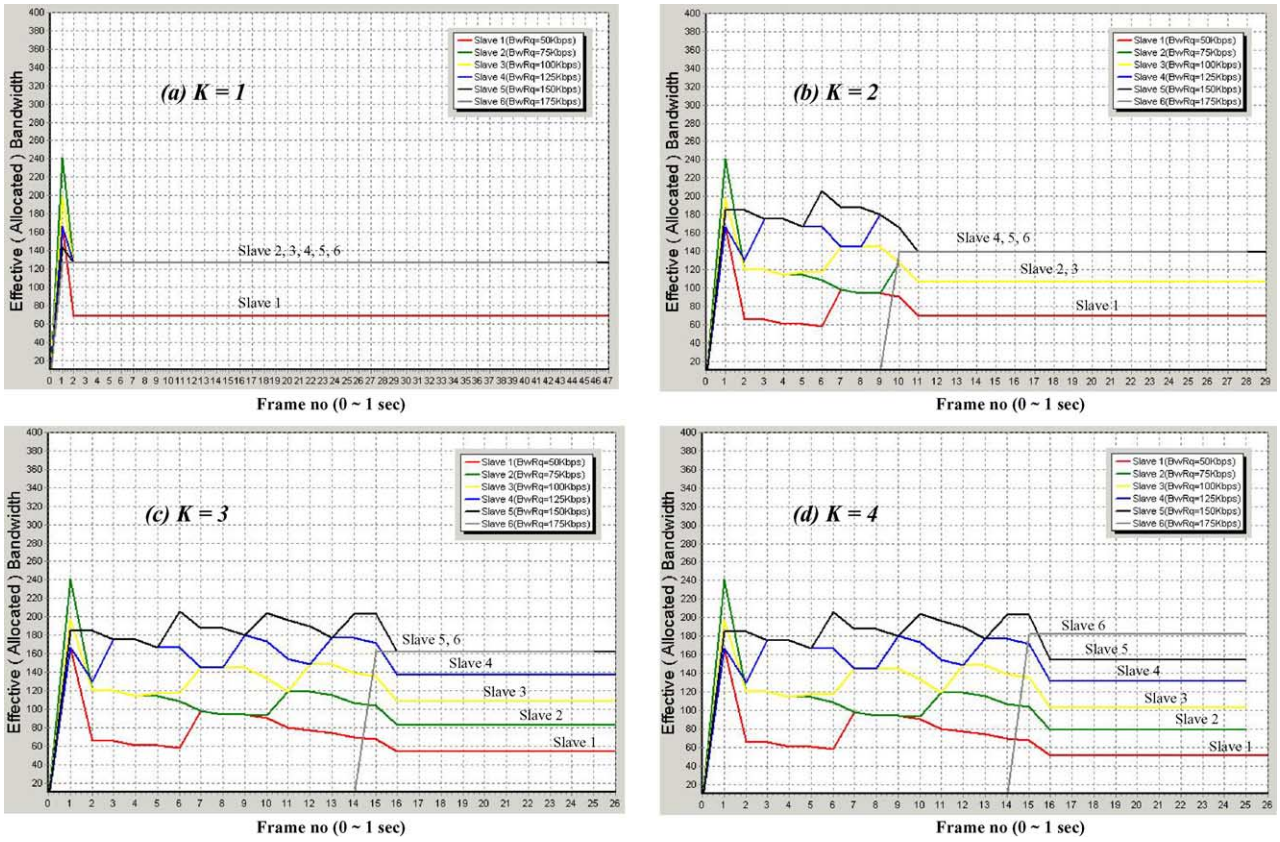
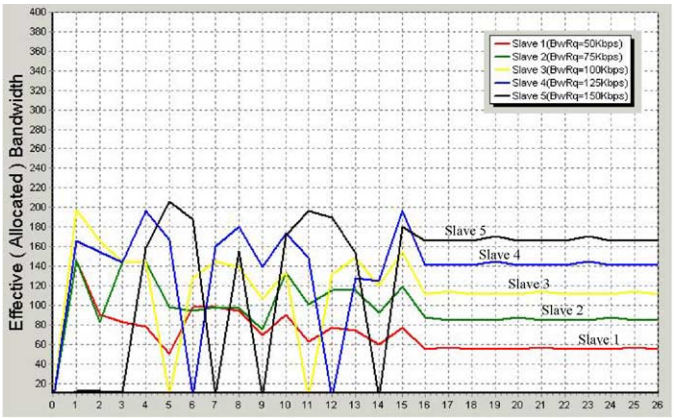
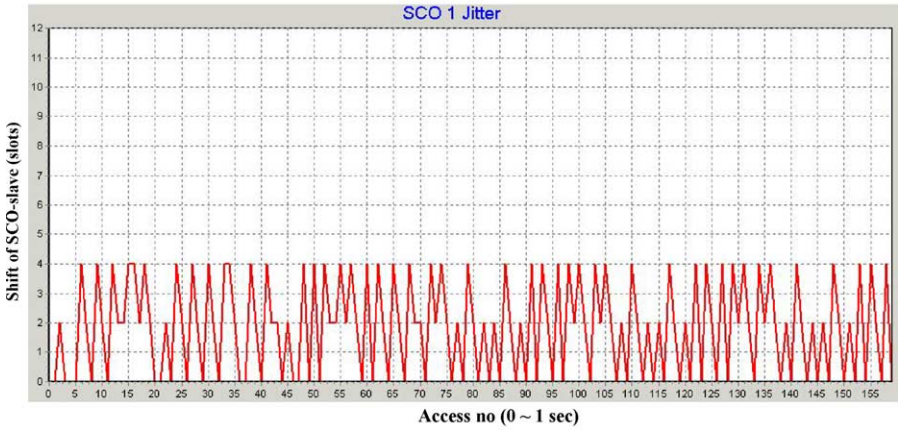


Fig. 8. Bandwidth allocation of each slave for test case 1, $K = 1 \sim 4$.



(a) Bandwidth allocation



(b) Shift of polling slots of the SCO-slave (slave 6)

Fig. 9. Bandwidth allocation and shift of the SCO-slave for test case 2, $K = 4$.

5.2. Simulation results

Simulation study has been conducted for performance evaluation of BBP and the scheduling algorithm. A couple of test cases are used to investigate the performance of BBP supporting all kinds of slaves. Some test cases are listed in Table 4. Before going through the figures obtained from the simulation study, we list some acronyms used in BBP for a better understanding of the figures:

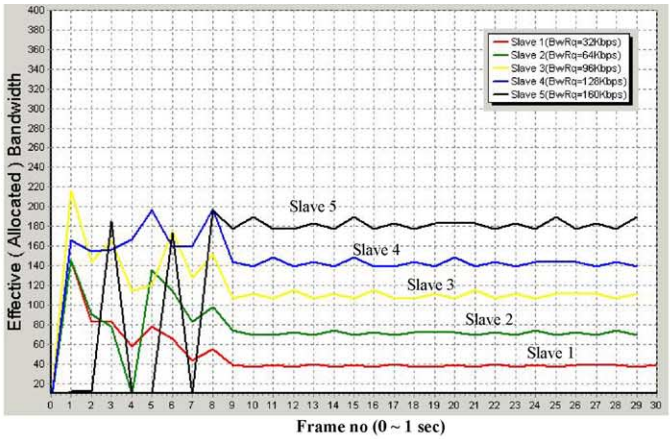
K : the maximum polling time in a frame for each slave
QoS-slave: a slave with bandwidth requirement (denoted by *BwRq*)
SCO-slave: a slave requesting a SCO (synchronous connection-oriented) link (the value of T_{sco} should be provided by the slave)
BE-slave: best-effort slave (no bandwidth requirement, no SCO link)

Test case 1 in Table 4 is used for investigating the flexibility of bandwidth allocation under different values of K. Bandwidth allocation of each slave in test case 1 for $K = 1-4$ is shown in Fig. 8(a)–(d). These figures have shown that a larger K for BBP can achieve more flexibility in bandwidth allocation at the expense of a longer time before reaching the equilibrium state.

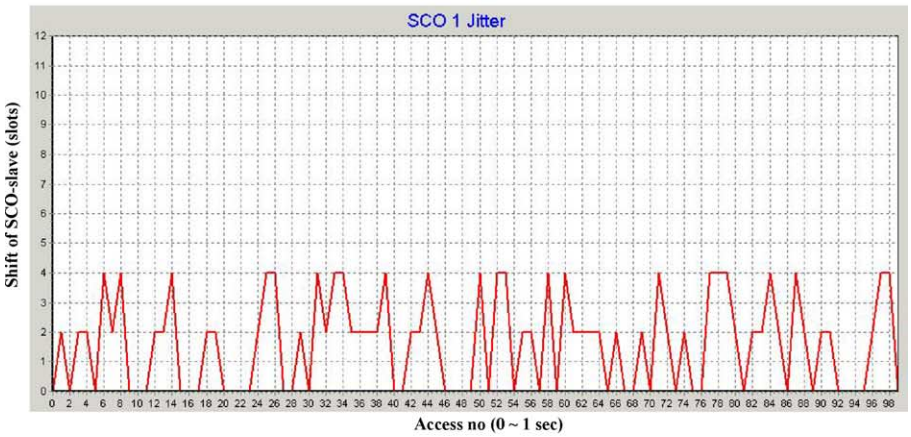
Test case 2 in Table 4 has demonstrated the presence of a SCO-slave. Bandwidth allocation of each QoS-slave and the shift of the polling slots for the SCO-slave (slave 6, $T_{sco} = 10$) in the case are displayed in Fig. 9(a) and (b), respectively. For this case of only one SCO-slave present in the piconet, the shift of the polling slot for the slave is always equal to or smaller than 4 slots as shown in Fig. 9(b). It is worth mentioning that the oscillation of bandwidth allocation for each QoS-slave in the equilibrium state in Fig. 9(a) is caused by the periodic insertion of the SCO-slave in the polling sequence and the pattern of the oscillation depends on the value of T_{sco} of the SCO-slave.

Two SCO-slaves are present in test case 3. Bandwidth allocation of each QoS-slave is displayed in Fig. 10(a). The shifts of polling slots of the SCO-slaves (slave 6 and slave 7) are shown in Fig. 10(b) and (c), respectively, which have also demonstrated that the maximum shift for two SCO-slaves present in the piconet is $4 + (2 - 1)*2 = 6$.

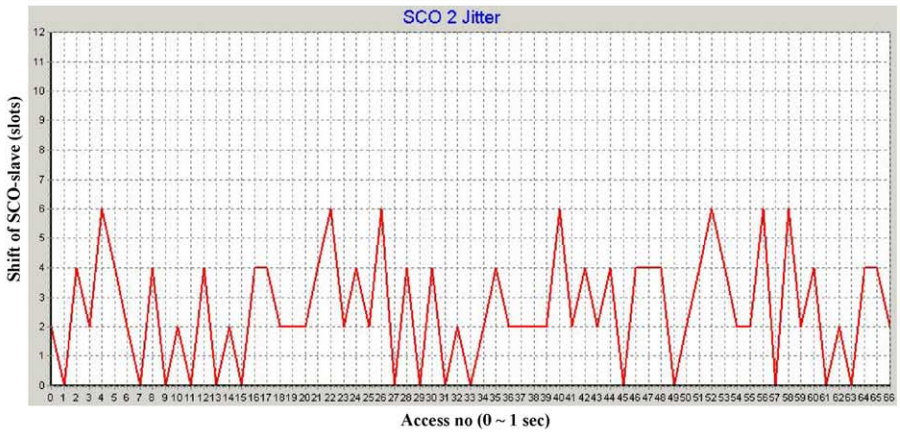
Test case 4 and test case 5 demonstrate the presence of the BE-slave in the piconet. Bandwidth allocation for test



(a) Bandwidth allocation



(b) Shift of polling slots of slave 6



(c) Shift of polling slots of slave 7

Fig. 10. Bandwidth allocation and shift of the SCO-slaves for test case 3, $K = 4$.

case 4 is shown in Fig. 11 in which the presence of the SCO-slave results in the oscillation of bandwidth allocation of the BE-slave. Bandwidth allocation displayed in Fig. 12 for test case 5 has demonstrated that the two BE-slaves get equal share of the rest capacity when the bandwidth requirements of all QoS-slaves are satisfied. Lastly, test case 6 has demonstrated the case in which a QoS-slave (slave 6) leaves earlier and its bandwidth share is then allocated to other QoS-slaves as displayed in Fig. 13.

5.3. Discussion

Simulation results have demonstrated the flexibility of multiple times of polling in bandwidth allocation.

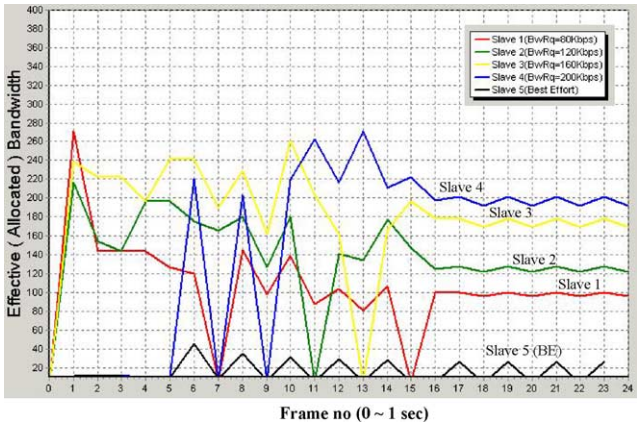


Fig. 11. Bandwidth allocation of the slaves for test case 4, $K = 4$.

The scheme of multiple times of polling seems to allow a greedy QoS-slave to have a large amount of bandwidth by requesting a large payload type. This case will not cause any problem as long as all other QoS-slaves still get a proper amount of bandwidth they have requested. On the other hand, if any of the other QoS-slaves does not have enough bandwidth, the bandwidth negotiation process is not finished. QoS-slaves that do not have enough bandwidth keep upgrading their payload type to request a larger amount of bandwidth in order to meet their requirements. As a consequence, the largest payload type (*DH5*K*) is assigned to all of the QoS-slaves and it results in equal share of the channel capacity for all QoS-slaves, which demonstrates the fairness of bandwidth allocation in BBP.

Moreover, there are cases that a malicious slave does not follow the rules of BBP and transmits packets of a larger payload type than it has been assigned. Once the case is detected, the master reduces the polling frequency of the malicious slave. The master either polls the slave fewer times than the slave has requested in a frame or polls the slave once in more than one frame as the penalty.

The scheme of BBP and associated scheduling algorithm only consider the QoS requirement (bandwidth or SCO) of the uplink data flows from slaves to the master. We assume the payload type of the downlink data flows from the master

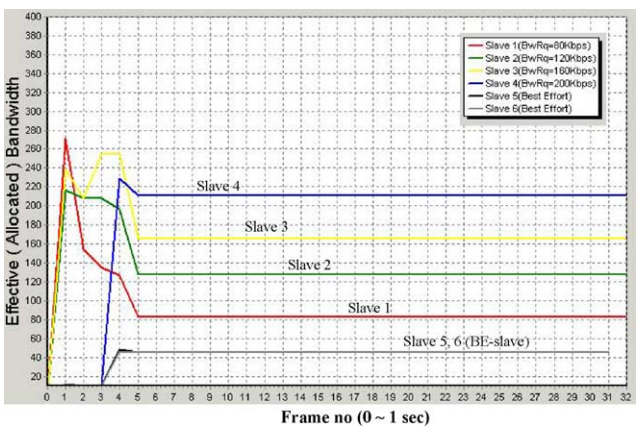


Fig. 12. Bandwidth allocation of the slaves for test case 5, $K = 4$.

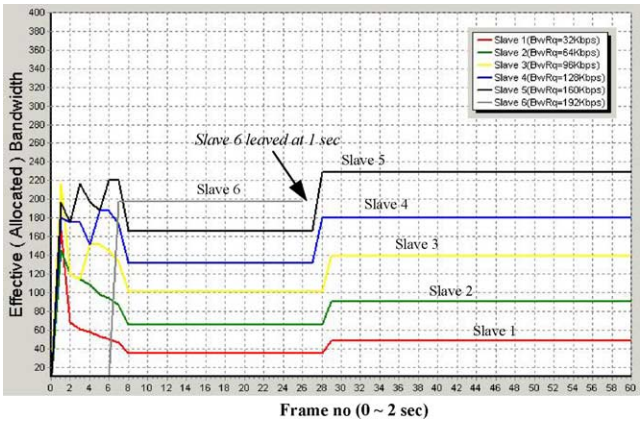


Fig. 13. Bandwidth allocation of the slaves for test case 6, $K = 4$.

to each slave is always DH1 in BBP. To consider the bandwidth requirement of downlink data flows, the master needs to maintain the bandwidth requirement to each slave and calculate a proper payload type according to the bandwidth requirement as well as the value of *PicoFrameLimit*. In the beginning of a frame, the master reserves enough slots for downlink flows by adding the number of slots to the value of *PicoFrameSize*, which is then passed to each slave. Moreover, the scheduling algorithm also needs to be modified to consider the payload types of both downlink and uplink data flows. The future work of the paper is to extend BBP scheme to support QoS for downlink data flows and propose a proper scheduling algorithm considering both downlink and uplink QoS requirements.

6. Conclusion

In this paper, a flexible bandwidth allocation scheme namely BBP has been proposed for supporting QoS in Bluetooth. BBP defines a framing structure of time and allows a slave to be polled multiple times in a frame to improve pure round robin scheme for high flexibility of bandwidth allocation. Calculation of the payload type and the polling time in a frame for a slave (QoS-slave) is according to its bandwidth requirement and the limit of the frame size controlled by the master. The master and slaves supporting BBP need to cooperate and exchange necessary information in the bandwidth negotiation process, and two new fields are added in the packet format of Bluetooth. The master either enlarges or shrinks the limit of the frame size for proper control of bandwidth allocation. Extension version of BBP has also been proposed to support slaves with the SCO link (SCO-slaves) and slaves with no requirement (best-effort slaves).

Adaptive T_{sco} has been adopted in the paper, in which SCO-slaves can select different values of T_{sco}. The scheduling algorithm of BBP provides a small upper bound of access delay for SCO-slaves, and thus the sniff mode can be applied to reduce power consumption. Simulation results have

demonstrated the efficiency and fairness of BBP in bandwidth allocation. Flexibility of bandwidth allocation depends on the maximum polling time (K) predefined by BBP. A larger K makes higher flexibility of bandwidth allocation. Nevertheless, a larger K also results in a longer latency before reaching the equilibrium state.

It is worth mentioning that BBP proposed in this paper adopts a distributed manner for bandwidth management, in which the information for bandwidth allocation is piggy-backed in data packets to and fro between the master and slaves. The concept of BBP can also be implemented in a centralized manner, in which the master collects bandwidth requirements of all slaves and calculates the best payload type for each slave in a frame. The master then notifies each slave of its payload type in the next frame. The centralized version of BBP does not introduce as much the latency of reaching the equilibrium state. However, the centralized version requires more computation power at the master than the distributed version.

References

[1] P. Bhagwat, Bluetooth: technology for short-range wireless apps, IEEE Internet Computing 5 (2001) 96–103.

[2] P. Johansson, M. Kazantzidis, R. Kapoor, M. Gerla, Bluetooth: an enabler for personal area networking, IEEE Network 15 (2001) 28–37.

[3] B. Chatschik, An overview of the Bluetooth wireless technology, IEEE Communications Magazine 39 (2001) 86–94.

[4] Bluetooth SIG, Specification of the Bluetooth System v1.0 B, Specification, vol. 1 and 2, December 1st 1999.

[5] The Bluetooth Web Site, http://www.bluetooth.com.

[6] A. Capone, M. Gerla, R. Kapoor, Efficient polling schemes for Bluetooth picocells, Proceedings of the IEEE ICC, 2001, pp. 1990–1994.

[7] A. Das, A. Ghose, A. Razdan, H. Saran, R. Shorey, Enhancing performance of asynchronous data traffic over the Bluetooth wireless ad-hoc network, Proceedings of the IEEE INFOCOM, 2001, pp. 591–600.

[8] R. Bruno, M. Conti, E. Gregori, Wireless access to Internet via Bluetooth: performance evaluation of the EDC scheduling algorithm, Proceedings of the ACM First Workshop on Wireless Mobile Internet, 2001, pp. 43–49.

[9] V. Sangvornvetphan, T. Erke, Traffic scheduling in bluetooth network, Proceedings of the Ninth IEEE International Conference on Networks, 2001, pp. 355–359.

[10] M. Kalia, D. Bansal, R. Shorey, Data scheduling and SAR for Bluetooth MAC, Proceedings of the IEEE VTC Spring (2000) 716–720.

[11] S. Grag, M. Kalia, R. Shorey, MAC scheduling policies for power optimization in Bluetooth: a master driven TDD wireless system, Proceedings of the IEEE VTC Spring (2000) 196–200.

[12] M. Kalia, D. Bansal, R. Shorey, MAC scheduling and SAR policies for Bluetooth: a master driven TDD pico-cellular wireless system, Proceedings of the IEEE Monuc'99, pp. 384–388.

[13] S. Chawla, H. Saran, M. Singh, QoS based scheduling for incorporating variable rate coded voice in Bluetooth, Proceedings of the IEEE International Conference on Communications (ICC 2001), 2001, pp. 1232–1237.

[14] I. Chakraborty, A. Kashyap, A. Kumar, A. Rastogi, H. Saran, R. Shorey, MAC scheduling policies with reduced power consumption and bounded packet delays for centrally controlled TDD wireless networks, Proceedings of the IEEE International Conference on Communications (ICC 2001), 2001, pp. 1980–1984.

[15] O. Sharon, E. Altman, An efficient polling MAC for wireless LANs, IEEE/ACM transactions on networking 9 (2001) 439–451.

[16] O. Kubbar, H.T. Mouftah, Broadband wireless networks: an investigation into the on-demand TDMA MAC protocol and the connection reestablishment performance guarantee, Proceedings of the IEEE International Conference on Communications (ICC 2001), 2001, pp. 1797–1801.

[17] O. Kubbar, H.T. Mouftah, Broadband wireless networks: an investigation into the traffic behavior control, and QoS guarantees, Proceedings of the IEEE International Conference on Communications (ICC 2000), 2000, pp. 985–989.

[18] R.S. Ranasinghe, L.L.H. Andrew, D.A. Hayes, D. Everitt, Scheduling disciplines for multimedia WLANs: embedded round robin and wireless dual queue, Proceedings of the IEEE International Conference on Communications (ICC 2001), 2001, pp. 1243–1248.

[19] M. Veeraraghvan, N. Cocker, T. Moors, Support of voice services in IEEE802.11 wireless LANs, Proceedings of the IEEE INFOCOM 2001, 2001, pp. 488–497.