A FLEXIBLE BANDWIDTH MANAGEMENT SCHEME IN BLUETOOTH

Chun-Chuan Yang and Chin-Fu Liu
Multimedia and Communications Laboratory
Department of Computer Science and Information Engineering
National Chi Nan University, Taiwan, R.O.C.
ccyang@csie.ncnu.edu.tw

Abstract

In this paper, 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. Moreover, BBP allows the master to poll a slave more than once in a time frame to achieve high flexibility for bandwidth allocation. Calculation of the payload type as well as the polling time in a frame for a slave with bandwidth requirement is presented in the paper. Actions of the master and the slave for supporting BBP are also presented in the paper. Simulation results have shown that a good performance and flexibility of bandwidth allocation can be achieved by BBP.

Key Words

Bluetooth, QoS, Bandwidth management, Polling

1. Introduction

Bluetooth [1, 2] 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, the channel is slotted and the medium access scheme is based on polling algorithm controlled by the master in a Time Division Duplex (TDD) 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 the research in Bluetooth focused on the performance improvement in terms of channel utilization. Different polling and scheduling schemes as well as SAR policies for improving utilization in Bluetooth had been proposed [3-7]. However, QoS for Bluetooth [8, 9] has attracted less attention in the literature.

Bandwidth allocation is direct but important in supporting QoS to some extent for Bluetooth. A good bandwidth management scheme should meet the various bandwidth requirements of slaves when the total bandwidth requirement does not exceed the channel capacity for slaves, and on the other hand maintain fairness when the channel is saturated. Fairness of bandwidth allocation indicates that when the bandwidth is over-requested, slaves still get the bandwidth they have requested or the equal share of the channel capacity.

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. Since synchronous connections in Bluetooth provide a circuit-oriented service with constant bandwidth and are based on a fixed and periodic allocation of slots, the proposed scheme mainly focuses bandwidth allocation for asynchronous connections.

The rest of the paper is organized as follows. The basic idea of the proposed scheme as well as the actions of slaves and the master are presented in section 2. Performance evaluation of the proposed scheme is presented in section 3. Finally section 4 concludes this paper.

2. Bandwidth-based Polling

2.1 Basic Idea

In order to allocate proper bandwidth, a framing structure of time is defined, and the master allocates proper number of slots for each active slave in a frame. The length of the frame should not be static but dynamic for flexible bandwidth allocation. Moreover, since only 3 payload types (1, 3, 5 slots) can be used for slaves, if the master equally polls each active slave in a time frame, only three levels of bandwidth can be allocated, which greatly reduce the flexibility of bandwidth allocation. Therefore, the proposed *Bandwidth-Based Polling* (*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),

Table I. Bytes and polling time for payload type in BBP

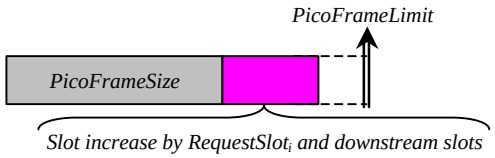
Payload type	DH1	DH3	DH5	DH6	DH8	DH10	DH11	DH13	DH15	…
Polling time	1	1	1	2	2	2	3	3	3	…
ByteCount	27	183	339	366	522	678	705	861	1017	…
Remarks				5+1	5+3	5+5	5+5+1	5+5+3	5+5+5	…

it is better for a slave to properly choose a larger payload for each poll, and the combinations of payload in BBP are shown in Table I. Byte count (*ByteCount*) as well as the polling time for a payload type are also included 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 a proper value for the maximum polling time, which also determines the maximum payload type. For instance, maximum polling time *K* results in maximum payload type DH5**K* in a time frame.

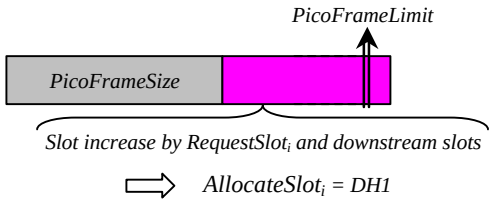
Calculation of the number of slots and the polling time in a time frame for a given bandwidth requirement is explained in the following. Given that the master restricts the frame size of the piconet within a limit value namely *PicoFrameLimit* (in *slots*) and the bandwidth requirement of $slave_i$ is $BwRQ_i$ (in *bps*), the number of bytes at most ($\#Bytes_i$) that need to be transmitted in a frame for $slave_i$ is $\#Bytes_i = BwRQ_i * PicoFrameLimit * 625ms$. Payload type for $slave_i$ in a frame should be the smallest one in Table I 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 crosses several time 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 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) *PicoFrameLimit*. Moreover, BBP adopts the soft-state bandwidth reservation, which means a 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 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 are explained respectively in the following sections.



(a) Successful upgrade (AllocateSlot$_i$ = RequestSlot$_i$)



(b) Unsuccessful upgrade (need to wait for the master enlarging PicoFrameLimit)

Figure 1. Upgrading the payload type

2.2 Actions at the Slave

While the master polls a slave, current *PicoFrameSize* and *PicoFrameLimit* are passed to the slave. The slave tries to upgrade its payload type from DH1. 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. However, upgrading the payload type will also increase the frame size of the piconet (*PicoFrameSize*). Upgrade of the payload type is successful only when the resulted frame size is still smaller than current *PicoFrameLimit*. The idea is illustrated in Figure 1. Increase of the slots for the upgrade is easily computed from the new payload type ($RequestSlot_i$) and 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. 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* in $RequestSlot_i$)/($BwRQ_i$*625μs)

$RequestSlot_i$ and $FrameLimit_i$ are both passed to the master for updating *PicoFrameSize* and *PicoFrameLimit*.

2.3 Actions at the Master

As mentioned above, the master passes *PicoFrameSize* and *PicoFrameLimit* to a slave and collects/records $RequestSlot_i$ and $FrameLimit_i$ returned by the slave. $RequestSlot_i$ and $FrameLimit_i$ represent the bandwidth request of the slave, and the master performs the same check (i.e. if the total number of slots is still under *PicoFrameLimit* or not) as the slave to grant the request or not. If the request is granted (in this case, the slave's $AllocateSlot_i = RequestSlot_i$), *PicoFrameLimit* remains unchanged and the increase of slots by the upgrade is added to *PicoFrameSize*.

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 might have the 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.

The master shrinks *PicoFrameLimit* on the leaving of a slave. In this case, the smallest $FrameLimit_i$ recorded at the master is assigned to the new value of *PicoFrameLimit* and the bandwidth negotiation process is activated again.

2.4 Discussion

If the bandwidth requirements of slaves in the piconet have not changed, BBP process reaches the equilibrium state when *PicoFrameLimit* and *PicoFrameSize* remain unchanged for two consecutive frames. A larger value of the maximum poling time in a frame may result in longer time before reaching the equilibrium state. Analysis of the worst case before reaching the equilibrium state for BBP is presented in the next section.

Moreover, in order to support BBP, the header of packets needs to be modified for information exchange between the master and each slave (from the master to the slave: *PicoFrameSize* and *PicoFrameLimit*, from the slave to the master: $RequestSlot_i$ and $FrameLimit_i$).

3. Performance Evaluation

3.1 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 combination for slaves, and (2) the longest time (in the worst case) needed to reach the equilibrium state for a given K. The total number of bandwidth combination for a given K is denoted by $N_{BW}(K)$, and the worst-case time to reach the equilibrium state is denoted by $T_E(K)$. Apparently, a larger K results in a larger $N_{BW}(K)$ but also a larger $T_E(K)$. Thus, there is a trade-off between

Table II. 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 III. $N_{PT}(K)$ vs. $N_{BW}(K)$ for S=7

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

$N_{BW}(K)$ *and* $T_E(K)$ in deciding the value of K. We assume only slaves with bandwidth request 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 $N_{BW}(K)$, we consider the combination of payload first. We denote the number of payload combination by $N_{PT}(K)$. 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 slaves (S is the number of active slaves in the piconet) is $C(3+S-1, S)$, which is actually the same as the number of ways to place S non-distinct objects into 3 distinct cells where a cell can hold more than one object. For a general K, there are 3K distinct cells (as shown in Table II) for S non-distinct objects. Thus, $N_{PT}(K) = C(3K+S-1, S)$.

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 slaves is actually the same for {(1, 5) (1, 5) (1, 5)} and {(2, 10) (2, 10) (2, 10)} for S=3. The actual number of bandwidth combination $N_{BW}(K)$ is calculated by a generator program of bandwidth combination. Values of $N_{PT}(K)$ and $N_{BW}(K)$ for S=7 are listed in Table III, 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 slave not satisfying its bandwidth requirement. The worst case happens when there is always a slave not satisfying its bandwidth requirement and the master enlarges

Table IV. Test cases for BBP

$BwRQ_i$ (Kbps)	Slave 1	Slave 2	Slave 3	Slave 4	Slave 5	Slave 6
Test Case 1	32	64	96	128	160	192
Test Case 2	50	75	100	125	150	175
Test Case 3	100	120	140	160	180	N/A

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:

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

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

3.2 Simulation results

We have conducted a simulation study for performance evaluation of BBP. Several test cases were investigated and three of them are listed in Table IV. We assume that all active slaves in the piconet require some amount of bandwidth and neither best effort slaves nor synchronous connections exist in the piconet. We also assume the downstream data from the master to each slave adopts 1-slot payload (DH1) in each poll.

Bandwidth allocation of each slave in the test cases for different values of K is illustrated in Figures 2 ~ 4. These figures have shown that a larger K for BBP can achieve more flexibility in bandwidth allocation at the expense of longer time before reaching the equilibrium state. Therefore, there is a tradeoff between flexibility of bandwidth allocation and time before reaching the equilibrium state while selecting a proper value of K for BBP.

4. Conclusion

Bandwidth-based Polling (*BBP*) for bandwidth management in Bluetooth is proposed in this paper. BBP adopts a dynamic framing structure of time, and the master allocates proper number of slots for each active slave in a frame. Moreover, BBP allows the master to poll a slave more than once in a time frame to achieve high flexibility in bandwidth allocation. Calculation of the payload type and the polling time in a frame for a slave with bandwidth requirement is presented in the paper. BBP adopts a progressive approach for bandwidth allocation, which crosses multiple time frames to finish. The master and slaves supporting BBP need to cooperate and exchange necessary information in the bandwidth negotiation process. Actions of the master and the slave are also presented in the paper. Simulation results have shown that a good performance and flexibility of bandwidth allocation are achieved by BBP.

References:

[1] Bluetooth SIG, *Specification of the Bluetooth System v1.0 B*, Specification Volume 1& 2, December 1st 1999.

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

[3] A. Capone, M. Gerla, and R. Kapoor, "Efficient polling schemes for Bluetooth picocells," Proc. IEEE ICC 2001, pp. 1990-1994.

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

[5] R. Bruno, M. Conti, and E. Gregori, "Wireless Access to Internet via Bluetooth: Performance Evaluation of the EDC Scheduling Algorithm," Proc. ACM 1st Workshop on Wireless Mobile Internet, 2001, pp. 43-49.

[6] M. Kalia, D. Bansal, and R. Shorey, "Data scheduling and SAR for Bluetooth MAC," Proc. IEEE VTC 2000-Spring, pp. 196-200.

[7] M. Kalia, D. Bansal, and R. Shorey, "MAC Scheduling and SAR policies for Bluetooth: A master Driven TDD Pico-Cellular Wireless System," Proc. IEEE Monuc'99, pp. 384-388.

[8] S. Chawla, H. Saran, and M. Singh, "QoS based scheduling for incorporating variable rate coded voice in Bluetooth," Proc. IEEE International Conference on Communications, 2001 (ICC 2001), pp. 1232-1237.

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

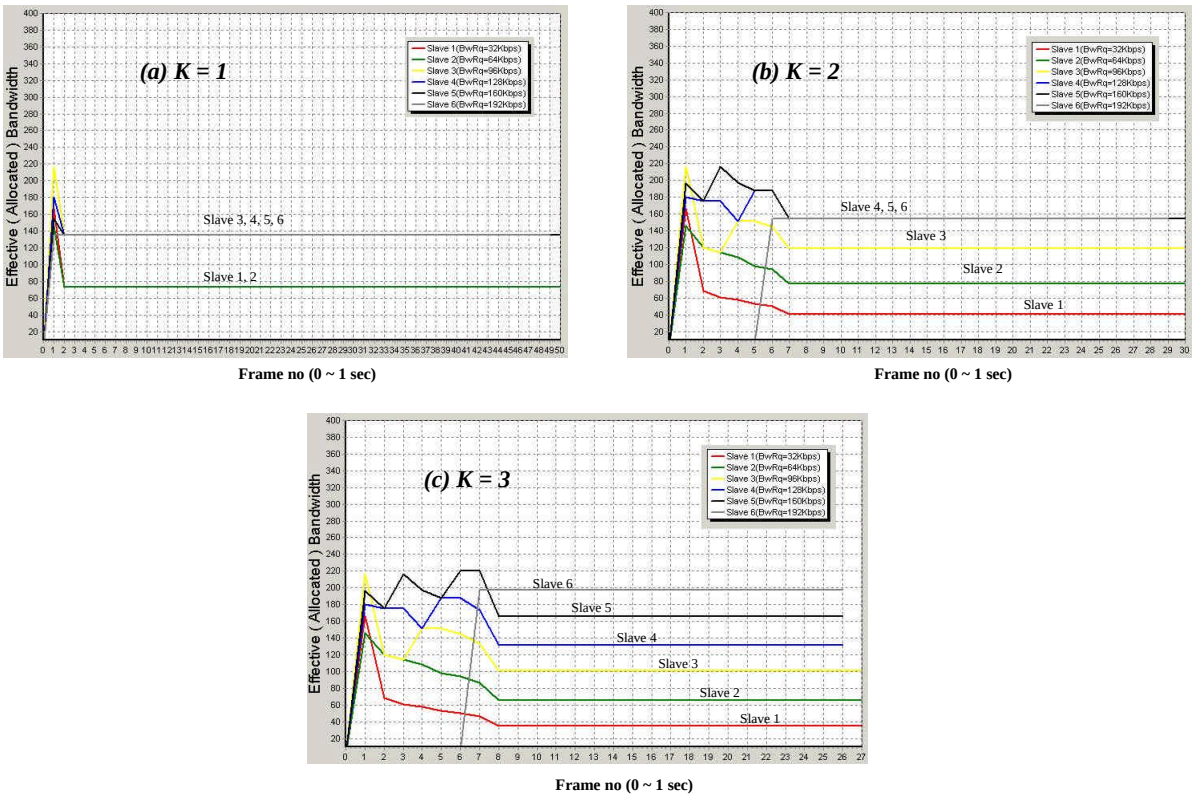


Figure 2. Bandwidth allocation of each slave for *test case 1, K = 1 ~ 3*

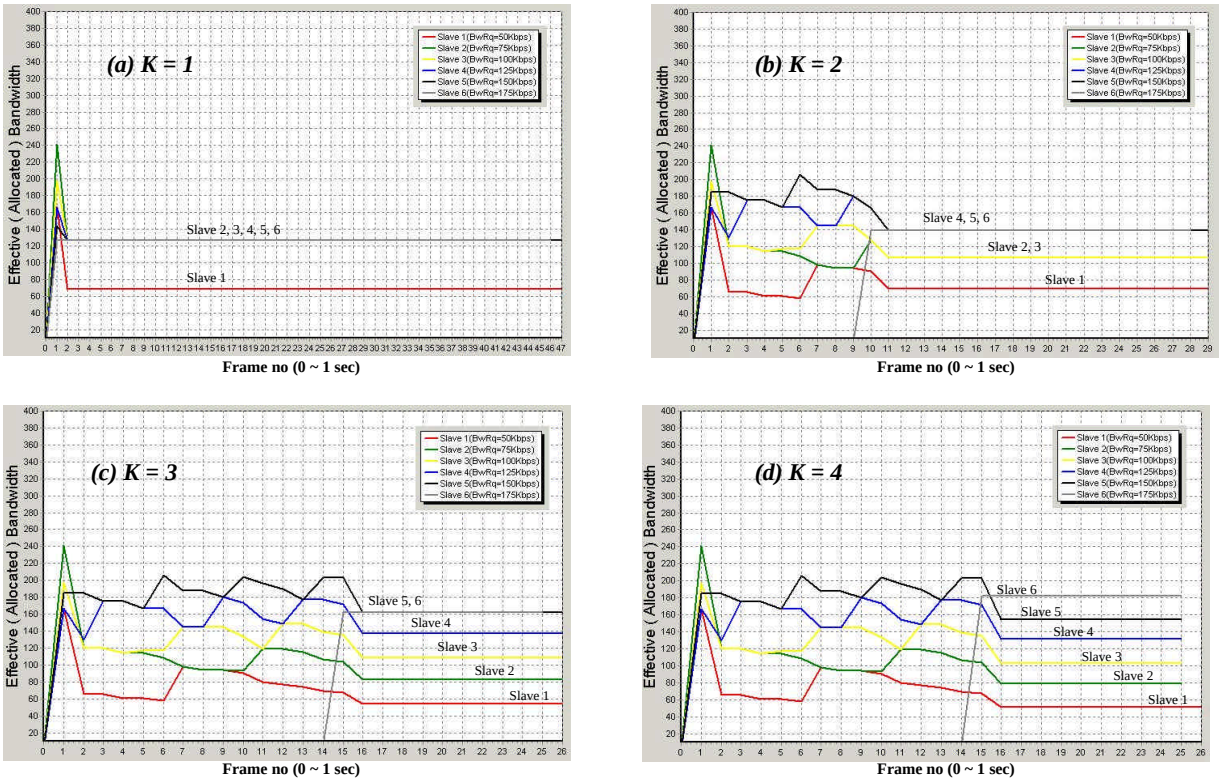
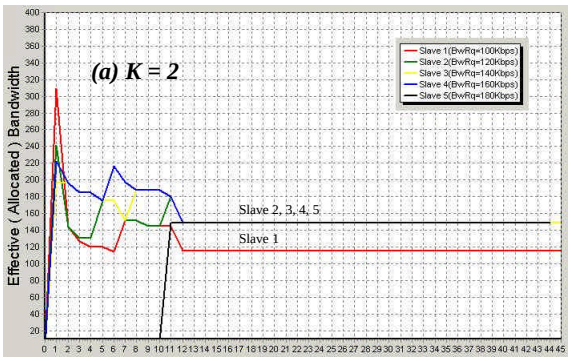
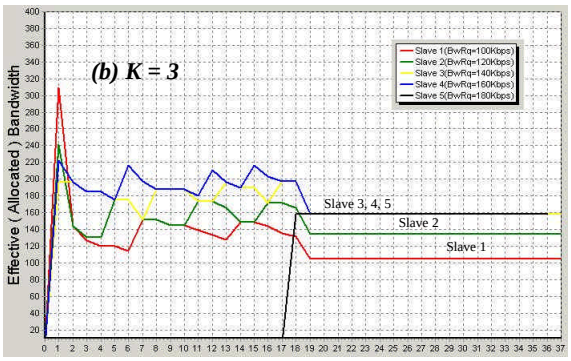
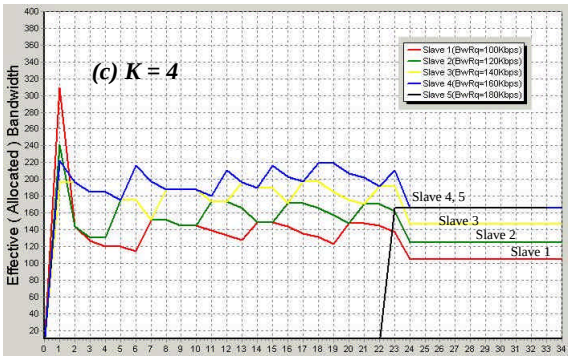


Figure 3. Bandwidth allocation of each slave for *test case 2, K = 1 ~ 4*







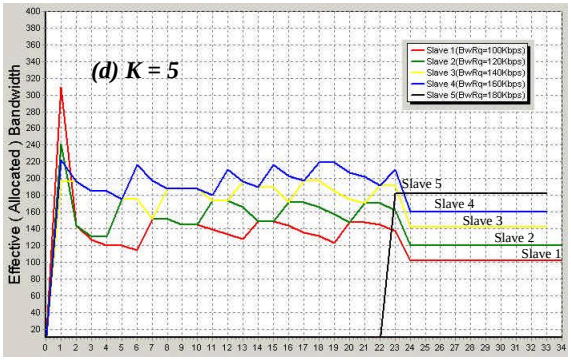


Figure 4. Bandwidth allocation of each slave for *test case 3, K = 2 ~ 5*