An Efficient Link Polling Policy by Pattern Matching for Bluetooth Piconets

TING-YU LIN[1], YU-CHEE TSENG[2] AND YUAN-TING LU[3]

[1]*Computer & Communications Research Laboratory, Industrial Technology Research Institute, Chu-Tung 310, Taiwan*
[2]*Department of Computer Science and Information Engineering National Chiao-Tung University, Hsin-Chu 300, Taiwan, ROC*
[3]*Department of Computer Science and Information Engineering National Central University, Chung-Li 320, Taiwan, ROC*
Email: {tylin,yctseng}@csie.nctu.edu.tw

Bluetooth has a master–slave configuration called a piconet. Unspecified in the Bluetooth standard, the link polling policy adopted by a master may significantly influence the bandwidth utilization of a piconet. Several works have been dedicated to this issue. However, none of them addresses the asymmetry of traffics between masters and slaves, and the different data packet types provided by Bluetooth are not fully exploited. In this paper, we propose an efficient pattern matching polling (PMP) policy for data link scheduling that properly resolves these deficiencies. A polling pattern is a sequence of Bluetooth packets of different type combinations (e.g. DH1/DH3/DH5/DM1/DM3/DM5) to be exchanged by a master–slave pair that can properly reflect the traffic ratio (i.e. asymmetry) of the pair. By judiciously selecting a proper polling pattern together with polling times for the link, the precious wireless bandwidth can be better utilized. The ultimate goal is to reduce the unfilled, or even null, payloads in each busy slot. In addition, an overflow mechanism is included to handle unpredictable traffic dynamics. Extensive simulations are presented to justify the capability of PMP in handling regular as well as bursty traffics.

Received 29 July 2002; revised 29 August 2003

1. INTRODUCTION

With master-driven, short-range radio characteristics, Bluetooth [1] is a promising wireless technology for personal-area networks (PANs), and has attracted much attention recently [2, 3]. The smallest network unit in Bluetooth is a *piconet*, which consists of one master and one or more slaves. A piconet owns one frequency-hopping channel, which is controlled by the master in a time-division duplex manner. A time slot in Bluetooth is 625 μs. The master always starts its transmission in an even-numbered slot, while a slave, on being polled, must reply in an odd-numbered slot. By interconnecting multiple piconets, a larger area network called *scatternet*, can be formed. In the literature, the scatternet performance issues are addressed in [4, 5, 6]. How to form scatternets is discussed in [7, 8, 9, 10]. In this paper, we will focus on the data link polling issue within a piconet involving one master and multiple slaves.

According to the Bluetooth protocol stack, the bottom layer is the Bluetooth Baseband, which controls the use of the radio. On top of the Baseband is the link manager (LM), which is responsible for link configuration and control, security functions, and power management. The corresponding protocol is called the link manager protocol (LMP). The logical link control and adaptation protocol (L2CAP) provide connection-oriented and connectionless datagram services to upper-layer protocols. Two major functionalities of L2CAP are protocol multiplexing, and segmentation and reassembly (SAR). The SAR function segments an L2CAP packet into several Baseband packets for transmission over the air, and reassembles those at the receiving side before forwarding them to the upper layer.

Two physical links are supported in Bluetooth: asynchronous connectionless (ACL) for data traffic and synchronous connection-oriented (SCO) for time-bounded voice communication. SCO voice links always have higher priority than ACL data connection. Three SCO packets are defined: HV1, HV2 and HV3. HV stands for high-quality voice. An HV1 packet carries 10, HV2 carries 20 and HV3 carries 30 information bytes. To achieve the specified 64 Kbps speech rate, the HV1 packet has to be delivered every two time slots, while the HV2 and HV3 need to be delivered every four and six time slots respectively. These packets are all single slot and are transmitted over reserved intervals without going through L2CAP. The remaining slots can be used by the ACL link. Section 2.1 will detail the ACL packets. The coexistence of SCO and ACL links is modeled and evaluated in [11, 12]; the result demonstrates that the existence of SCO links does significantly reduce the data rate of ACL connections.

This paper focuses on the management of the Bluetooth ACL link involving one master and multiple slaves. Unspecified in the Bluetooth standard, the link polling policy adopted by the master may significantly influence the bandwidth utilization of a piconet. A number of works have addressed the polling issue in a piconet [13, 14, 15, 16, 17]. References [16, 17] consider the coexistence of an ACL link with an SCO link (HV3). Since the HV3 link will partition time slots into a number of free segments each of four slots, each master–slave pair can only exchange data by 1-to-1, 3-to-1 or 1-to-3 slot patterns. According to the available patterns and the leading packet sizes at the heads of the buffers, each master–slave pair is prioritized properly, based on which polling policy is selected. A K-fairness scheme is further proposed to guarantee channel access for master–slave pairs with low priorities (starvation avoidance). A learning function is proposed in [14] to predict the polling interval for each master–slave pair. Hence bandwidth waste is reduced. Since the next polling time is known, the slave may go to the low-power sniff mode to save energy. Also, bounded packet delay is guaranteed. However, the learning function is pretty complex and the cost of control messages could be significant.

More practical polling policies are proposed in [13, 15]. In [13], three polling schemes are proposed: pure round robin (PRR), exhaustive round robin (ERR), and exhaustive pseudo-cyclic master queue length (EPM). Assuming a fixed serving order, PRR naively polls each slave sequentially. With a fixed order, ERR will exhaust each master–slave pair's payload on both sides in each polling before moving on to the next slave. As for EPM, it is similar to ERR except that the polling order is dynamically adjusted in each round based on the master's queues for slaves. The SAR and polling issues are addressed in [15]. Three polling strategies are proposed: adaptive flow-based polling (AFP), sticky and sticky adaptive flow-based polling (StickyAFP). A new flow bit is defined for each master–slave pair. The bit is set to TRUE if the buffered data at any entity is above a threshold. AFP then dynamically adjusts each slave's polling interval based on the corresponding flow bit. Whenever the flow bit is 1, the polling interval is reduced to the minimum, and whenever a poll is replied by a NULL packet, the polling interval is doubled if a certain upper bound is not exceeded. The sticky strategy defines a new parameter, num_sticky, to indicate the maximum number of consecutive polls that a master–slave pair can be served, under the condition that the corresponding flow bit is 1. Finally, the StickyAFP policy is a combination of the above two.

From the above reviews, we observe two deficiencies associated with the existing methods. First, they all fail to address the asymmetry of traffics between masters and slaves. That is, each master–slave pair may exhibit distinct traffic load in each direction. Second, the different packet types provided by Bluetooth are not fully exploited to match the traffic need.

In this paper, supposing that the traffic ratio between each master–slave pair can be approximated, we propose a pattern matching polling (PMP) policy for ACL link scheduling. A polling pattern is a sequence of Bluetooth packets of different type combinations (e.g. DH1/DH3/DH5/DM1/DM3/DM5) to be exchanged by a master–slave pair. Since each Bluetooth packet has its payload efficiency, different patterns can reflect different traffic ratios of the two sides. We show how to judiciously select the polling pattern, as well as the polling time, that best matches each master–slave pair's traffic characteristics. The ultimate goal is to reduce the unfilled, or even null, payloads in each packet. As a result, the traffic asymmetry problem can be properly handled, and the precious wireless bandwidth can be better utilized. We demonstrate how to apply this policy to single- and multi-slave environments. In addition, an overflow mechanism is included to handle unpredictable traffic dynamics. This further enhances the robustness of our PMP policy to deal with bursty traffics. Extensive simulation results are presented to justify the capability of the proposed PMP policy in processing regular as well as bursty traffics.

TABLE 1. Summary of Bluetooth ACL data packets.

Type	Payload header (bytes)	User payload (bytes)	FEC	CRC	Bandwidth efficiency (bytes/slot)
DM1	1	0–17	2/3	yes	17
DH1	1	0–27	no	yes	27
DM3	2	0–121	2/3	yes	40.3
DH3	2	0–183	no	yes	61
DM5	2	0–224	2/3	yes	44.8
DH5	2	0–339	no	yes	67.8
AUX1	1	0–29	no	no	29

The rest of this paper is organized as follows. Preliminaries are provided in Section 2. Section 3 proposes the PMP policy. Performance evaluation is presented in Section 4. Finally, Section 5 concludes the paper.

2. PRELIMINARIES

2.1. Bluetooth Data Packets

Since our main focus is on ACL connections, we need to introduce the available packet types in Bluetooth. Table 1 summarizes all the supported packet types. DM stands for data-medium rate, and DH for data-high rate. DM packets are all 2/3-FEC encoded to tolerate possible transmission errors. Not encoded by FEC, DH packets are more error-vulnerable, but can carry more information. DM1/DH1 packets occupy one time slot, while DM3/DH3 and DM5/DH5 packets occupy three and five time slots, respectively. The AUX1 packet is similar to DH1, but has no CRC code. We define bandwidth efficiency as the number of payload bytes per slot. From Table 1, we see that DH5 has the highest efficiency, which is followed subsequently by DH3, DM5, DM3, AUX1, DH1 and DM1.

By monitoring the channel conditions, a Bluetooth unit can pick the proper packet types (DM or DH) for use. However, in this paper, we assume an error-free environment and only consider DH1/DH3/DH5 packets. For an error-prone environment, our PMP policy can be tailored to include DM1/DM3/DM5 packets easily.

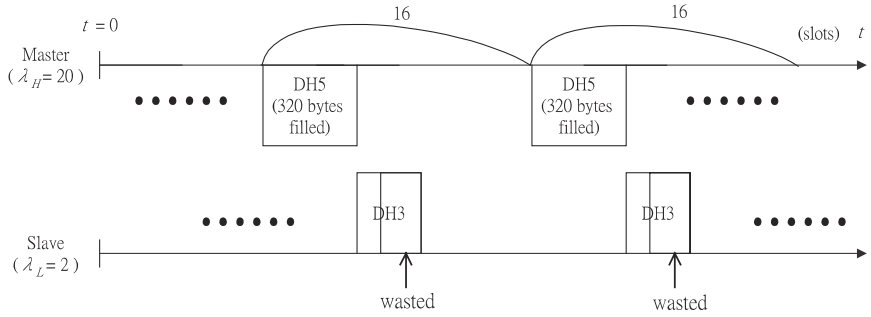

FIGURE 1. A naive greedy polling example.

2.2. The ACL link polling problem

In this paper, we consider a polling problem as follows. Suppose a long-term scenario (e.g. remote data exchange through TCP) where communication traffics in both directions (up-/down-link) have been stable and approached certain average arrival rates. In a piconet, we assume that from history, or by approximation, the average traffic arrival rates of each pair of master and slave are known factors. Note that these rates are not necessarily the same for all master–slave pairs. In addition, unpredictable, but rare, bursty traffics may appear on any side. The objective is to determine a good polling policy that should be adopted by a master as well as a replying policy of a slave, when being polled. The ultimate goal is to increase bandwidth efficiency while keeping delays low.

To motivate this problem, we demonstrate a naive greedy protocol (NGP) solution below (later on we will show a better solution). Suppose that a master–slave pair has traffic loads of 20 and 2 bytes/slot in each direction. Since DH5 is most bandwidth-efficient, a greedy approach may work as follows. The master may always delay its polling time until a DH5 packet is full or close to full. A possible scenario is shown in Figure 1, where the master always polls the slave whenever it has collected $[339/20] \times 20 = 320$ bytes, which fit into a DH5 packet. On the other side, the slave may have collected $[339/20] \times 2 = 32$ bytes, and will reply with a DH3 packet. Then the same polling pattern will be repeated every 16 time slots. As can be observed, although all forward packets are almost fully loaded, the backward packets are hardly filled, resulting in a lot of bandwidth waste. Since $16(20+2)$ bytes are delivered in every $5+3$ slots, the bandwidth efficiency is 44.

In general, suppose that the master and slave have loads of λ_H and λ_L (bytes/slot), respectively, and $\lambda_H \geq \lambda_L$. In every $339/\lambda_H$ slot, the master will poll the slave with a DH5 packet. In response, the slave may return $\alpha = 339 \cdot \lambda_L/\lambda_H$ bytes with the smallest possible packet of $f(\alpha)$ slots, where

$$f(\alpha) = \begin{cases} 1, & \text{if } \alpha \leq 27, \\ 3, & \text{if } 27 < \alpha \leq 183, \\ 5, & \text{otherwise.} \end{cases}$$

Then the bandwidth efficiency is

$$\beta = \frac{339 + \alpha}{5 + f(\alpha)}. \qquad (1)$$

The value of β heavily depends on λ_H and λ_L. Taking the above example, we have $\beta = 46.6$. This is still far beyond the best possible efficiency of 67.8 offered by DH5.

3. THE PMP POLICY

The basic idea of PMP is to use different combinations of Bluetooth packet types to match the traffic characteristics of masters and slaves. For ease of presentation, only DH1/DH3/DH5 will be used (however, our result can be extended to other packet types easily).

3.1. Polling patterns

In this subsection, we consider only one master–slave pair. Under long-term steady communication patterns, let λ_M and λ_S be their traffic loads, respectively (unit = bytes/slot). Let $\lambda_H = \max\{\lambda_M, \lambda_S\}$ and $\lambda_L = \min\{\lambda_M, \lambda_S\}$. Also, let ratio $\rho = \lambda_H/\lambda_L$. We denote by N_H and N_L the units with loads λ_H and λ_L respectively. Note that in reality, traffic arrival is by packets, not by bytes. Our assumption is that even if traffic arrives in packets, in the long run, it will still exhibit some steady arrival pattern that can be modeled by a byte arrival process. It is based on this model that we derive our results. For simplicity, we may use numbers 1/3/5 to represent DH1/DH3/DH5 packets.

A polling pattern is a sequence of packet types that will be exchanged by a master–slave pair. Let k be a positive integer. A length-k pattern consists of two k-tuples: $(H_1, H_2, \ldots, H_k)$ and $(L_1, L_2, \ldots, L_k)$, where $H_i, L_i = 1$, 3, or 5, each representing a packet type. The former are packet types used by unit N_H, and the latter by N_L. Intuitively, the sequence of packets $(H_1, L_1, H_2, L_2, \ldots, H_k, L_k)$ will be exchanged by N_H and N_L, and the sequence will be repeated periodically, as long as the ratio ρ is unchanged and there is no bursty traffic. For instance, when length $k = 1$, there are four available patterns, as shown in Figure 2a, which offers four different traffic ratios. Note that other patterns not listed in the figure also exist, such as $H_1 = 3$ and $L_1 = 3$. However, since the offered ratio will be equal to that of $H_1 = 5$ and $L_1 = 5$ and the bandwidth efficiency will be lower, we omit such a possibility in the figure. By increasing the pattern length to $k = 2$, Figure 2b summarizes all possible patterns. Figure 3 illustrates this concept.

	Pattern 1	Pattern 2	Pattern 3	Pattern 4
N_H	(5)	(5)	(3)	(5)
N_L	(5)	(3)	(1)	(1)
Traffic ratio (ρ_i)	$\rho_1 = 1.0$	$\rho_2 = 1.86$	$\rho_3 = 6.8$	$\rho_4 = 12.6$

(a)

	Pattern 1	Pattern 2	Pattern 3	Pattern 4	Pattern 5	Pattern 6
N_H	(5, 5)	(5, 5)	(5, 5)	(5, 1)	(5, 5)	(5, 3)
N_L	(5, 5)	(5, 3)	(5, 5)	(3, 1)	(5, 1)	(3, 1)
Traffic ratio (ρ_i)	$\rho_1 = 1.0$	$\rho_2 = 1.3$	$\rho_3 = 1.43$	$\rho_4 = 1.75$	$\rho_5 = 1.86$	$\rho_6 = 2.49$

	Pattern 7	Pattern 8	Pattern 9	Pattern 10	Pattern 11
N_H	(5, 5)	(3, 1)	(3, 3)	(5, 3)	(5, 5)
N_L	(3, 1)	(1, 1)	(1, 1)	(1, 1)	(1, 1)
Traffic ratio (ρ_i)	$\rho_7 = 3.23$	$\rho_8 = 3.9$	$\rho_9 = 6.8$	$\rho_{10} = 9.7$	$\rho_{11} = 12.6$

(b)

FIGURE 2. Traffic ratios supported by pattern lengths (a) $k = 1$ and (b) $k = 2$.

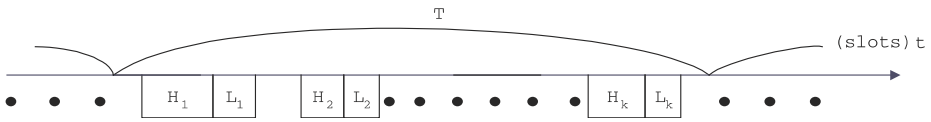

FIGURE 3. Illustration of our PMP policy with pattern length = k.

As k grows, the number of offered traffic ratios ρ will increase exponentially. On the other hand, the computational complexity to obtain all available traffic ratios also increases exponentially for larger k. In reality, we would not use a k value that is too large. This issue will be further investigated through simulations. Figure 4 illustrates the distribution of all supported traffic ratios for $k = 1, \ldots, 10$. As can be expected, with a larger k, our PMP policy could be more flexible. However, note that the set of traffic ratios supported by a larger k is not necessarily a superset of that of a smaller k. Hence, a longer pattern does not necessarily match the traffic need better than a shorter one.

Let K be a system parameter, which represents the largest allowable pattern length that can be used. Below, we derive the bandwidth efficiency β given a pattern $(H_1, H_2, \ldots, H_k)$ and $(L_1, L_2, \ldots, L_k)$, where $k \leq K$. First, we need to define a period T during which we can execute one iteration of the pattern. The basic idea is to fill the payloads of all available packets as much as possible. As a result, we define T to be (unit = slots)

$$T = \min\left\{ \frac{f(H_1) + f(H_2) + \cdots + f(H_k)}{\lambda_H}, \frac{f(L_1) + f(L_2) + \cdots + f(L_k)}{\lambda_L} \right\}, \quad (2)$$

where

$$f(i) = \begin{cases} 27, & \text{for } i = 1, \\ 183, & \text{for } i = 3, \\ 339, & \text{for } i = 5. \end{cases}$$

Here we take a min function because otherwise buffer overflow may occur after a long time. In a period of T slots, the expected number of bytes that will be transmitted is $\lambda_H \cdot T + \lambda_L \cdot T$. Divided by the total number of slots used, the bandwidth efficiency is

$$\beta = \frac{\lambda_H \cdot T + \lambda_L \cdot T}{(H_1 + H_2 + \cdots + H_k) + (L_1 + L_2 + \cdots + L_k)}. \quad (3)$$

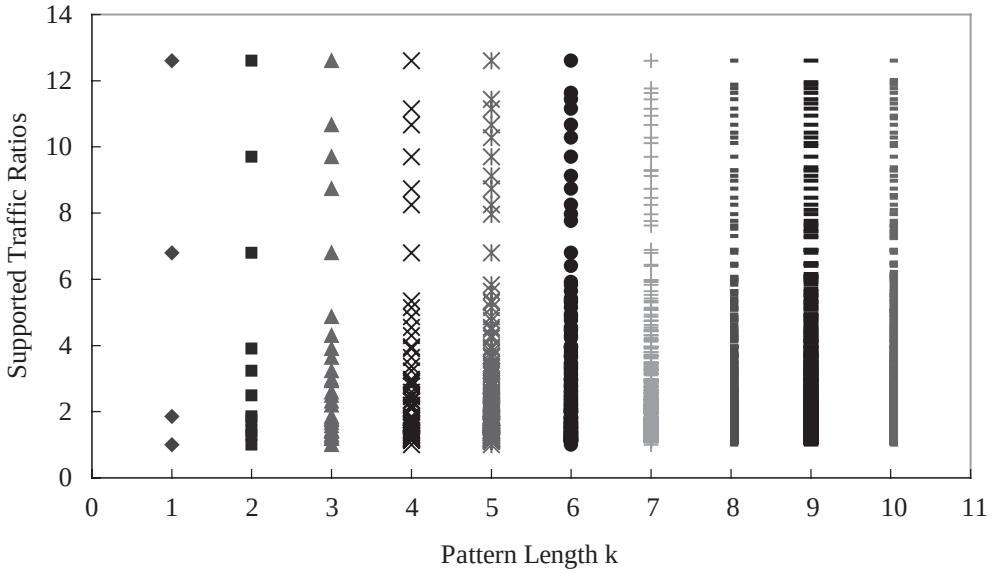

FIGURE 4. The distribution of supported traffic ratios for pattern lengths $k = 1, \ldots, 10$.

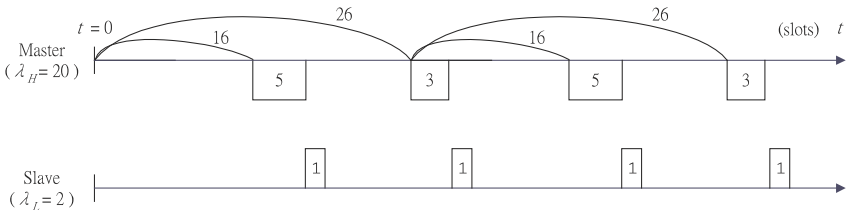

FIGURE 5. The PMP policy given as $\lambda_H = 20$ for the master and as $\lambda_L = 2$ for the slave ($K = 3$).

3.2. Polling policy for one master–slave pair

We have derived the bandwidth efficiency of a polling pattern. Given traffic loads λ_H and λ_L of a master–slave pair, we propose to choose the polling pattern that gives the highest bandwidth efficiency for use. Let $(H_1, H_2, \ldots, H_k)$ and $(L_1, L_2, \ldots, L_k)$ be the best pattern. Below, we present the corresponding polling policy. Note that here a time unit is one time slot, and we assume for simplicity that our protocol starts from slot 0.

Step 1. Initially, let $t = 0$ and $i = 1$.

Step 2. Define $j = ((i - 1) \bmod k) + 1$. The next polling is expected to appear Γ_j time slots after t, where

$$\Gamma_j = \begin{cases} \max\left\{\dfrac{H_1 + H_2 + \cdots + H_j}{\lambda_H}, \dfrac{L_1 + L_2 + \cdots + L_j}{\lambda_L}\right\}, \\ \quad \text{for } j = 1, \ldots, k-1, \\ \min\left\{\dfrac{H_1 + H_2 + \cdots + H_k}{\lambda_H}, \dfrac{L_1 + L_2 + \cdots + L_k}{\lambda_L}\right\}, \\ \quad \text{for } j = k. \end{cases}$$

Then at time slot $t + \Gamma_j$, the master polls the slave with a proper packet type H_j or L_j (depending on whether it has the higher or lower load). In return, the slave replies with a proper packet type H_j or L_j.

Step 3. If $j = k$, then move t ahead by setting $t = t + T$, where T is as defined in Equation (2). Finally, let $i = i + 1$ and go to Step 2.

The above steps may be repeated infinitely until the master determines that the traffic loads have changed. Note that in our protocol, a master and a slave will determine their own traffic loads λ_M and λ_S. This load information can be exchanged by a user-defined control packet. Since both the master and the slave will run the same algorithm to determine the polling pattern, a consistent polling pattern will be used. So only the load information needs to be exchanged, and there is no special packet to notify the chosen polling pattern. When the traffic rate on either side changes, the master and slave should exchange with each other by piggybacking the new traffic load information. This implies that a user-defined control packet format is needed for this purpose. Then the best polling pattern for this pair should be re-determined. In this work, we do not handle misbehaving slaves. Instead, we assume cooperative slaves, which always follow the polling algorithm based on computed polling patterns.

In the polling algorithm, index i is the current number of polls being counted starting from the very beginning, while index j represents the number of polls in every polling pattern cycle. For $j = 1, \ldots, k - 1$, Γ_j is the time slot when both entities already have sufficient data to fill the next predicted packet type (reflected by the max function). For $j = k$, $\Gamma_j = T$ and then completes one pattern cycle. Figure 5 illustrates how our PMP policy solves the earlier example of $\lambda_H = 20$ and $\lambda_L = 2$. Assuming $K = 3$, Equation (3) can be used to determine the best pattern to be (5, 3) for the master,

and (1, 1) for the slave. Here, $\Gamma_1 = 16$ and $\Gamma_2 = 26$. This gives a bandwidth efficiency of $\beta = 57.2$, which is about 23% better than the earlier NGP policy.

The above policy is derived based on an ideal assumption that the traffic pattern behaves perfectly as we predicted. However, in practice, traffic may not be as regular as we expect, and in some cases bursty traffic may appear. For this reason, we further enhance our policy by defining an overflow bit to prevent buffer overloading. The overflow bit is set to TRUE whenever an entity (master or slave) finds its buffer reaching a pre-defined threshold value. On discovering such a situation, the entity will ignore the polling pattern and will immediately send out a DH5 packet to relieve its backlog. Here we assume that the buffer status is checked whenever an entity is scheduled to transmit data as requested by our PMP policy. In such a case, the overflow bit will be piggybacked in the DH5 packets to inform the other entity. This overflow bit may be placed in one of the four reserved bits in the 2-byte payload header of DH5. The entity that does not have the overflow situation also stops its pre-defined pattern, when seeing overflow = 1, and selects a packet type that can cover as many queued data as possible. The polling activity will be repeated in a back-to-back manner, until both sides' buffers are emptied, after which we will reset the polling pattern by letting $i = 1$ and goto Step 2. Also, we will move t to the current time slot.

3.3. Polling policy for multiple master–slave pairs

For an environment with only one master–slave pair, bandwidth efficiency may not be an important factor, since we may have plenty of free slots and it may not be desirable to adopt the PMP policy to save bandwidth at the cost of longer packet delays due to waiting. However, for an environment with multiple master–slave pairs, bandwidth efficiency becomes more critical. How efficiently slots are utilized will significantly affect the maximum throughput that can be supported in a piconet. In Section 3.2, we first proposed a polling policy for a single master–slave pair. In this section, we describe a polling policy for multiple master–slave pairs based on the approach for a single pair.

When there are multiple master–slave pairs in an ACL link, we will choose for each pair, the most bandwidth-efficient pattern. From the pattern, the polling times are determined as mentioned earlier. As there are multiple master–slave pairs, the master should place all polling activities in a time line and conduct polling one by one. However, the polling activities of different master–slave pairs may overlap each other in time. In this case, we adopt the following rules to determine the polling priorities.

- For two overlapping polling activities, we compare their leading slots. The one with an earlier leading slot will be served first. The other one will be queued and served immediately after the earlier one is completed.
- When the leading slots are the same, the last polling in a polling pattern has a higher priority. Intuitively, we consider such polling to be more urgent since it is supposed to consume all traffic loads of a master–slave pair in each pattern interval (i.e. T) to avoid buffer overflow.
- In case of ties in both the above rules, the AM_ADDRs of slaves are compared to break the ties such that the smaller one wins.

4. PERFORMANCE EVALUATION

To demonstrate the effectiveness of the proposed PMP solution, we develop a C++ simulator to observe the performance. Two measurement metrics are evaluated: bandwidth efficiency and average delay time. We adopt the simulation assumptions suggested in [15] that the master keeps separate buffers for slaves, and that the buffer size for each entity is 2048 bytes. The buffer threshold to turn the overflow bit on is 80%. Each experiment lasts for 80,000 time slots. Three other policies are compared: NGP (described in Section 2.2), ERR [13], and StickyAFP [15]. In the ERR approach, the master can only observe its local queues without knowledge of slaves' buffer status. A control bit indicating buffer emptiness is piggybacked in slave-to-master packets, so that the master can decide to stop polling or not. In StickyAFP, the initial polling interval $P_0 = 14$ (slots), and the maximum allowable polling interval $P_{\max} = 56$ (slots). The flow bit is set to TRUE whenever the buffer exceeds 80%. The parameter num_sticky is set to 16 packets as suggested in [15]. For both ERR and StickyAFP, whenever a master/slave decides to send, it will examine its queue and choose the most appropriate packet type that can consume as many bytes in its queue as possible. Traffic is modeled by a byte arrival process with a certain rate. From time to time, we also inject a large volume of data to model bursty traffic.[1]

4.1. Single master–slave pair without bursty traffic

We first simulate one master–slave pair with Poisson traffic arrival rates λ_H and λ_L (bytes/slot) at the master and slave sides, respectively. By fixing $\lambda_L = 1$, we adjust λ_H to observe how different traffic ratios affect the network performance.

Figure 6 illustrates the bandwidth efficiency and average delay against various ratios $\rho = \lambda_H/\lambda_L$. Four values of K (3, 5, 7 and 9) for our PMP strategy are simulated. When $\rho \leq 12.6$, our PMP strategy successfully improves the bandwidth efficiency with moderate average delay. For NGP, when $\rho \leq 12.6$, only three ρs (1, 1.85 and 12.6) can be handled properly with high bandwidth efficiency. For $\rho > 12.6$, our PMP always selects the pattern $H_1 = 5$ and $L_1 = 1$, and thus acts the same as NGP. StickyAFP and ERR achieve low bandwidth efficiency due to very frequent polls and inadequate selections of packet types.

Note that for PMP, the case of $K = 7$ and $K = 9$ only slightly improve over $K = 5$ in terms of bandwidth efficiency. With $K = 3$, our PMP already outperforms

[1] We comment that the ERR and StickyAFP are designed based on a packet arrival process, but adopting a byte arrival process would not hurt their performance.

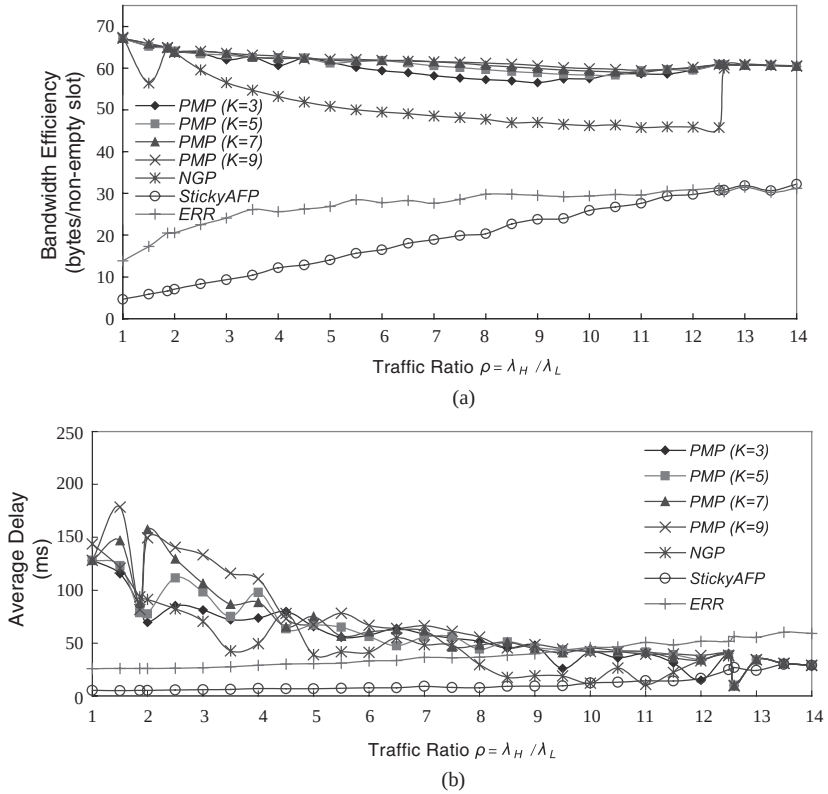

FIGURE 6. Effect of traffic ratio ρ when there is one master–slave pair: (a) bandwidth efficiency and (b) average delay.

other polling schemes significantly. Hence we conclude that it suffices to set K between 3 and 5 to balance between computational cost and performance.

4.2. Single master–slave pair with bursty traffic

In this experiment, on top of the regular (Poisson) traffics at the master and slave sides, we also inject irregular bursty traffics. The bursty traffic occurs on average every 3000 slots with instant increase of $2048 \times 0.8 = 1638$ bytes to a buffer. As Figure 7a shows, bursty traffic has very limited impact on our PMP. For NGP, StickyAFP and ERR, the bandwidth efficiency gets improved, since bursty traffic helps fill those unfilled payloads. Note that, in Figure 7b, the delay of NGP increases sharply and remains the highest for all traffic ratios. The reason is that NGP does not implement an overflow bit to handle sudden traffic burst. Due to the lack of overflow indication, NGP is unable to properly adapt to bulky data arrivals. This phenomenon is especially serious when traffic rates are low, which implies that NGP keeps the infrequent polling patterns without realizing that bursty traffic has occurred, thus resulting in long delays.

4.3. Multiple master–slave pairs without bursty traffic

In the following experiments, we enlarge the piconet by including more slaves. Under such a situation, the low efficiency of one master–slave pair may deprive the chances of other pairs using the resource (i.e. slots), which is more likely to bring the network to the saturated level. Thus, slots should be used more cautiously. We simulate seven slaves in a piconet. The arrival rates of the seven master–slave pairs are denoted as $\lambda_{H1}/\lambda_{L1}$, $\lambda_{H2}/\lambda_{L2}, \ldots,$ and $\lambda_{H7}/\lambda_{L7}$. To add heterogeneity, we let $\lambda_{H1}/\lambda_{L1} = 2$, $\lambda_{H2}/\lambda_{L2} = 4$, $\lambda_{H3}/\lambda_{L3} = 6$, $\lambda_{H4}/\lambda_{L4} = 8$, $\lambda_{H5}/\lambda_{L5} = 10$, $\lambda_{H6}/\lambda_{L6} = 12$ and $\lambda_{H7}/\lambda_{L7} = 14$. The total piconet traffic load λ is the sum of these rates.

Figure 8 plots the piconet throughput and average delay against various total loads λ. We observe that the throughput of PMP saturates at the highest level compared to the other approaches. This is because PMP utilizes bandwidth more efficiently, thus saving more bandwidth space to accommodate more traffic. In other words, the proposed PMP effectively reduces unnecessary bandwidth waste, which improves piconet throughput. For the cases of $K = 3$ and $K = 5$, the differences are almost indistinguishable. This further confirms that a simple/short pattern length is sufficient to achieve very good performance. Note that after the saturation points, a lot of data bytes may be dropped. However, the delays of dropped bytes are not taken into account. This is why we do not see significant increase in delays in Figure 8 after the network is saturated.

4.4. Multiple master–slave pairs with bursty traffic

Again, we add bursty traffic to the regular Poisson traffic for each master–slave pair. As Figure 9 illustrates, the PMP saturates at the highest throughputs with the lowest packet delays.

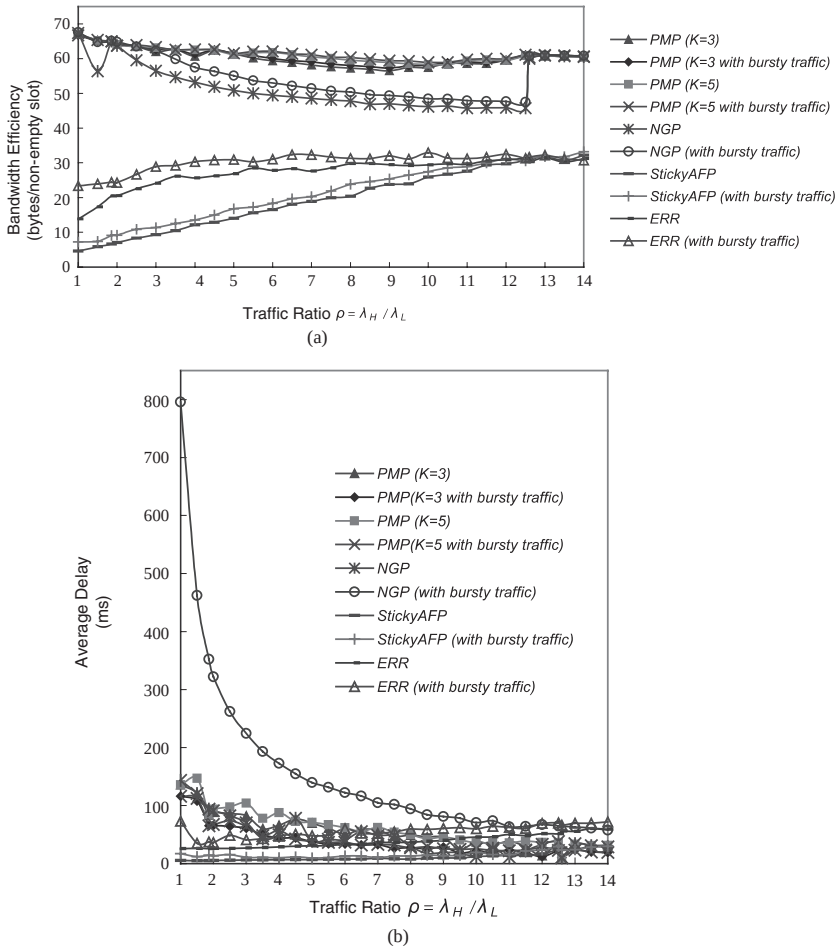

FIGURE 7. Effect of bursty traffic when there is one master–slave pair: (a) bandwidth efficiency and (b) average delay.

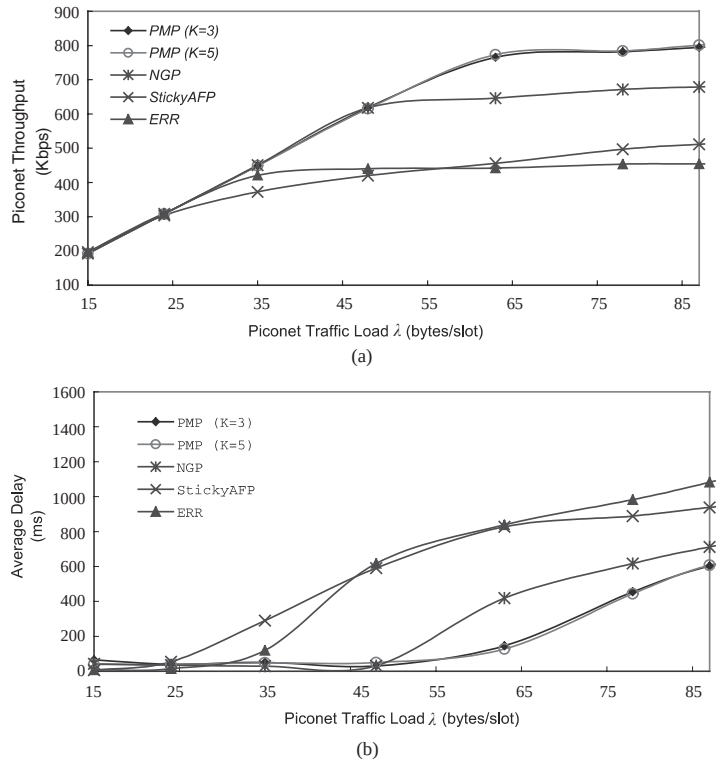

FIGURE 8. Piconet performance when there are multiple master–slave pairs: (a) throughput and (b) average delay.

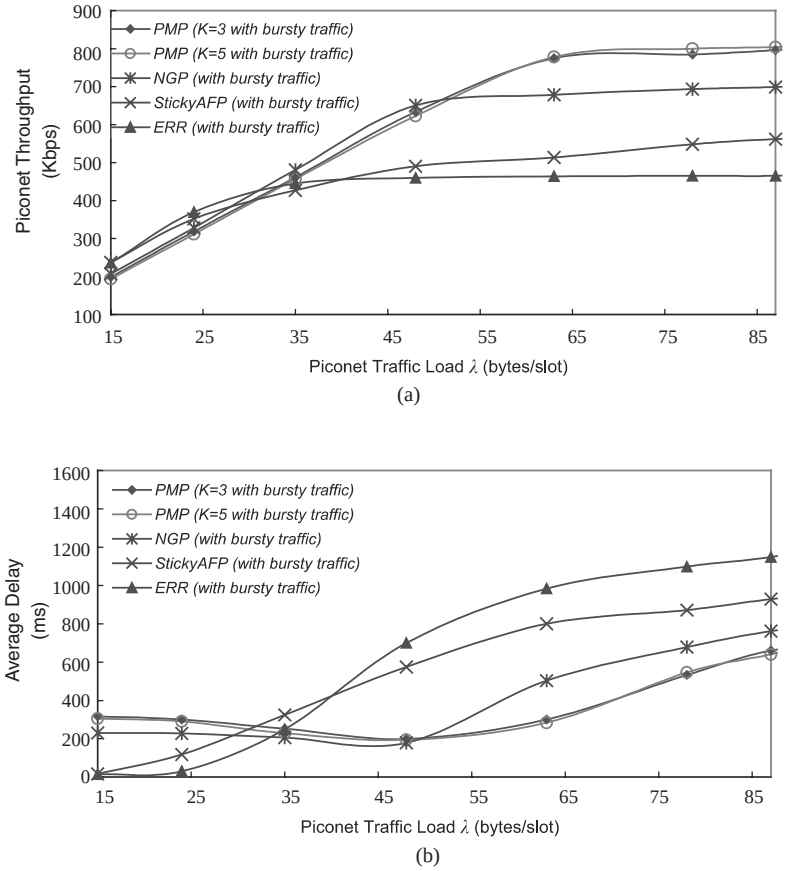

FIGURE 9. Effect of bursty traffic when there are multiple master–slave pairs: (a) throughput and (b) average delay.

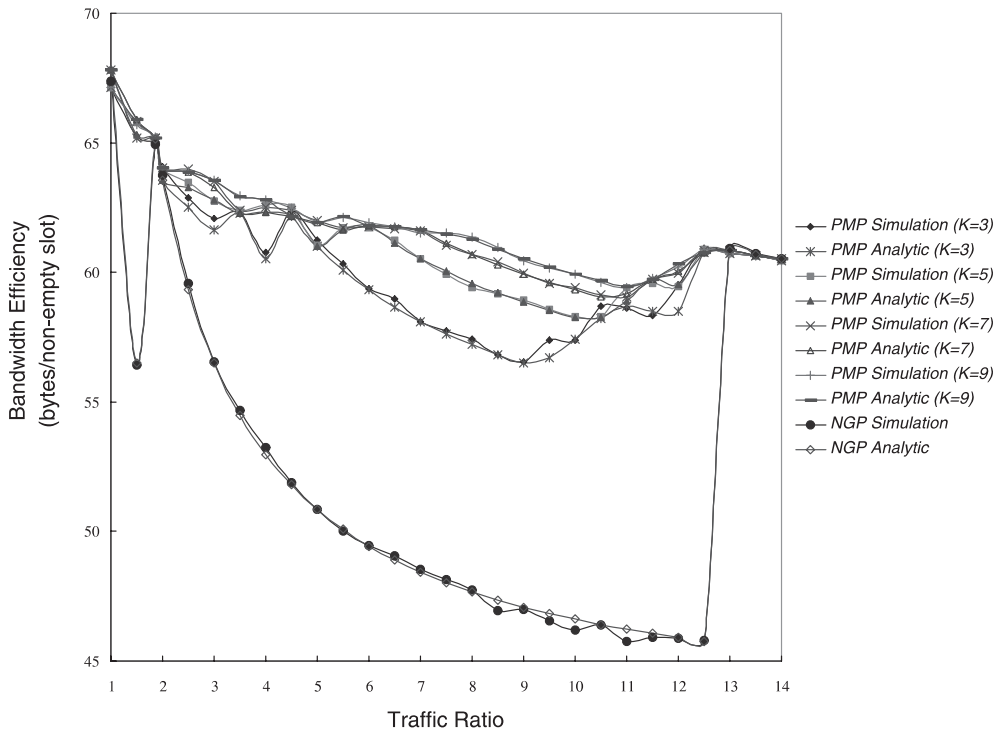

FIGURE 10. Comparison of simulation results and analytic values.

4.5. Comparison of simulation and analytic results

Recall that analytic predictions have been derived in Equations (1) and (3). In Figure 10, we compare these analytic results against simulation results, under a single master–slave pair, for PMP ($K = 3, 5, 7, 9$) and NGP. Note that it is infeasible to do this for bursty traffics. The result verifies the consistency of our previous analyses with simulations.

5. CONCLUSIONS

In this paper, we have proposed an efficient PMP policy for ACL connections in a Bluetooth piconet. For each master–slave pair, by estimating both sides' packet arrival rates, the master judiciously selects a polling pattern that can best utilize the network bandwidth. Based on the selected pattern, the master then polls the slave with proper packet types at proper time slots. In return, the slave also replies with proper packet types. The ultimate goal is to reduce the number of NULL packets and unfilled payloads so as to increase bandwidth efficiency. The PMP policy has properly addressed the asymmetry of up- and down-link traffics and the available packet types in Bluetooth. Another merit of PMP is its simplicity—a pattern length of $K = 3$ or 4 can already perform very well. So the computational complexity can be kept low. Simulation experiments have demonstrated that the proposed PMP policy improves bandwidth efficiency and network throughput at the expense of moderate packet delays, compared with other polling approaches. In our discussion, only DH1/3/5 are considered. To include DM1/3/5, we propose to estimate the packet error probability. Whenever the probability is below a threshold, we will adopt DH1/3/5; otherwise, we will switch to DM1/3/5, and the derivation of polling patterns is similar.

In our current model, traffic is simulated by byte arrival, not packet arrival. So delay is computed based on bytes, not packets. Since we do not make explicit upper-layer traffic behavior, we were unable to translate from byte to packet delay. In order to provide further insight about the packet delay, higher level traffic behavior must be modeled, and this may be directed to future work.

ACKNOWLEDGEMENTS

This work is co-sponsored by the Lee and MTI Center for Networking Research at the National Chiao-Tung University and the MOE Program for Promoting Academic Excellence of Universities under grant numbers A-91-H-FA07-1-4 and 89-E-FA04-1-4.

REFERENCES

[1] Bluetooth Specification v1.1 (2001) http://www.bluetooth.com. February.

[2] Groten, D. and Schmidt, J. (2001) Bluetooth-based mobile ad hoc networks: Opportunities and challenges for a telecommunications operator. *IEEE Vehicular Technology Conference (VTC)*, 1134–1138.

[3] Johansson, P., Kazantzidis, M., Kapoor, R. and Gerla, M. (2001) Bluetooth: An enabler for personal area networking. *IEEE Network*, September/October, 28–37.

[4] Kalia, M., Garg, S. and Shorey, R. (2000) Scatternet structure and inter-piconet communication in the Bluetooth system. *IEEE National Conference on Communications*, New Delhi, India.

[5] Miklos, G., Racz, A., Turanyi, Z., Valko, A. and Johansson, P. (2000) Performance Aspects of Bluetooth Scatternet Formation. *ACM Int. Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, pp. 147–148.

[6] Salonidis, T., Bhagwat, P. and Tassiulas, L. (2000) Proximity awareness and fast connection establishment in Bluetooth. *ACM Int. Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, pp. 141–142.

[7] Law, C., Mehta, A. K. and Siu, K.-Y. (2001) Performance of a new Bluetooth scatternet formation protocol. *ACM Int. Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, pp. 183–192.

[8] Ramachandran, L., Kapoor, M., Sarkar, A. and Aggarwal, A. (2000) Clustering Algorithms for Wireless Ad Hoc Networks. *ACM DIAL M Workshop*, pp. 54–63.

[9] Salonidis, T., Bhagwat, P., Tassiulas, L. and LaMaire, R. (2001) Distributed topology construction of Bluetooth personal area networks. *IEEE INFOCOM*, pp. 1577–1586.

[10] Zaruba, G. V., Basagni, S. and Chlamtac, I. (2001) Bluetrees—Scatternet formation to enable Bluetooth-based ad hoc networks. *IEEE Int.'l Conference on Communications (ICC)*, pp. 273–277.

[11] Lee, T.-J., Jang, K., Kang, H. and Park, J. (2001) Model and performance evaluation of a piconet for point-to-multipoint communications in Bluetooth. *IEEE Vehicular Technology Conference (VTC)*, pp. 1144–1148.

[12] Sangvornvetphan, V. and Erke, T. (2001) Traffic scheduling in Bluetooth Network. *IEEE Int.'l Conference on Networks (ICON)*, pp. 355–359.

[13] Capone, A., Gerla, M. and Kapoor, R. (2001) Efficient polling schemes for Bluetooth picocells. *IEEE Int. Conference on Communications (ICC)*, pp. 1990–1994.

[14] Chakrabory, I., Kashyap, A., Kumar, A., Rastogi, A., Saran, H. and Shorey, R. (2001) MAC scheduling policies with reduced power consumption and bounded packet delays for centrally controlled TDD wireless networks. *IEEE Int. Conference on Communications (ICC)*, pp. 11–14.

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

[16] Kalia, M., Bansal, D. and Shorey, R. (1999) MAC Scheduling and SAR policies for Bluetooth: A master driven TDD pico-cellular wireless system. *IEEE Int. Workshop on Mobile Multimedia Communications (MoMuC)*, pp. 15–17.

[17] Kalia, M., Bansal, D. and Shorey, R. (2000) Data scheduling and SAR for Bluetooth MAC. *IEEE Vehicular Technology Conference (VTC)*, pp. 716–720.