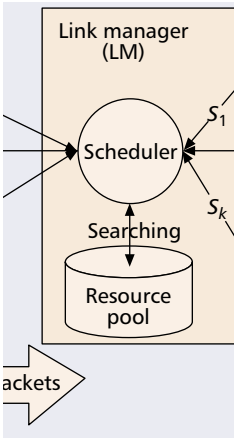
AN ADAPTIVE SNIFF SCHEDULING SCHEME FOR POWER SAVING IN BLUETOOTH

TING-YU LIN AND YU-CHEE TSENG, NATIONAL CHIAO-TUNG UNIVERSITY



Bluetooth is expected to be an important basic constructing component for Smart Homes. In a Smart Home environment, many devices will be portable and battery-operated, making power saving an essential issue. The authors study the problem of managing the low-power sniff mode in Bluetooth.

ABSTRACT

Bluetooth is expected to be an important basic constructing component of smart homes. In a smart home environment, many devices will be portable and battery-operated, making *power saving* an essential issue. In this article we study the problem of managing the low-power *sniff mode* in Bluetooth, where a slave is allowed to be awake only periodically. One challenging problem is how to schedule each slave's sniffing period in a piconet so as to resolve the trade-off between traffic and power-saving requirements, to which we refer as the *sniff-scheduling problem*. We propose an adaptive protocol to dynamically adjust each slave's sniff parameters, with a goal of catching the varying, and even asymmetric, traffic patterns among the master and slaves. Compared to existing works, our work is unique. First, our scheduling considers multiple slaves simultaneously. Existing work only considers one slave, and different slaves are treated independently. Second, our scheduling is more accurate and dynamic in determining the sniff-related parameters based on slaves' traffic patterns. Most work is restricted to a naive exponential adjustment in sniff interval/sniff-attempt window. Third, our proposal includes the placement of sniff-attempt periods of sniffed slaves on the time axis when multiple slaves are involved. This issue is ignored by earlier work. Extensive simulation results are presented. Among many observations, one interesting result is that with proper settings, our protocol can save significant power while achieving higher network throughput than a naive always active round-robin scheme.

INTRODUCTION

Computing and communication anytime, anywhere is a global trend in development today. Ubiquitous computing has been made possible by the advance of wireless communication technology and the availability of many lightweight compact portable computing devices. This area has attracted a lot of attention recently, and various types of network architectures have been proposed, such as wireless LANs, ad hoc networks, sensor networks, and personal area networks.

One emerging environment gaining more and more attention is the *smart home*. The basic idea behind smart homes is to provide various human-friendly services with the goal of facilitating human life. Typical home electronic appliances will not be considered clumsy anymore. Instead, they are capable of coordinating with each other and adapting to surroundings. Such capabilities are achieved by equipping these appliances with embedded computing and communication devices. There are diverse aspects and technologies involved in the development of Smart Homes. One promising technology supported by numerous organizations and companies is Bluetooth. With the design goals of compactness, low cost, and low power, Bluetooth is expected to be a promising basic constructing component of smart homes.

This article focuses on Bluetooth [1], which is characterized by indoor low-power low-complexity short-range radio wireless communications with a frequency-hopping time-division duplex channel model. A master-slaves configuration called a *piconet* is adopted. Readers can refer to [2–4] for more general details of the Bluetooth standard description.

Since low cost is one design goal of Bluetooth, a large number of widespread deployments of Bluetooth are expected. Within home environments, the deployment could consist of various portable devices. One essential issue for almost all kinds of portable devices is *power saving*. Mobile devices have to be supported by batteries, and without power they become useless. Battery power is a limited resource, and it is expected that battery technology is not likely to progress as fast as computing and communication technologies do. Hence, lengthening the lifetime of batteries in portable devices is an important issue. Solutions for power saving can be generally categorized into several approaches.

Transmission power control: In wireless communication, transmission power has a strong impact on bit error rate, transmission rate, and inter-radio interference. These are typically contradicting factors. Power control to reduce interference for ad hoc networks is addressed in [5]. Dynamically adjusting transmission powers of mobile hosts in ad hoc networks to control network topology, known as *topology control*, is addressed in [6]. Increasing network throughput by power adjustment for packet radio networks is addressed in [7].

1536-1284/02/$17.00 © 2002 IEEE

Power-aware routing: Power-aware routing protocols for ad hoc networks are discussed in [8, 9].

Management of low-power modes: More and more wireless devices can provide low-power modes. IEEE 802.11 has a power-saving mode in which a radio only needs to be awake periodically. HiperLAN allows the mobile host, which is in power-saving mode, to define its own active period [10]. As for active hosts, they can save power by turning off their equalizers according to the transmission bit rate. Bluetooth provides three low-power modes: *sniff*, *hold*, and *park* [1].

We study the management of low-power sniff mode in Bluetooth to conserve power; thus, this falls into the third category above. In sniff mode, a slave's listening activity is reduced. Slaves only listen in specified time slots regularly spaced by sniff intervals. One challenging problem is how to schedule each slave's sniffing period in a piconet to balance the trade-off between traffic and power-saving requirements, to which we refer as the *sniff scheduling problem*.

In this article, an adaptive sniff scheduling protocol is proposed to periodically adjust the sniff parameters. An *evaluator* is used by each master and its slaves to determine its traffic pattern and sniff-related parameters. A *scheduler* is deployed on the master's side to schedule each slave's sniffing period. Since each slave's sniffing period can be regarded as an infinite sequence of time slots and multiple slaves are considered in this article, we propose a concept called *resource pool* (RP) to manage the available/occupied time slots in the piconet. The master periodically checks the needs of its slaves by running the evaluator, and allocates suitable slot resources from the RP for them. On the other hand, slaves can exercise their own evaluators and issue requests for slot resources as well. Two scheduler policies are proposed: *Longest Sniff Interval First* (LSIF) and *Shortest Sniff Interval First* (SSIF). Simulation results are presented to verify the effectiveness of our protocol.

Compared to existing work, our work is unique in the following senses. First, our scheduling scheme considers multiple slaves simultaneously. Existing work only considers one slave, and different slaves are treated independently. Second, our scheduling is more accurate and dynamic in determining the sniff-related parameters based on slaves' traffic patterns. Most work is restricted to a naive exponential adjustment in sniff interval/sniff-attempt window. Third, our proposal includes the placement of sniff-attempt periods of sniffed slaves on the time axis when multiple slaves are involved. This issue is ignored by earlier work.

Related work includes [11–14]. In [12, 14], the polling priorities of slaves are determined based on their traffic loads; however, how to combine this with the sniff mode is not addressed. In [13] it is proposed to dynamically adjust the sniff parameters according to a slave's slot utilization. In [11] a learning function is proposed to determine the sniff interval. However, the sniff interval is adjusted on a per-interval basis, and thus the overhead of control messages might be pretty high. Both [11, 13] suffer the problems that only one single slave is considered in an independent way, and the placement of sniff-attempt windows is not addressed.

The rest of this article is organized as follows. Preliminaries are given in the next section, followed by our sniff scheduling protocol. We propose two policies for our scheduler. Simulation results are then provided. Finally, we draw conclusions.

PRELIMINARIES

BLUETOOTH TECHNOLOGY REVIEW

Bluetooth is a master-driven time-division duplex short-range radio wireless system. The smallest network unit is called a *piconet*, and has a master-slaves configuration. A time slot in Bluetooth is 625 μs. The master sends data to slaves in even-numbered slots, while slaves send data to the master in odd-numbered slots. A slave only transmits packets after the master polls or sends data to it.

According to the Bluetooth protocol stack [1], on top of RF is the Bluetooth baseband, which controls use of the radio. Four important operational modes are supported by the baseband: *active*, *sniff*, *hold*, and *park*. Active mode is most energy-consuming, where a Bluetooth unit is turned on most of the time. Sniff mode allows a slave to go to sleep and only wake up at a specific time. In hold mode, a slave can temporarily suspend supporting data packets on the current channel; the capacity can be made free for other things, such as scanning, paging, and inquiring. While in hold mode, a unit can also attend other piconets. Prior to entering hold mode, an agreement should be reached between the master and slave on hold duration. When a slave does not want to participate in the piconet channel, but still wants to remain synchronized, it can enter park mode. The parked slave has to wake up regularly to listen to the channel, to stay synchronized or check broadcast messages.

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 Link Manager Protocol (LMP). The Logical Link Control and Adaptation Protocol (L2CAP) provides connection-oriented and connectionless datagram services to upper-layer protocols. Two major functionalities of L2CAP are protocol multiplexing, and segmentation and reassembly (SAR).

The Service Discovery Protocol (SDP) defines the means for users to discover which services are available in their neighborhood and the characteristics of these services. The RFCOMM protocol provides emulation of serial ports over L2CAP to support many legacy applications based on serial ports over Bluetooth without any modifications. Up to 60 serial ports can be emulated.

THE LOW-POWER SNIFF MODE

Our main focus, the power-saving issue of Bluetooth, is discussed in more detail in this section. In active mode, the Bluetooth unit is turned on most of the time to participate in send/receive activities. An active slave listens in even slots

According to the Bluetooth protocol stack, on top of RF is the Bluetooth Baseband, which controls the use of the radio. Four important operational modes are supported by the baseband: active, sniff, hold, and park.

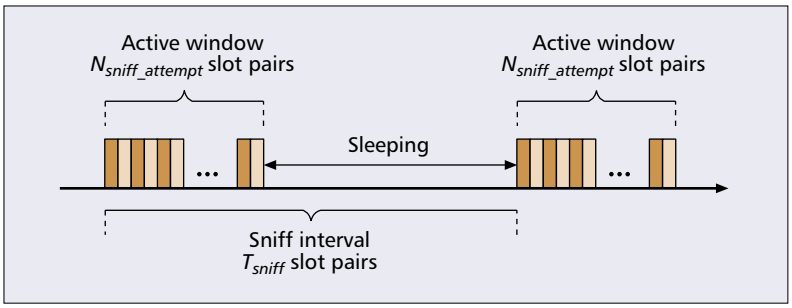


Figure 1. *Sniff interval and active window of Bluetooth (darkened parts are even slots).*

for packets. If the slave is not addressed in the current packet, it may sleep until the next even slot the master transmits. From the type indication of the packet, the slave can derive the number of slots to be used by the master to transmit the current packet. The addressed slave will reply in the next odd slot after the master's transmission.

In the sniff mode, the slave's listening activities are reduced to save energy. For a sniffed slave, the time slots when the master can communicate to that slave are limited to some specific time slots. These so-called *sniff-attempt slots* arrive periodically, as illustrated in Fig. 1. In the Bluetooth specification, there are three parameters specified for such sniff activity: T_{sniff}, $N_{sniff\_attempt}$, and $N_{sniff\_timeout}$. Since Bluetooth specification separates even and odd slots for the master and slave transmissions, the values of these sniff parameters are all based on *slot pairs* (one even plus one odd slot). In every T_{sniff} slot pair, the slave will wake up to listen to the master for $N_{sniff\_attempt}$ consecutive (even) slots for possible packets destined to it. After every reception of a packet with a matching address, the slave continues listening for $N_{sniff\_timeout}$ more slots or for the remaining of the $N_{sniff\_attempt}$ slot pairs, whichever is greater. In this article, we call T_{sniff} the *sniff interval* and $N_{sniff\_attempt}$ the *active window*.

The control packets exchanged between two communicating LMs via LMP are termed LMP_PDUs. There are four LMP_PDUs involved in sniff mode management: LMP_sniff_req, LMP_accepted, LMP_not_accepted, and LMP_unsniff_req. These PDUs are for making/rejecting/accepting requests and returning to normal active mode. To enter sniff mode, an LMP_sniff_req request packet can be initiated by either a master or a slave carrying the proposed parameters. Upon receipt of the request, the receiver side can negotiate with the other side on the related sniff parameters by issuing another LMP_sniff_req request packet carrying the suggested parameters. If an agreement can be seen, an LMP_accepted packet is used to place the slave into sniff mode. Otherwise, an LMP_not_accepted packet is returned with a reason code for rejection. Also, note that the sniff parameters can be negotiated on a per-master-slave basis.

Sniff mode can be ended by sending an LMP_unsniff_req packet. The counterpart must reply with an LMP_accepted packet. If this is requested by the slave, it will enter active mode after receiving LMP_accepted. If this is requested by the master, the slave will enter active mode immediately after receiving LMP_unsniff_req.

Problem Statement

Although the operations of sniff mode are given in the Bluetooth specification, how to determine the sniff-related parameters is left as an open issue for the designers. One should dynamically determine a proper set of sniff parameters for a slave based on its traffic pattern to save as much of its power as possible without incurring too much delay in packet delivery. Furthermore, multiple slaves could be in sniff mode at the same time. How to schedule their active windows on the time axis is a challenging problem since these windows arrive periodically and may extend, conceptually, to infinity on the time axis. Overlapping of active windows is allowed but undesirable. Collectively, we call this the *sniff scheduling problem*.

Finally, we are aware of the possibility of using hold or park mode for power saving. However, this article only considers sniff mode because it can serve various types of traffic with power saving in mind.

The Sniff Scheduling Protocol

In this section we propose a protocol to exploit the low-power sniff mode of Bluetooth. Because of the master-driven centrally controlled architecture of Bluetooth, we will maintain an RP on the master's side. The sniff parameters of each slave will be adjusted dynamically based on many factors such as the slave's traffic load, current backlog, previous utilization of sniff slots, and availability of the RP. The ultimate goal is to save slaves' power while keeping packet delays as small as possible. Note that because of Bluetooth's separation of even and odd slots, all calculations refer to slot pairs unless stated otherwise.

Figure 2 shows the architecture of our sniff scheduling protocol in a piconet with K active slaves, $1 \le K \le 7$. On the master side, there are three main entities: evaluator, scheduler, and RP. The master periodically runs the evaluator for each slave k, $1 \le k \le K$, to evaluate its condition. If necessary, a value S_k, which is used to reflect the estimated traffic load of slave k, is generated and fed into the scheduler to readjust this slave's sniff parameters. The scheduler then searches the RP to schedule a new set of sniff parameters for the slave.

On the other hand, slaves also run their own evaluators periodically. These distributed evaluators will pass their desired S_k values to the master via an LMP_sniff_req packet. The scheduler then searches the RP, and arranges new sniff parameters for them. In our protocol, when the scheduler is unable to find suitable sniff parameters for a slave, an LMP_unsniff_req packet will be issued to invite it into active mode. This usually happens when the traffic load of the slave is quite large. When the master finds it possible to allocate a proper sniff scheduling for the slave, it may bring the slave back to sniff mode again.

Since the low-power modes of Bluetooth are

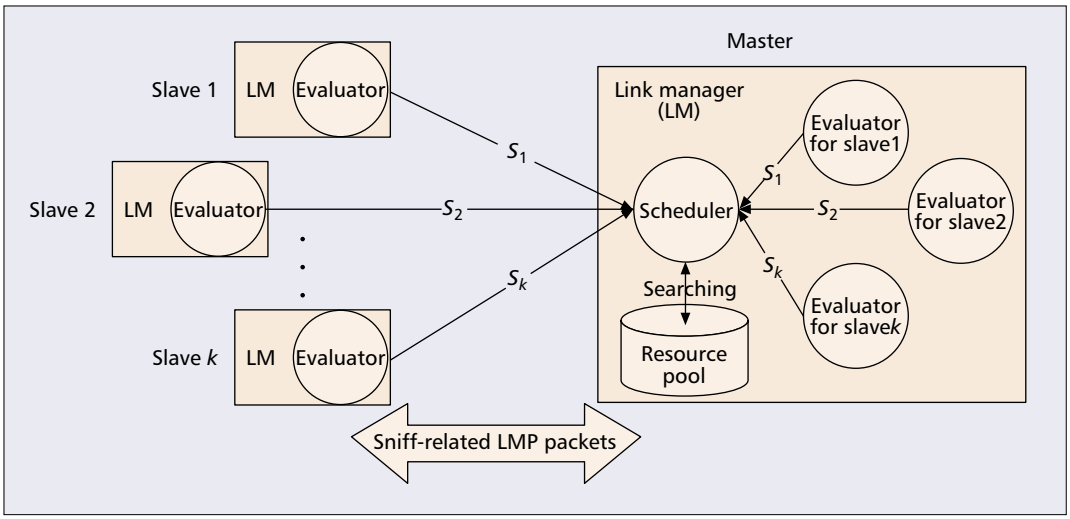


Figure 2. *The architecture of our sniff scheduling protocol.*

Since the low-power modes of Bluetooth are managed by the link manager, our protocol should reside in the LM layer of each Bluetooth unit, monitoring the backlogs of the lower baseband buffers and issuing proper sniff-scheduling packets.

managed by the link manager (LM), our protocol should reside in the LM layer of each Bluetooth unit, monitoring the backlogs of the lower baseband buffers and issuing proper sniff scheduling packets. In this article we follow the same assumption as in [12]: the master keeps separate buffers in the baseband layer for its slaves. Each buffer queues the data dedicated to the corresponding slave. Below, we discuss our design in more detail.

The Evaluator

The purpose of the evaluator is to measure how efficiently or inefficiently a slave uses the sniff-attempt slots assigned to it and, if necessary, to trigger the scheduler to readjust its sniff parameters.

The fundamental parameters are explained below. First, we define (T_k, N_k, O_k) as the current sniff parameters (sniff interval, active window size, and offset, respectively) associated with slave k. For each slave k, we have to measure its U_k. This is a ratio between (including) 0 and 1, indicating how many slot pairs of the sniff-attempt slots are used effectively for real data communication under the current setting for slave k. Also, we use B_k to denote the buffer backlog for slave k (i.e., the number of packets currently queued in the local baseband buffer). By U_k and B_k, we derive a weighted value W_k to measure the current requirement of slave k:

$$W_k = \alpha\, U_k + (1 - \alpha) B_k / B_{max}, \qquad (1)$$

where B_{max} is the maximum buffer space, and α is a constant between 0 and 1 to differentiate the importance of U_k and B_k. The resulting W_k will be tested against the condition $r_{lb} < W_k < r_{ub}$, where r_{lb} and r_{ub} are predefined tolerable lower bound and upper bound, respectively, of W_k. If this condition is violated, the master will be triggered to readjust slave k's current sniff parameters; otherwise, no readjustment is needed.

Based on W_k, our objective is to determine the desired slot occupancy S_k of slave k, which represents the expected ratio of the new N_k to the new T_k:

$$S_k = (N_k/T_k) \times W_k/\delta. \qquad (2)$$

Intuitively, N_k/T_k is slave k's current slot occupancy. Multiplying this ratio by W_k gives the slot occupancy ratio expected to be assigned to this slave. The factor δ is a positive constant below 1 to enlarge the expected ratio to tolerate a certain level of inaccuracy in our estimation.

Note that a minor logic flaw we intentionally omit in the above discussion (for ease of presentation) is that when the slave is in active mode, it has no sniff parameters, so the ratio N_k/T_k becomes meaningless. In this case we simply replace this ratio by the recent slot occupancy of slave k (with all slots as the denominator). The rest is all the same.

The Resource Pool

The available sniff-attempt slots that can be allocated to slaves are managed by the RP on the master side. One may regard the sniff-attempt slots of a slave as a periodical infinite sequence. However, we need an efficient finite data structure for representation.

In this section we propose to use a two-dimensional $d_1 \times d_2$ matrix M for the representation. The basic idea is as follows. We group (infinite) slot pairs appearing with period $d_1 \cdot d_2$ into one set. For $p = 0..d_1 \cdot d_2 - 1$, define

$$G_p = \{p + q \cdot d_1 \cdot d_2 \mid q \text{ is any non-negative integer}\}.$$

Each entry in matrix M is used to represent the availability of one such slot group, so we define, for $i = 0..d_1 - 1, j = 0..d_2 - 1$,

$$M[i,j] = \begin{cases} 0 \text{ if } G_{i \cdot d_2 + j} \text{ is free} \\ 1 \text{ if } G_{i \cdot d_2 + j} \text{ is busy} \end{cases}. \qquad (3)$$

Note that variables d_1 and d_2 are adjustable parameters. The value of $d_1 \cdot d_2$ should be large enough to capture the behavior of those slaves that have very low traffic load and would like to spend very low energy on sniff-attempts. Also note that $d_1 \cdot d_2$ indicates the maximum allowed sniff interval. To facilitate exponential adjustment of sniff intervals, we configure d_1 to be power of 2, denoted 2^u, where u is an non-negative integer. Besides, d_2 is replaced with T, which indicates the minimum allowable sniff interval.

Different values of u and T will provide different levels of flexibility, as shown later. Table 1 plots several examples of M with size 8×15.

The 2D matrix M can provide us much flexibility in managing periodical time slots. We can manipulate both sniff intervals and active windows of sniffing slaves easily. Two groups that are adjacent in M can be framed together to double the active window. Two groups spaced by a certain distance in the matrix can be grouped together too to divide the sniff interval by half. This can be extended to the combination of more groups easily. Reversibly, we may decrease the active window or increase the sniff interval of a slave to reduce its power consumption by partitioning groups. For example, given the state in Table 1a, if a slave's packet mean arrival rate is 16/120, we show four possible ways (b, c, d, e) to arrange the slave's sniff-attempt slots. Table 1b indicates that the slave is awake every 120 slot pairs, each time lasting for 16 slot pairs. Table 3c shows that it is awake every 60 slot pairs, each time lasting for 8 slot pairs. Table 1d means it is awake every 30 slot pairs, each time lasting for 4 slot pairs. Table 1e means it is awake every 15 slots pairs, each time lasting for 2 slot pairs.

With such a formulation, the resource management problem becomes one of allocating proper entries in matrix M. Later, we will propose two searching policies for this problem.

Row 7 1 0 0 0 0 1 1 1 1 1 1 0 0 0 0
Row 6 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1
Row 5 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0
Row 4 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
Row 3 1 0 0 0 0 1 1 1 1 1 1 0 0 0 0
Row 2 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1
Row 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0
Row 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 Column
(a)

Row 7 1 0 0 0 0 1 1 1 1 1 1 0 0 0 0
Row 6 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1
Row 5 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0
Row 4 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
Row 3 1 0 0 0 0 1 1 1 1 1 1 0 0 0 0
Row 2 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1
Row 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 0
Row 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 Column
(b)

Row 7 1 0 0 0 0 1 1 1 1 1 1 0 0 0 0
Row 6 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1
Row 5 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0
Row 4 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
Row 3 1 0 0 0 0 1 1 1 1 1 1 0 0 0 0
Row 2 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1
Row 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0
Row 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 Column
(c)

Row 7 1 0 0 0 0 1 1 1 1 1 1 0 0 0 0
Row 6 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1
Row 5 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0
Row 4 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
Row 3 1 0 0 0 0 1 1 1 1 1 1 0 0 0 0
Row 2 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1
Row 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0
Row 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 Column
(d)

Row 7 1 0 0 1 1 1 1 1 1 1 1 0 0 0 0
Row 6 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1
Row 5 0 0 0 1 1 0 0 0 1 1 1 1 1 1 0
Row 4 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
Row 3 1 0 0 1 1 1 1 1 1 1 1 0 0 0 0
Row 2 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1
Row 1 0 0 0 1 1 0 0 0 1 1 1 1 1 1 0
Row 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 Column
(e)

Table 1. *Example: Initial state of an 8×15 matrix* M *and feasible assignments in matrix* M *(dark) for a slot occupancy of 16/120.*

LMP_PDU Flows

In the Bluetooth specification, both master and slaves can initiate a sniffing request. In our protocol, we also allow a sniffing request to be master- or slave-activated. This section discusses the related LMP_PDU exchanges. Recall the calculation of W_k for slave k. All the following discussion is triggered by violating the constraint $r_{lb} < W_k < r_{ub}$.

Figure 3 shows four possible master-activated scenarios. The first one demonstrates the master proposing a new set of sniff parameters (O'_k, T'_k, N'_k) for slave k. In response, the slave runs its evaluator to determine its local S_k. If the assigned slot occupancy derived from N'_k/T'_k is not less than S_k, an LMP_accepted can be returned.

The second scenario demonstrates that slave k disagrees on the assigned parameter, so an LMP_sniff_req is returned. However, note that since slave k does not know the status of the RP, we actually intend to return its estimated S_k to the scheduler for an arrangement. Since S_k is a ratio between 0 and 1, we simply use the two fields T_{sniff} and $N_{sniff\_attempt}$ in LMP_sniff_req to carry two values, T''_k and N''_k, respectively, such that $N'''_k/T''_k \approx S_k$. The scheduler then tries to allocate a new set of sniff parameters based on N''_k/T''_k for slave k. In response, the slave issues an LMP_accepted.

The third scenario is similar to the second one, but the master fails to allocate a large enough active window from its RP (probably because matrix M is too crowded or fragmented). In this case, the master will request the slave to unsniff itself.

The fourth scenario is where, from the estimated S_k, the master directly finds it has difficulty allocating a large enough active window from its RP, so an unsniff request is directly sent to the slave.

Figure 4 shows slave-activated negotiation. This is similar to the aforementioned second and third scenarios, but it is triggered by finding violation of the condition $r_{lb} < W_k < r_{ub}$ on the slave's side. Again, since the slave does not know

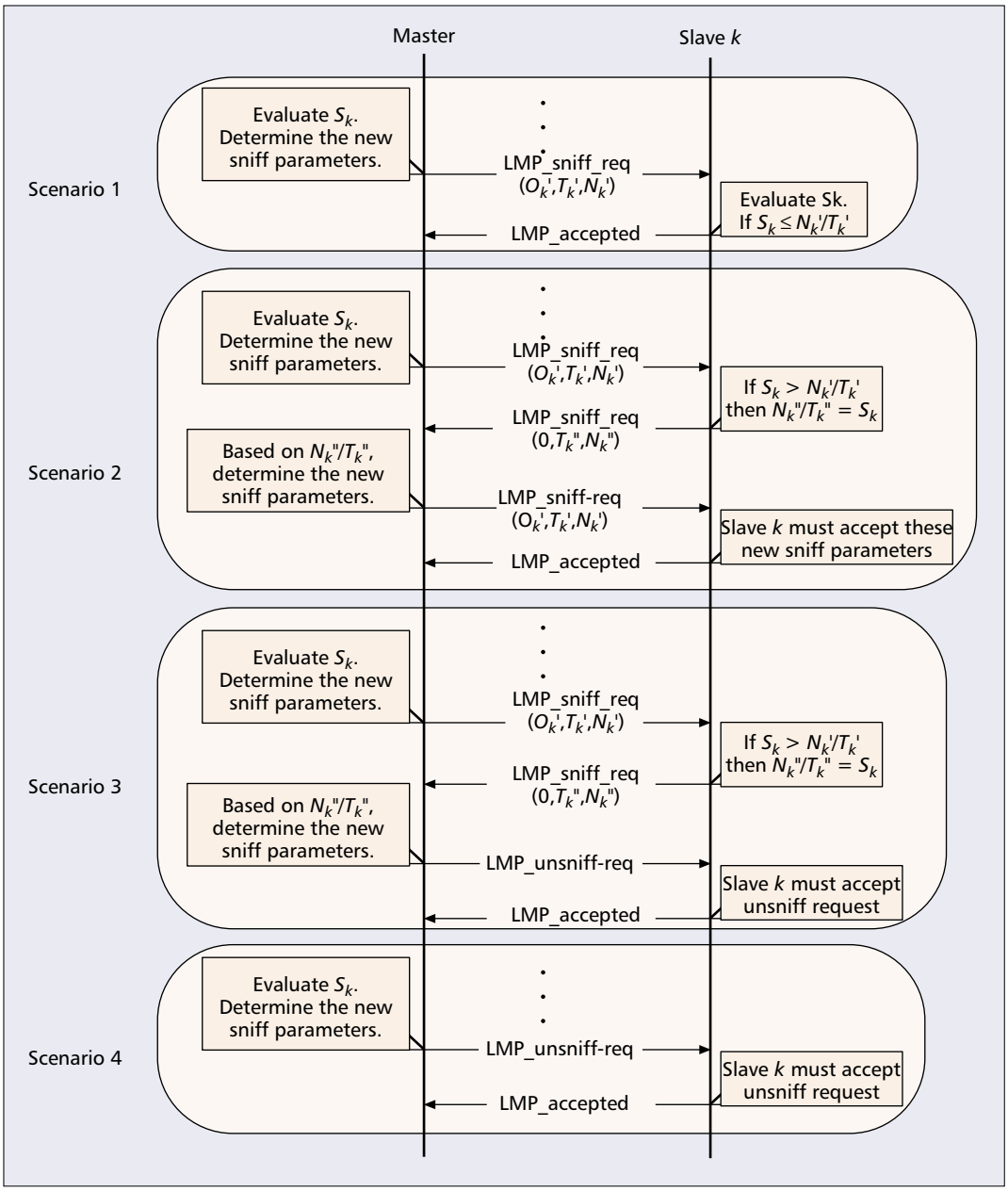


Figure 3. *Scenarios of master-activated sniff parameter negotiation.*

the status of the RP, we use the two fields T_{sniff} and $N_{sniff\_attempt}$ in LMP_sniff_req to convey the desired S_k to the master. It then follows with an LMP_sniff_req containing the new sniff parameters or an LMP_unsniff_req packet, depending on the crowdedness of the RP.

Scheduling Policies for the Resource Pool

The job of the scheduler is to take an input S_k and determine a suitable set of sniff parameters (denoted O_k', T_k', and N_k' below) for slave k. These parameters in fact represent a set of slot groups in matrix M. Before making the allocation, the old occupancy by this slave on M should be released, which is an easy job. In the following, we propose two policies for searching M, named *LSIF* and *SSIF*.

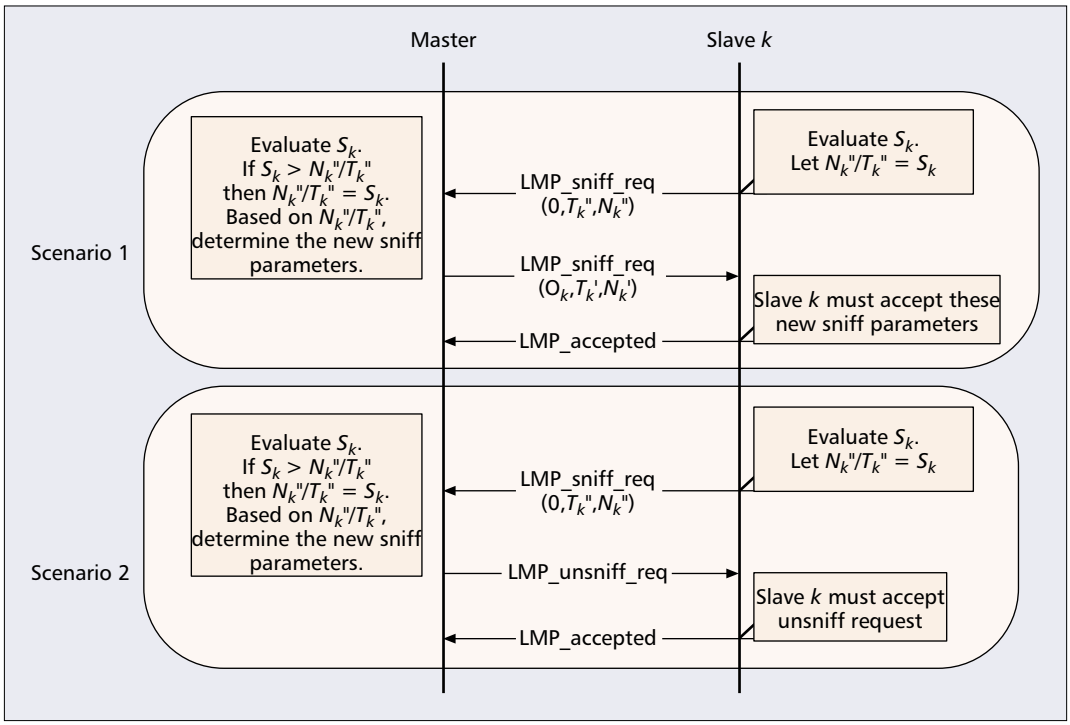
Longest Sniff Interval First

In the LSIF policy, we search matrix M starting from the longest sniff interval, which is $2^u \cdot T$. If the search fails, we divide the interval by 2, which is $2^{u-1} \cdot T$, and do the search again. If the search also fails, we further use the interval $2^{u-2} \cdot T$ to do the search. The search stops once a satisfactory set of slot groups is found. This is repeated until the shortest interval T is tried, in which case we will bring the slave into active mode, as discussed earlier.

Below, we discuss the detailed steps when we search matrix M with a sniff interval $2^p \cdot T$, where $0 \le p \le u$. Since the matrix is of size $2^u \times T$, we will "fold" M into a smaller matrix of size $2^p \times T$. Specifically, we partition M horizontally and evenly into 2^{u-p} pieces, each of size $2^p \times T$. Then we fold all pieces together by executing a bitwise OR operator. The meaning of OR is to

The available sniff-attempt slots that can be allocated to slaves are managed by the resource pool at the master side. One may regard the sniff-attempt slots of a slave as a periodical, infinite sequence. However, we need an efficient, finite data structure for the representation.

We have developed a simulator to verify the effectiveness of our protocol. The goal is to observe the interaction between the two seemingly contradicting factors: network throughput and power consumption.



■ **Figure 4.** *Scenarios of slave-activated sniff parameter negotiation.*

ensure that each piece of submatrices has a free entry. Let the folded matrix be M'. We then search M' sequentially for possible existence of q consecutive free entries (with value 0), where

$$q = \left\lfloor \frac{S_k \cdot 2^u \cdot T}{2^{u-p}} \right\rfloor = \left\lfloor S_k \cdot 2^p \cdot T \right\rfloor.$$

The reason we adopt a floor function instead of a ceiling function here is that the desired slot occupancy has been enlarged in our calculation (by dividing by a $\delta < 1$). Once this search succeeds, we can return $T_k' = 2^p \cdot T$, $N_k' = q$, and $O_k' = i \cdot T + j$, where O_k' is the offset indicating the starting point, say $M'[i, j]$, of the q consecutive free entries in M'.

An example is in Table 2, given $T = 15$, $u = 3$, $r_{lb} = 0.2$, $r_{ub} = 0.8$, and $\delta = 0.8$. Assuming that there are $K = 5$ slaves, in the beginning round 0 all slaves share 1/5 of slot groups. In round 1, the estimated W_2 of slave 2 decrease to 0.18. So we release its occupancy on M and allocate a new space for it. In the table, the ratio q/t means that the target number of 0's on M' is q and the searched sniff interval is t. For example, in round 1 we succeed in ratio 2/60 (underlined), which means we find two consecutive 0s when the searched sniff interval is 60. Similarly, in rounds 2 and 3 we succeed in ratios 3/120 and 6/30, respectively.

Shortest Sniff Interval First

The SSIF policy only differs from the LSIF policy in that it searches starting from smaller sniff intervals and gradually increasing the searched interval. Specifically, we will start from the shortest sniff interval T. If the search fails, we double the interval and repeat the search, until the longest interval $2^u \cdot T$ is tried. The intuition is that although the slot occupancy may remain the same, with a smaller sniff interval the buffers for this slave may experience less chance of overflow. Hence, SSIF has potential to improve network throughput. However, the cost is put on the energy, since the slave needs to wake up more frequently. An example of this policy is in Table 3 (with all inputs the same as in Table 2).

One minor detail we intentionally omitted in the above discussion is that when the slave's traffic load is very low, it is possible that the value of q is always less than 1 throughout the searching. In this case, we simply take the sniff interval $2^p \cdot T$ such that $S_k \cdot 2^p \cdot T$ is closest to 1 to do the search again by enforcing $q = 1$.

Simulation Results

We have developed a simulator to verify the effectiveness of our protocol. The goal is to observe the interaction between the two seemingly contradicting factors *network throughput* and *power consumption*. We simulated a single piconet with six Bluetooth units (one master and five slaves). The programming environment was GNU C++ on UNIX SunOS 5.7. No mobility was modeled (i.e., no device joining or leaving the piconet during the simulation process). The power consumption of the master was not a concern since we assume it has plug-in electricity. Each slave might switch between active and sniff modes, and we didn't consider other modes, such as hold and park. When switching modes or changing sniff parameters, hosts send control packets as described in Figs. 3 and 4. For each slave there is a separate buffer queue on the master side. Physical transmission problems such as fading and interference were not taken into account.

Our power model is derived based on experi-

ences in Lucent WaveLAN cards and Bluetooth [13], which is summarized below. It takes half a unit of power for a Bluetooth device to receive a one-slot packet, and one unit to transmit a one-slot packet. Voice traffic is not simulated (but we can simply reserve 1s in matrix M spaced by regular distances to model such traffic). When a slave hears packets dedicated to others, a lower amount of power is required since it can turn off its receiver after monitoring the packet header. In this case, it only consumes 1/6 of receiving power, which is 0.083 units. Bluetooth also defines some short packets, such as ACK and NULL (to respond to a poll with no data). We assume the power consumption for delivering such packets to be 1/6 of the transmission power, which is 0.167 units.

In addition to our LSIF and SSIF policies, three other approaches are simulated for comparison. The first one is called Always Active (AA), where we enforce all slaves to always stay in active mode. The master polls its slaves using a round-robin policy. Note that, while naive, AA is used here only as a reference point. The second approach is called Always-Sniff-with-Varying-Sniff-Interval (AS_VSI), where we enforce slaves to always stay in sniff mode, but the active window must remain constant. When a slave's slot utilization $\leq r_{lb}$, its sniff interval will be doubled. On the other hand, its sniff interval will be cut in half if slot utilization $\geq r_{ub}$. The smallest sniff interval is T, while the largest possible is $2^u \cdot T$. The third approach is called Always-Sniff-with-Varying-Active-Window (AS_VAW), where we enforce the sniff interval to be constant. The active window will be doubled when slot utilization $\geq r_{ub}$, but cut in half when $\leq r_{lb}$. The lower and upper bounds of active window size are 1 and $T/5$, respectively.

Below, we divide our presentation into two parts. The next section shows the results under fixed traffic patterns, the following section under varying traffic patterns. The latter is intended to model real system traffic and demonstrate the flexibility of our protocol in catching such dynamics. Through the presentation we shall provide a guideline for choosing proper parameters for our protocol. In our simulation we always adopt r_{lb} = 0.3 and r_{ub} = 0.7. Each simulation run lasts for 100,000 slot pairs.

Round0 $N_1/T_1=N_2/T_2=N_3/T_3=N_4/T_4=N_5/T_5=3/15$

S1	S1	S1	S2	S2	S2	S3	S3	S3	S4	S4	S4	S5	S5	S5
S1	S1	S1	S2	S2	S2	S3	S3	S3	S4	S4	S4	S5	S5	S5
S1	S1	S1	S2	S2	S2	S3	S3	S3	S4	S4	S4	S5	S5	S5
S1	S1	S1	S2	S2	S2	S3	S3	S3	S4	S4	S4	S5	S5	S5
S1	S1	S1	S2	S2	S2	S3	S3	S3	S4	S4	S4	S5	S5	S5
S1	S1	S1	S2	S2	S2	S3	S3	S3	S4	S4	S4	S5	S5	S5
S1	S1	S1	S2	S2	S2	S3	S3	S3	S4	S4	S4	S5	S5	S5
S1	S1	S1	S2	S2	S2	S3	S3	S3	S4	S4	S4	S5	S5	S5

Round1 $W_2=0.18<r_{lb}\Rightarrow S_2=[(3/15)*0.18]/0.8=0.045$
$0.045=5/120=\underline{2/60}=1/30=0/15\Rightarrow(O_k',T_k',N_k')=3,60,2)$

S1	S1	S1				S3	S3	S3	S4	S4	S4	S5	S5	S5
S1	S1	S1				S3	S3	S3	S4	S4	S4	S5	S5	S5
S1	S1	S1				S3	S3	S3	S4	S4	S4	S5	S5	S5
S1	S1	S1	S2	S2		S3	S3	S3	S4	S4	S4	S5	S5	S5
S1	S1	S1				S3	S3	S3	S4	S4	S4	S5	S5	S5
S1	S1	S1				S3	S3	S3	S4	S4	S4	S5	S5	S5
S1	S1	S1				S3	S3	S3	S4	S4	S4	S5	S5	S5
S1	S1	S1	S2	S2		S3	S3	S3	S4	S4	S4	S5	S5	S5

Round2 $W_3=0.11<r_{lb}\Rightarrow S_3=[(3/15)*0.11]/0.8=0.028$
$0.028=\underline{3/120}=1/60=0/30=0/15\Rightarrow(O_k',T_k',N_k')=(5,120,3)$

S1	S1	S1							S4	S4	S4	S5	S5	S5
S1	S1	S1							S4	S4	S4	S5	S5	S5
S1	S1	S1							S4	S4	S4	S5	S5	S5
S1	S1	S1	S2	S2					S4	S4	S4	S5	S5	S5
S1	S1	S1							S4	S4	S4	S5	S5	S5
S1	S1	S1							S4	S4	S4	S5	S5	S5
S1	S1	S1							S4	S4	S4	S5	S5	S5
S1	S1	S1	S2	S2	S3	S3	S3		S4	S4	S4	S5	S5	S5

Round3 $W_4=0.9>r_{lb}\Rightarrow S4=[(3/15)*0.9]/0.8=0.23$
$0.23=27/120=13/60=\underline{6/30}=3/15\Rightarrow(O_k',T_k',N_k')=(18,30,6)$

S1	S1	S1	S4	S4	S4	S4	S4	S4				S5	S5	S5
S1	S1	S1										S5	S5	S5
S1	S1	S1	S4	S4	S4	S4	S4	S4				S5	S5	S5
S1	S1	S1	S2	S2								S5	S5	S5
S1	S1	S1	S4	S4	S4	S4	S4	S4				S5	S5	S5
S1	S1	S1										S5	S5	S5
S1	S1	S1	S4	S4	S4	S4	S4	S4				S5	S5	S5
S1	S1	S1	S2	S2	S3	S3	S3					S5	S5	S5

Table 2. *Searching example of LSIF.*

Fixed Traffic Load

Here we assume a Poisson process with packet arrival rate λ = 0.2 (packets per time slot) for each buffer (on both the master and slave sides). Figure 5 illustrates the power consumption and network throughput against buffer size B_{max}. We observe that with enough buffer space ($\geq$ 50), all five approaches can achieve high throughput close to 0.99. This is because fixed traffic loads are easy to catch. Compared to AA, the other four schemes consume significantly less power.

These observations motivate us to derive the following simulations, where traffic is nonuniform, to reveal the strength of our proposals.

Varying Traffic Load

Here we adopt a variable traffic model similar to that proposed in [14]. Packets still arrive by the Poisson process, but at different rates. Two types of traffic patterns will be explored. The first is denoted TypeI(λ_M–λ_S), which means the arrival rate on the master's side is λ_M, and that on the slave's side is λ_S. The second is denoted TypeII(λ_A–λ_B), which means there are two kinds of arrival rates, λ_A and λ_B, for both the master and the slave. The master and the slave change states between rates λ_A and λ_B independently, and the transition probability from one rate to the other is 0.01 in both directions.

We first investigate the effect of weight α on our LSIF and SSIF schemes. Figure 6a plots the network throughput against α, where five slaves with traffic patterns TypeI(0.2–0.2), TypeI(0.19–0.01), TypeI(0.01–0.19), TypeII(0.19–0.01), and TypeII(0.19–0.01) are simulated. It indicates that

the throughput can be consistently improved as α increases, until $\alpha \leq 0.7$. At $\alpha = 0.0$, LSIF and SSIF give the worst throughput of all. The reason is that the evaluating metric is all based on buffer information when $\alpha = 0.0$, which is unfair. As a result, the assigned sniff-attempt slots are unable to handle future traffic well, thus degrading performance. After $\alpha > 0.7$, the throughput starts to degrade as α grows. However, even when $\alpha = 1.0$, our LSIF and SSIF still outperform the other three schemes. Thus, a value between 0.6 and 0.7 for α could be the best choice.

The above simulation in fact reveals two interesting phenomena. First, the slot utilization factor alone cannot predict the traffic well. One should take both slot utilization and buffer backlog information to predict future traffic. Second, and much to our surprise, our LSIF and SSIF schemes can even provide better network throughput than a naive AA round-robin scheme as α is properly set. The reason is that, in the AA case, the master wastes much time polling slaves with no backlogs, resulting in reduced throughput. This indicates a prospective direction in which one can save power and improve network throughput at the same time (which are contradicting factors by intuition).

Figure 6b illustrates the impact of weight δ on LSIF and SSIF, with $\alpha = 0.7$ and the same traffic pattern as above. It shows that before $\delta \leq 0.6$, the throughput can be improved as δ grows. Recall that δ is a factor to enlarge the expected sniff-attempt slots to tolerate a certain level of inaccuracy in our estimation. With a too small value (e.g., $\delta = 0.1$), LSIF and SSIF will degenerate into the AA scheme, since slaves can hardly obtain such a large slot occupancy. As a result, all slaves may remain active most of the time. This also violates our goal of conserving power. After $\delta > 0.6$, the throughput starts to decline as δ increases. After $\delta > 0.8$, our throughput falls behind that of the AA scheme. This is because our prediction is too conservative to catch the dynamics of variable traffic. Thus, a reasonable value for δ would be between 0.5. and 0.7.

Also with the same traffic pattern, in Fig. 7a, we study network throughput with different values of u, which controls the largest possible sniff interval. For both LSIF and SSIF, it shows that the throughput remains at around the same level when $u = 0$, 1 and 2, but decreases as u is larger. With $u = 0$, there is only one sniff interval available, which is T. With $u = 1$ and 2, there are two and three sniff intervals available, respectively. A larger sniff interval means that the slave needs to switch modes less frequently, which is more favorable. Based on these considerations, one may choose a proper value of u to use.

One may note that in the previous simulation, the advantage of using our 2D matrix M is not well justified. When $u = 0$, M degenerates to a 1D matrix; so why one should need a 2D M remains a question. The reason is that we have adopted $T = 100$. Since the lowest traffic load we may inject for each entity is 0.01, a sniff interval of $T = 100$ and active window of 1 can properly catch such traffic without much waste. To justify this point, we have simulated hosts with very low traffic loads. We have conducted another experiment with five slaves having the following traffic patterns: TypeI(0.2–0.2), TypeI(0.001–0.001), TypeI(0.002–0.002), TypeII(0.002–0.005), and TypeII(0.002–0.005). The result is in Fig. 7b. It indicates that the throughput can be improved as u increases, until $u \leq 4$. Before $u < 2$, the throughput of LSIF and SSIF is worse than AA. This phenomenon is due to slot waste caused by sniff intervals that are too short. (For example, to handle a low arrival rate of 0.002, the scheduler should reserve one out of every 500 slots in average. When u is set too small, say $u < 2$, the scheduler will reserve too much resource for such slaves. Therefore, both network throughput and power consumption might get hurt.) As mentioned earlier, the value of $2^u \cdot T$ should be large enough to capture the behavior of those slaves that have very low

Round0 $N_1/T_1=N_2/T_2=N_3/T_3=N_4/T_4=N_5/T_5=3/15$

S1	S1	S1	S2	S2	S2	S3	S3	S3	S4	S4	S4	S5	S5	S5
S1	S1	S1	S2	S2	S2	S3	S3	S3	S4	S4	S4	S5	S5	S5
S1	S1	S1	S2	S2	S2	S3	S3	S3	S4	S4	S4	S5	S5	S5
S1	S1	S1	S2	S2	S2	S3	S3	S3	S4	S4	S4	S5	S5	S5
S1	S1	S1	S2	S2	S2	S3	S3	S3	S4	S4	S4	S5	S5	S5
S1	S1	S1	S2	S2	S2	S3	S3	S3	S4	S4	S4	S5	S5	S5
S1	S1	S1	S2	S2	S2	S3	S3	S3	S4	S4	S4	S5	S5	S5
S1	S1	S1	S2	S2	S2	S3	S3	S3	S4	S4	S4	S5	S5	S5

Round1 $W2=0.18<r_{lb} \Rightarrow S2=[(3/15)*0.18]/0.8=0.045$
$0.045=5/120=2/60=\underline{1/30}=0/15 \Rightarrow (O_k',T_k',N_k')=(3,30,1)$

S1	S1	S1				S3	S3	S3	S4	S4	S4	S5	S5	S5
S1	S1	S1	S2			S3	S3	S3	S4	S4	S4	S5	S5	S5
S1	S1	S1				S3	S3	S3	S4	S4	S4	S5	S5	S5
S1	S1	S1	S2			S3	S3	S3	S4	S4	S4	S5	S5	S5
S1	S1	S1				S3	S3	S3	S4	S4	S4	S5	S5	S5
S1	S1	S1	S2			S3	S3	S3	S4	S4	S4	S5	S5	S5
S1	S1	S1				S3	S3	S3	S4	S4	S4	S5	S5	S5
S1	S1	S1	S2			S3	S3	S3	S4	S4	S4	S5	S5	S5

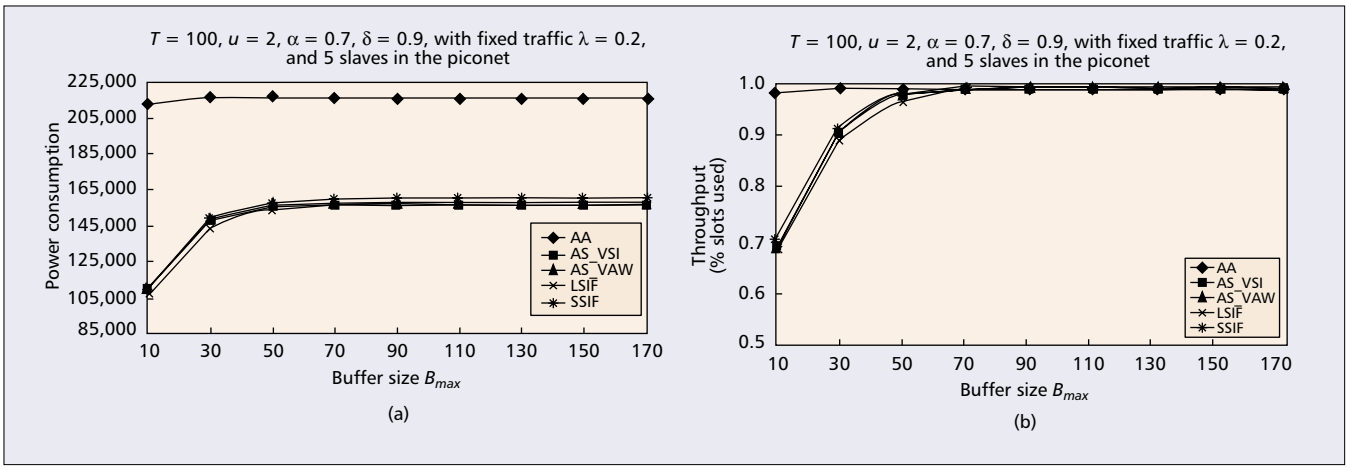
Round2 $W_3=0.11<r_{lb} \Rightarrow S_3=[(3/15)*0.11]/0.8=0.028$
$0.028=3/120=\underline{1/60}=0/30=0/15 \Rightarrow (O_k',T_k',N_k')=(4,60,1)$

S1	S1	S1							S4	S4	S4	S5	S5	S5
S1	S1	S1	S2						S4	S4	S4	S5	S5	S5
S1	S1	S1							S4	S4	S4	S5	S5	S5
S1	S1	S1	S2	S3					S4	S4	S4	S5	S5	S5
S1	S1	S1							S4	S4	S4	S5	S5	S5
S1	S1	S1	S2						S4	S4	S4	S5	S5	S5
S1	S1	S1							S4	S4	S4	S5	S5	S5
S1	S1	S1	S2	S3					S4	S4	S4	S5	S5	S5

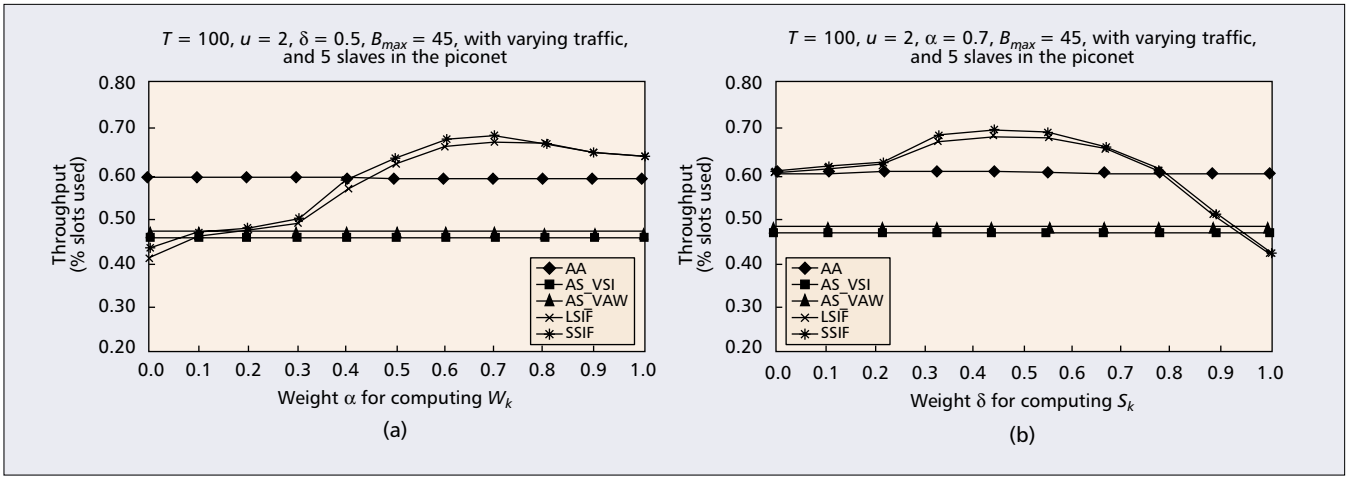
Round3 $W_4=0.9>r_{lb} \Rightarrow S4=[(3/15)*0.9]/0.8=0.23$
$0.23=27/120=13/60=6/30=\underline{3/15} \Rightarrow (O_k',T_k',N_k')=(5,15,3)$

S1	S1	S1			S4	S4	S4					S5	S5	S5
S1	S1	S1	S2		S4	S4	S4					S5	S5	S5
S1	S1	S1			S4	S4	S4					S5	S5	S5
S1	S1	S1	S2	S3	S4	S4	S4					S5	S5	S5
S1	S1	S1			S4	S4	S4					S5	S5	S5
S1	S1	S1	S2		S4	S4	S4					S5	S5	S5
S1	S1	S1			S4	S4	S4					S5	S5	S5
S1	S1	S1	S2	S3	S4	S4	S4					S5	S5	S5

■ Table 3. *Searching example of SSIF.*



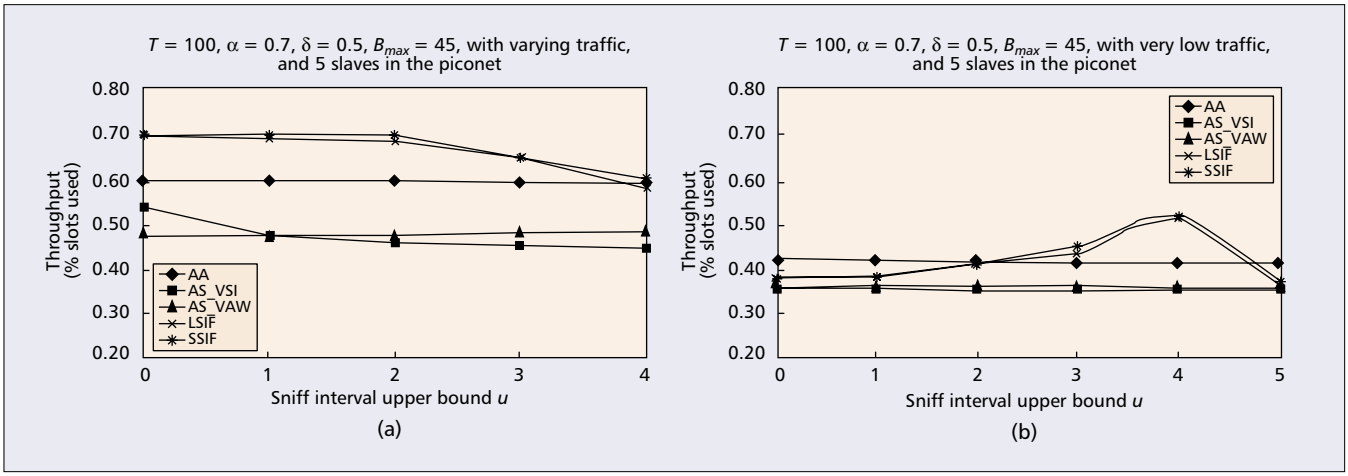
■ Figure 5. *The effect of* B_{max} *under fixed traffic load.*



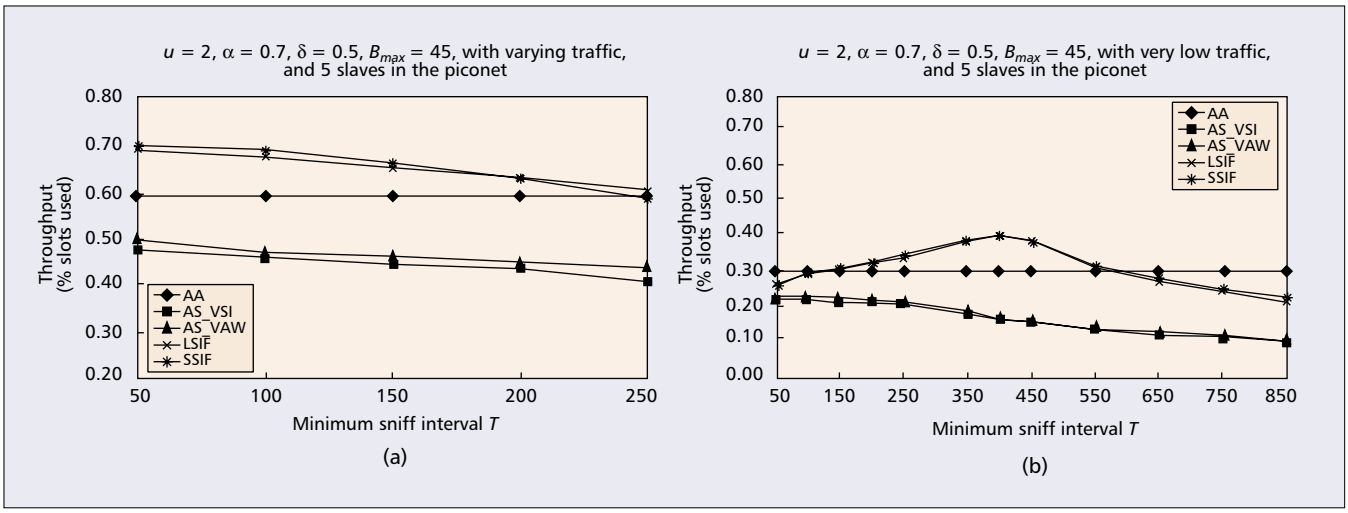
■ Figure 6. *The effect of a)* α *and b)* δ *on LSIF and SSIF under varying traffic load.*

traffic load. In Fig. 7b, a value of $u = 4$ will perform the best. This justifies that our 2D matrix representation of M is flexible enough to schedule sniffing slots for hosts with both high and very low traffic loads.

In Fig. 8a, we investigate the effect of T, which represents the smallest possible sniff interval, under the traffic patterns of TypeI(0.2–0.2), TypeI(0.19–0.01), TypeI(0.01–0.19), TypeII(0.19–0.01), and TypeII(0.19–0.01). It shows that the throughput of LSIF and SSIF will slightly decrease as T grows, so a T between 50 and 100 will be proper. The reason for the degradation in throughput is that we activate the evaluator based on the value of T. A larger T will trigger the protocol to reevaluate its traffic load less frequently. The inaccuracy in load estimation will cause slot waste. We believe this prob-



■ Figure 7. *The effect of* u *on LSIF and SSIF under a) varying traffic and b) very low traffic load.*



■ Figure 8. *The effect of* T *on LSIF and SSIF under a) varying traffic and b) very low traffic load.*

lem can be fixed by using a different rule to trigger our evaluators, which will be directed to future research.

Another experiment to investigate the impact of T is in Fig. 8b, where we try low traffic patterns: TypeI(0.2–0.2), TypeI(0.001–0.001), TypeI(0.002–0.002), TypeII(0.002–0.005), and TypeII(0.002–0.005). Similar to the earlier observations, lower traffic loads require larger $2^u \cdot T$. This is why we see continuous improvement before $T \leq 400$. Once T is too large, our evaluators will react to traffic changes too slowly, causing degradation in throughput. The results from Figs. 8a and 8b suggest T between 100 and 200.

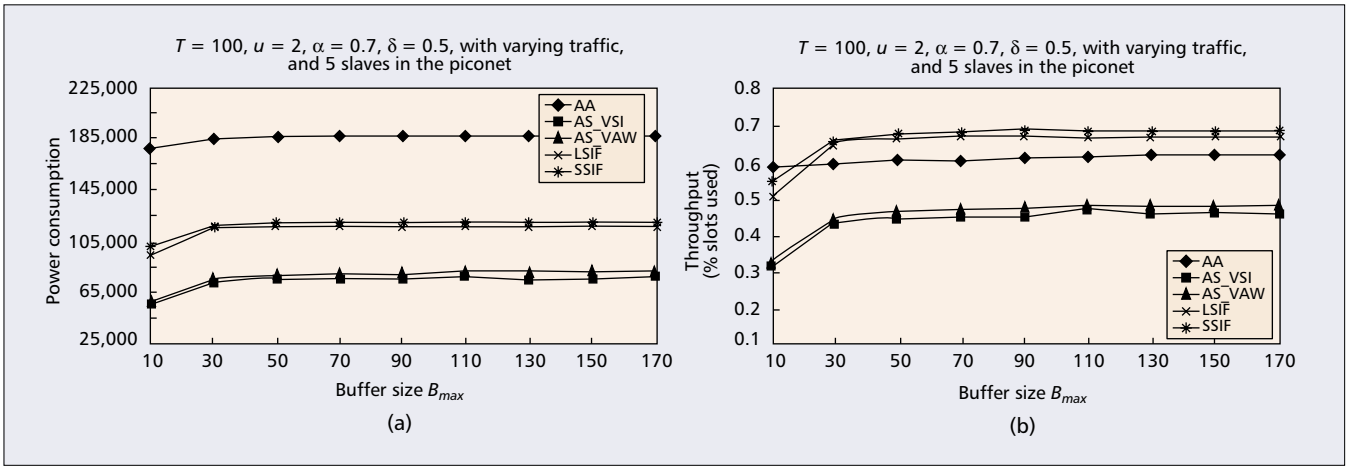
In Fig. 9, we set our parameters as recommended above and look at the impact of buffer spaces, B_{max}. We observe both power consumption and network throughput against different buffer spaces. It shows that the throughput climbs as B_{max} increases, up to $B_{max} = 50$. Once $B_{max} \geq 50$, the throughput remains almost the same. Thus, a buffer space between 30 and 50 is proper. With $B_{max} \approx 30$, LSIF and SSIF increase system throughput by around 16.7 percent compared to AA, while reducing power consumption by around 37.8 percent compared to AA.

Conclusions

We propose an adaptive and efficient protocol for managing the low-power sniff mode in Bluetooth. Two essential parts of our protocol are the evaluator and scheduler, which are responsible for measuring how well slaves utilize their sniff-attempt slots and arranging the sniff parameters for slaves in the sniff mode. A new representation based on a two-dimensional matrix is proposed to maintain slaves' sniff-attempt slots, which are conceptually infinite sequences of periodical slots. Two searching strategies, Longest Sniff Interval First and Shortest Sniff Interval First, are proposed to look for available sniff-attempt slots in the two-dimensional matrix. Our simulation results indicate that, with proper settings and buffer spaces, the protocol can potentially improve network throughput, while reducing power consumption, over a naive always active round-robin protocol.

Acknowledgments

Y.-C. Tseng would like to thank the Lee and MTI Center for Networking Research at NCTU, the Ministry of Education (contract nos. 89-H-FA07-1-4 and 89-E-FA04-1-4), and the National Science



■ Figure 9. *Effect of* B_{max} *under varying traffic load.*

Council, Taiwan (contract no. NSC90-2213-E009-154) for financially supporting this research.

REFERENCES

[1] Bluetooth SIG http://www.bluetooth.com, "Bluetooth Specification v1.1," Feb. 2001.
[2] J. C. Haartsen, "The Bluetooth Radio System," *IEEE Pers. Commun.*, Feb. 2000.
[3] J. C. Haartsen and S. Mattisson, "Bluetooth — A New Low-Power Radio Interface Providing Short-Range Connectivity," *Proc. IEEE*, vol. 88, Oct. 2000.
[4] R. Bruno, M. Conti, and E. Gregori, "WLAN Technologies for Mobile Ad Hoc Networks," *IEEE Proc. 34th Hawaii Int'l. Conf. Sys. Sci.*, 2001.
[5] S.-L. Wu, Y.-C. Tseng, and J.-P. Sheu, "Intelligent Medium Access for Mobile Ad Hoc Networks with Busy Tones and Power Control," *IEEE JSAC*, vol. 18, Sept., 2000, pp. 1647–57.
[6] R. Ramanathan and R. Rosales-Hain, "Topology Control of Multihop Wireless Networks Using Transmit Power Adjustment," *IEEE INFOCOM*, 2000, pp. 404–13.
[7] C.-F. Huang *et al.*, "Increasing the Throughput of Multihop Packet Radio Networks with Power Adjustment," *Int'l. Conf. Comp. Commun. and Nets.*, 2001.
[8] J. H. Ryu, S. Song, and D.-H. Cho, "A Power-Saving Multicast Routing Scheme in 2-tir Hierarchical Mobile Ad-Hoc Networks," *IEEE VTC*, vol. 4, 2000, pp. 1974–78.
[9] S. Singh, M. Woo, and C. S. Raghavendra, "Power-Aware Routing in Mobile Ad Hoc Networks," *Int'l. Conf. Mobile Comp. and Net.*, 1998, pp. 181–90.
[10] H. Woesner *et al.*, "Power-Saving Mechanisms in Emerging Standards for Wireless LANs: The MAC Level Perspective," *IEEE Pers. Commun.*, June 1998, pp. 40–48.
[11] I. Chakrabory *et al.*, "MAC Scheduling Policies with Reduced Power Consumption and Bounded Packet Delays for Centrally Controlled TDD Wireless Networks," *IEEE ICC*, 2001.
[12] A. Das *et al.*, "Enhancing Performance of Asynchronous Data Traffic over the Bluetooth Wireless Adhoc Network," *IEEE INFOCOM*, 2001.
[13] S. Garg, M. Kalia, and R. Shorey, "MAC Scheduling Policies for Power Optimization in Bluetooth: A Master Driven TDD Wireless System," *IEEE VTC*, 2000.
[14] M. Kalia, D. Bansal, and R. Shorey, "Data Scheduling and SAR for Bluetooth MAC," *IEEE VTC*, 2000.

BIOGRAPHIES

TING-YU LIN (tylin@csie.nctu.edu.tw) received her B.S. degree in computer science from National Chiao-Tung University, Taiwan, in 1996. She is currently a Ph.D. candidate at the Department of Computer Science and Information Engineering at the same university. Her research interests include wireless communication, mobile computing, personal-area networks, and energy conservation.

YU-CHEE TSENG (yctseng@csie.nctu.edu.tw) is currently a full professor in the Department of Computer Science and Information Engineering, National Chiao-Tung University, Taiwan. He has served as a Program Committee Member for several international conferences and as a Guest Editor for several journals, including *IEEE Transactions on Computers*, *Wireless Communications and Mobile Computing*, *Wireless Networks*, and *Journal of Internet Technology*. His research interests include wireless communication, network security, parallel and distributed computing, and computer architecture. He is a member of the IEEE Computer Society.

Our simulation results have indicated that, with proper settings and buffer spaces, the protocol can potentially improve network throughput, while reducing power consumption, compared to a naive always active round-robin protocol.