# A Traffic-Aware Scheduling for Bluetooth Scatternets

Jang-Ping Sheu, *Senior Member*, *IEEE*, Kuei-Ping Shih, *Member*, *IEEE Computer Society*, Shin-Chih Tu, and Chao-Hsun Cheng

**Abstract**—Bluetooth is a low cost, low power, short-range radio technology used for wireless personal area networks (PANs). Bluetooth scatternet is a set of piconets interconnected via bridge devices. Good interpiconet schedulings are necessary for bridge devices to switch among piconets they participate in. This paper proposes an interpiconet scheduling algorithm named "Traffic-Aware Scatternet Scheduling" (TASS), for bridges in Bluetooth scatternets. According to masters' traffic information, TASS can adaptively switch the bridge to high traffic load masters, and increase the usage of the bridge. In addition, TASS can reduce the number of failed "unsniffs" and the overhead of "bridge switch wastes" to further increase overall system performance. Simulation results show that TASS outperforms existing interpiconet scheduling in both network throughput and adaptability for various traffic loads.

Index Terms—Ad hoc networks, Bluetooth, interpiconet scheduling, mobile computing, personal area networks (PANs).

## **1** INTRODUCTION

**B**<sub>LUETOOTH</sub> is a low-cost, low-power, and short-range radio technology used for wireless personal area networks (PANs) [1]. It operates in the unlicensed 2.4 GHz ISM band. A Frequency Hopping Spread Spectrum (FHSS) scheme is used. The hopping frequencies cover 79 channels, each channel being 1 MHz wide. A *piconet* is a basic structure in Bluetooth, which is constructed in an ad hoc fashion by one master and up to seven active slaves [2], [3], [4]. A piconet can only contain one master and the master administers the whole piconet. A slave may connect to more than one master. A slave connecting to two or more masters is called a *bridge*. A set of piconets that are interconnected by bridges is referred as a *scatternet*. Although a bridge can participate in two or more piconets, it can only serve in one piconet at a time. The bridge will switch among all connected piconets in a time-sharing fashion.

The scheduling of a bridge switching among piconets is also referred to as *interpiconet scheduling*. Obviously, an illconsidered scheduling may cause severe system degradation. However, interpiconet scheduling is not specified in Bluetooth specification. This makes it imperative that interpiconet scheduling be developed and be well designed so as to help the bridge switch efficiently among piconets. On the other hand, the *intrapiconet scheduling* is referred as the scheduling of a master serving the slaves connected by him. *Polling* is a general scheme adopted for intrapiconet scheduling. Much research has been done on intrapiconet

For information on obtaining reprints of this article, please send e-mail to: tmc@computer.org, and reference IEEECS Log Number TMC-0113-0304. scheduling in the literature [5], [6], [7], [8], [9], [10]. However, intrapiconet scheduling is not in the scope of this paper.

Recently, a number of researches on interpiconet scheduling have been proposed [11], [12], [13], [14], [15], [16], [17], [18], [19], [20]. In [11], an Adaptive Presence Point Density scheme (APPD) for interpiconet scheduling is proposed. In APPD, a credit value is attached to each connection of a bridge that is established. According to the credit values, the bridge can decide whether to switch to another piconet or not. The power saving mode used to switch among piconets is referred to as the *sniff* mode. In order to reduce the guard time waste caused by the switching of the bridge among piconets and to avoid starvation of some waiting master, a minimum average service time is introduced to guarantee a minimum service time. However, the decision to switch is controlled by the bridge, without negotiating with the serving master. This may result in one or more packets being lost since the serving master still transmits packets to the bridge because the master assumes that the bridge is still in service. On the other hand, to preserve fairness among the connections, in the APPD scheme, the probabilities of masters getting the usage of the bridge are the same. Nevertheless, it may lead to a bottleneck, since the master with a high traffic load may have not enough service time to finish its transmission. In addition, reducing the number of failed unsniffs is not considered in the APPD scheme. In [17], an RV-maxmin optimal forwarding throughput algorithm is proposed. Like [11], the bridge also uses the sniff mode to switch to another piconet at an RV point. In the RV maxmin algorithm, each device will exchange their RV points information and estimate the possible forwarding throughput. In order for the algorithm to work correctly, each device has to exchange and store a lot of information. Thus, the communication overhead is substantial.

Instead of using the sniff mode, the authors in [12], [13], [16], [18] use the *hold* mode to switch among piconets. In

<sup>•</sup> J.-P. Sheu, S.-C. Tu, and C.-H. Cheng are with the Department of Computer Science and Information Engineering, National Central University, Jhongli 320, Taiwan. E-mail: sheujp@csie.ncu.edu.tw, shinzi@axp1.csie.ncu.edu.tw, josh.cheng@quantatw.com.

K.-P. Shih is with the Department of Computer Science and Information Engineering, Tamkang University, Tamshui 251, Taipei, Taiwan. E-mail: kpshih@mail.tku.edu.tw.

Manuscript received 22 Mar. 2004; revised 29 Oct. 2004; accepted 6 Feb. 2005; published online 16 May 2006.

SHEU ET AL.: A TRAFFIC-AWARE SCHEDULING FOR BLUETOOTH SCATTERNETS

[12], [13], a Load Adaptive Algorithm (LAA) for interpiconet scheduling is proposed for small scale scatternets. Two conditions are considered by a bridge to determine whether to switch to another piconet. One is when the bridge stays in a piconet past a time limit. The other is when the bridge reaches a commitment time with another piconet, or the queue size of the bridge to another piconet has exceeded a certain threshold value. In [18], two simple algorithms are presented, LAMS (Load Adaptive Master/slave Scheduling) and LASS (Load Adaptive Slave/slave Scheduling), for the scheduling of interpiconets in scatternets with an MS and an SS bridge, respectively. Both algorithms can dynamically adjust the switches among the bridge and the masters in order to minimize the end-to-end packet delays. Both algorithms are able to achieve near-optimal performance. However, the scheduling algorithms in [13], [18] can be applied to a bridge which only connects to two piconets. In [16], a distributed scatternet-scheduling scheme is proposed. The scheme uses the hold mode for bridges to switch among piconets. This scheme can adapt to nonuniform traffic. Fair allocation of bandwidth to each Bluetooth unit is also taken into account in this paper. However, the hold mode causes scheduling inflexibility and high negotiation overhead.

Therefore, a novel interpiconet scheduling protocol, Traffic-Aware Scatternet Scheduling (TASS), is presented in this paper. In it, the traffic which will last for a period of time is considered. As a result, the sniff mode is efficient and flexible for a bridge to switch among the piconets to which it connects. In this paper, the master that the bridge is serving is called the serving master, and the other masters that the bridge is connected to, but is not in service with, are called the waiting masters. In TASS, a bridge will switch to another piconet only when the current serving master notifies it to switch off. As a result, there is no packet loss when the bridge switches to another piconet. Each master maintains a scheduling table. The table records all the information on the traffic of the masters and their bridge usage status, such as how long a master has waited for the usage of the bridge, how long the master could not use the bridge, and so on. The scheduling table of the serving master will be transferred to the new serving master through the bridge when the serving master decides to release the usage of the bridge. In other words, a master that gets the usage of the bridge also gets the traffic information of all the other waiting masters. Based on the scheduling table, the serving master can predict the time it may not get the usage of the bridge after it releases the usage of the bridge. After releasing the bridge, the master will not unsniff the bridge during the time interval it has predicted. Therefore, the number of failed unsniffs is reduced substantially. When the new serving master gets the scheduling table from the bridge, it can figure out the minimum time it can freely use the bridge. The bridge can dynamically switch among the connected piconets according to the master's traffic loads and waiting time. To reduce the bottleneck of the master with a high traffic load, TASS can reflect the traffic of all masters that the bridge is connected to, and reserve enough service time to the serving master. It can reduce packet transmission delay

and further increase the throughput. Simulation results demonstrate that TASS outperforms the APPD scheme with a higher network throughput and better flexibility in a various traffic load environment.

The rest of this paper is organized as follows: Section 2 presents the challenges of interpiconet scheduling. In Section 3, Traffic-Aware Scatternet Scheduling is introduced. The bridge switching problem and its solution are proposed in Section 4. Simulation results are analyzed in Section 5, and Section 6 concludes this paper.

# 2 CHALLENGES OF THE INTERPICONET SCHEDULING

The power saving mode that a bridge uses to switch among piconets directly influences the performance of the scatternet. Bluetooth specifies three power saving modes, the sniff, hold, and park modes. In general, the sniff mode is used for a bridge to switch among piconets on a regular and periodical basis. A device in the sniff mode only wakes up periodically in prearranged sniff slots. The master and the slave must negotiate the sniff timing information, such as the first sniff slot, sniff interval  $(T_{Sniff})$ , sniff attempt, and the sniff timeout. The sniffing slave only listens for the traffic during the sniff slots. If no message is addressed to the sniffing slave, the sniffing slave ceases listening for packets. If a message is received in a sniff slot, the sniffing slave continues listening for further sniff timeout slots after the sniff slot. In other words, the transmission time is flexible between the master and the sniffing slave. The master and the sniffing slave can only communicate with each other in their prescheduled sniff slots. If either one can not receive packets from the other in a sniff slot, it is called a failed unsniff. A failed unsniff will lead to the loss of one packet. Consequently, too many failed unsniffs will significantly degrade the performance of the piconet, even that of a scatternet.

A bridge will switch among piconets in a time-sharing fashion. An interpiconet scheduling is needed for a bridge to switch among piconets. However, this scheduling is not straightforward. The following are the challenges to be considered by an interpiconet scheduling.

- The variation in traffic load. Since the traffic of a network must be variable, therefore, an interpiconet scheduling must have the flexibility to dynamically adjust the scheduling to meet the various traffic loads [11].
- The trade-off between throughput and transmission delay. To achieve the maximum throughput of a scatternet, it is preferred to allocate more bridge service time to the master with the high traffic loads. However, this may increase the transmission delay of the masters with low traffic loads [21]. Therefore, in addition to increasing the throughput, reduction of transmission delay must also be considered.
- The frequency of a bridge switching among piconets. In Bluetooth, each piconet has its own timeframe according to the master's clock. Therefore, the timing of one piconet is different from the next. As a result, when a bridge switches to a new

TABLE 1	
Scheduling Table	

MID	LT	QCT	WT	$\alpha$
:	÷	÷	÷	÷

piconet, the bridge may not match the timing of the new piconet. Therefore, the bridge has to wait until the next even slot to be unsniffed by the new serving master. The time that a bridge waits for being unsniffed after switching to a new piconet is called *guard time waste*. Reducing the switching frequency of a bridge among piconets can also reduce the guard time waste [22].

In general, the communication between two devices always lasts for a specific amount of time. Consequently, the traffic style will be very similar from time to time, meaning that the traffic has temporal locality. Consequently, interpiconet scheduling of a bridge can utilize the historical information to enhance the efficiency of the scheduling.

# 3 TRAFFIC-AWARE SCATTERNET SCHEDULING (TASS) PROTOCOL

#### 3.1 System Model

TASS is operated on a constructed scatternet. Only ACL (Asynchronous Connectionless) link is considered in the paper for the connection between a master and a slave. In TASS, the sniff mode is used as the operating mode for the bridge to switch among piconets. The sniff interval negotiated by a bridge with its serving master is  $T_{Sniff}$ .

In TASS, each master maintains a *scheduling table*, which contains the traffic information of all masters that the bridge is connected to. When the serving master decides to release the usage of the bridge, it has to update its traffic information in the scheduling table. The bridge will transfer the scheduling table from the old serving master to the new serving master. According to the scheduling table, the new serving master can figure out the time it can use the bridge, and the waiting master can calculate the time it needs not to poll the bridge in the following sniff slots. Therefore, with the scheduling table, each master can record its traffic information in the table and obtain the traffic information of the neighboring masters at the same time. The scheduling table is very helpful in designing the TASS scheme.

A scheduling table is shown in Table 1, where MID represents the identity of the master and  $LT_i$ ,  $QCT_i$ ,  $WT_i$ , and  $\alpha_i$  are the traffic information of master *i*. The details of the fields in the scheduling table are described below. The scheduling table includes the following fields:

- *QCT* (Queue Consuming Time): the estimated time that a link will need the bridge to serve.
- *LT* (Lost Time): the estimated time that a master can not get the usage of the bridge.
- *WT* (Waiting Time): the time that a master has been waiting for the usage of the bridge.

• *α*: the historical information of, on average, the traffic generation rate per slot between the master and the bridge.

The unit of time described above is a time slot.

QCT is defined as the time that a link needs to transmit all the data packets in the queues of the master and the bridge. There is a queue agent to monitor the status of the queue on either side of a link. The bridge will notify the master about this information at each communication with the master. Based on this information, the master can obtain the QCT.

*LT* is defined as the time that a master cannot use the bridge.

The QCTs of all masters connected by the bridge are stored in the scheduling table. When the serving master has to release the usage of the bridge, according to the QCTs, the serving master can predict the duration from the time it releases the bridge to the time it obtains the bridge next time. This duration is called LT. LT can be used to reduce the number of failed unsniffs of the waiting masters. For example, when master A has to release the usage of the bridge to master B, master A will compute the  $LT_A$  to predict how many time slots that it may lose the usage of the bridge in the future. Thus, after master A releases the usage of the bridge, master A will skip the sniff slots during the  $LT_A$ . Therefore, master A can reduce the number of the failed unsniffs.

WT is the time that a master has been waiting for the usage of the bridge since it released the usage of the bridge.  $\alpha$  represents the history of the traffic loads, which is defined as the historical information on the average traffic generation rate per slot between the master and the bridge. Since the decision of the master to release the bridge depends much on the value of QCT, the precision of QCT will influence the performance of TASS. Therefore, to obtain a precise QCT, the history of the traffic loads is counted so as to evaluate the QCT due to the temporal locality of the traffic. Let  $\rho$  be the increment of the traffic in queue during a fixed time period, say T. The queue agent responds to maintain  $\rho$ . Thus,  $\alpha$  can be obtained as  $\alpha = \frac{\rho}{T}$  and will be computed for every T time period. After  $\alpha$  is obtained, the queue agent will reset  $\rho$  to zero. For example, suppose T = 20. Assume that a master and the bridge will generate two DH5 packets in T. So,  $\rho = 10$  and  $\alpha = 10/20 = 0.5$ . This means that the QCT of the master-bridge link will increase 0.5 packets per slot on average. When the serving master has to release the usage of the bridge, it records the  $\alpha$  in the scheduling table. Hence, when the new serving master gets the usage of the bridge, it can evaluate the QCT more precisely for a waiting master.

We have introduced how to obtain QCT precisely by means of  $\alpha$ . In the following, we will introduce how to obtain *LT* by means of *QCT* and  $\alpha$ .

LT means the time that the serving master will not get the bridge after it releases the usage of the bridge. When the serving master *i* has to release the usage of the bridge, it will find a candidate to be the new serving master, say *j*, and will update the  $LT_i$ . The serving master *i* first finds the minimum  $LT_j$  from the scheduling table for some *j*. If there



Fig. 1. An example of *LT*.

are more than one minimum LT, then it selects the one with the maximum WT. This means that the waiting master j has the highest priority to get the usage of the bridge once the serving master releases the bridge.

The serving master has to update  $LT_i$  once it decides to release the usage of the bridge to the new serving master j. However,  $QCT_j$  in the scheduling table of master i is an outof-date value since it is recorded when the master j has released the usage of the bridge. Therefore, it does not stand for the current traffic loads of master j. As a result, we can use  $\alpha_j$  to roughly estimate  $QCT_j$ . Therefore, the time that the serving master i will not get the usage of the bridge, let's call it WS, can be obtained as follows:

$$WS = QCT_i + \alpha_i * WT_j.$$

At the same time, to avoid excessive transmission delay of the waiting masters, a waiting threshold ( $W_{thold}$ ) is used to limit the transmission delay. If the WS is larger than  $W_{thold}$ , then WS is set to  $W_{thold}$ . Because a master can only communicate with a bridge on the sniff slots, WS may not coincide with the sniff slots. So we have to add an offset,  $\Delta$ , to match the sniff slot exactly.  $\Delta$  can be calculated as follows:

$$\Delta = T_{Sniff} - ((WS + 2 - D) \mod T_{Sniff}),$$

where D is the number of slots from the current slot to the next sniff slot. Therefore, the time that the serving master i will not get the bridge after it has released the usage of the bridge is

$$LT_i = WS + \Delta.$$

Take Fig. 1 as an example, where master *A* is the serving master and  $T_{Sniff} = 8$ . At the current slot, master *A* finds that  $LT_B = 0$ . The scheduling table of master *A* at the current slot is shown at the upper left corner in Fig. 1. Master *A* now has to decide whether to release the bridge or not. Assume that master *A* is going to release the bridge. Master *A* now has to estimate how long it may not get the bridge, its  $LT_A$ . In this example,  $WS = QCT_B + \alpha_B * WT_B = 12 (QCT_B = 12, \alpha_B = 0, \text{ and } WT_B = 4)$  and D = 4.  $\Delta = 8 - ((12 + 2 - 4) \mod 8) = 6$ . Hence,

$$LT_A = WS + \Delta = 12 + 6 = 18.$$

#### 3.2 The Protocol

TASS consists of two phases: the *bridge phase* and the *bridgeless phase*. The serving master executes the *bridge phase* and all the other waiting masters perform the *bridgeless phase*.

#### 3.2.1 Bridge Phase

If a serving master i gets the usage of the bridge, it first finds the minimum  $LT_i$  from the scheduling table, for some j. According to this information, master i will know how much time it has available to use the bridge freely. In addition, master i is responsible for the maintenance of the scheduling table. That is to say, serving master *i* should add 1 to each WT and subtract 1 from each LT, per slot, in the scheduling table. When  $LT_i = 0$ , master *i* must check if it has to release the bridge to the waiting master *j*. When the release condition is satisfied, the serving master *i* has to release the usage of the bridge to the waiting master j. Serving master *i* then performs the serving master part of the Bridge Release Procedure. As described above, once serving master *i* intends to release the bridge, it will calculate  $LT_i$  by means of the scheduling table. After the  $LT_i$ is calculated, master *i* updates  $LT_i$  in the scheduling table and resets the  $WT_i$  to zero. Master *i* then transmits the scheduling table to the bridge, and informs the bridge to serve the new serving master j. The role of master i is turned from being a serving master to that of a waiting master. Therefore, afterward, master i will perform the bridgeless phase. The bridge receiving the scheduling table will perform the bridge part of the Bridge Release Procedure as well. The bridge then waits for being unsniffed by the new serving master j and maintains the scheduling table during this waiting period. Maintenance means that the bridge will record the time slot count (sc) during the period from the time it returns an ACK to the old serving master to the time it returns another ACK to the new serving master, acknowledging the unsniff of the new serving master. The period should include the guard time difference between the old and the new serving masters. When the bridge is unsniffed by the new serving master, it subtracts sc from each LT, adds sc to each WT in the scheduling table, and then transmits the scheduling table to the new serving master. The bridge phase and the bridge release procedure are demonstrated in Algorithms 1 and 2, respectively.

#### Algorithm 1 Bridge Phase.

{The serving master should execute the algorithm *per slot.*} **Step 1**:

The serving master, say *i*, maintains the scheduling table. The maintenance is to add 1 to every WT, subtract 1 from every LT (for all waiting masters), and update the  $QCT_i$  in the scheduling table according to its queue status. **Step 2**:

if there is no data to send between the serving master i and the bridge **then** 

Execute the Bridge Release Procedure.

#### end if

Step 3:

if there is no any LT except  $LT_i$  in scheduling table is equal to zero **then** 

Go to **Step 8**. end if

if there are more than one waiting master with LT = 0 then Select the waiting master j with the largest WT and the other LTs are reset to  $T_{Sniff}$ 

end if

Step 5:

if  $WT_j > W_{thold}$  then

Execute the *Bridge Release Procedure* {TIME event} Go to **Step 8**.

## end if

#### Step 6:

if  $QCT_j + \alpha_j * WT_j > QCT_i + QC_{thold}$  then Execute the Bridge Release Procedure {QUEUE event} Go to Step 8.

end if Step 7:

Reset  $LT_j$  to  $T_{Sniff}$  {EXTEND event}

Go to Step 8.

## Step 8:

End.

Under the following conditions, serving master i has to release the usage of the bridge to the waiting master j.

(C1)  $WT_j > W_{thold}$ , (TIME event),

(C2)  $QCT_j + \alpha_j * WT_j > QCT_i + QC_{thold}$ , (QUEUE event).

(C1) implies that master j has been waiting for the bridge past the  $W_{thold}$ . (C2) implies that all the data required to be transmitted completely between master j and the bridge is larger than those between master i and the bridge plus a  $QC_{thold}$ . The  $QC_{thold}$  is designed for avoiding the ping-pong effect when  $QCT_i$  and  $QCT_j$  are too close to each other.

(C1) is used to avoid excessive transmission delay of the waiting master. The released event triggered by this condition is termed a TIME event. (C2) is used to allocate more service time to the link with high traffic loads. The released event triggered by this condition is termed a QUEUE event. If none of the two conditions are satisfied, then the serving master i can keep using the bridge. This is termed an EXTEND event. It is worth mentioning that an EXTEND event will result in a failed unsniff for the waiting master which has the highest possibility of getting the usage of the bridge in the near future (here, this implies waiting master j). However, the EXTEND event implies that the traffic load for waiting master j is not larger than the load of serving master i by a prespecified threshold.

To improve the throughput of a scatternet, the master with high traffic loads will be allocated more service time. However, when an EXTEND event is triggered, it also implies that the  $LT_j$  of the waiting master j expires. Therefore, master j will try to unsniff the bridge on the sniff slots in the future. Consequently, the  $LT_j$  in the scheduling table of the serving master i must reset to  $T_{Sniff}$ .

Algorithm 2 Bridge Release Procedure.

The part to be executed by the serving master. Step 1:

Calculate  $LT_i$ .

# Step 2:

Update  $LT_i$  and reset  $WT_i$  to zero in the scheduling table. Step 3:

IEEE TRANSACTIONS ON MOBILE COMPUTING, VOL. 5, NO. 7, JULY 2006

Transfer the scheduling table to the bridge and inform the bridge to be unsniffed by the new serving master. **Step 4**:

Wait for the ACK from the bridge. Go to the *Bridgeless phase*.

# The part to be executed by the bridge.

{The bridge receiving the scheduling table and being informed by the serving master i to switch to another piconet to serve the new serving master j will perform the following operations.}

## Step 1:

Send an ACK to the old serving master i at the following odd slot after it receives the scheduling table.

# Step 2:

Maintain the time slot count sc during the time it returns an ACK to the old serving master to the time it returns another ACK to the unsniff message from the new serving master j, including the guard time difference between the two piconets mastered by i and j.

## Step 3:

Maintain the scheduling table. Add sc to every WT and subtract sc from every LT in the scheduling table.

#### Step 4:

Transfer the scheduling table to the new serving master *j*.

# 3.2.2 Bridgeless Phase

The waiting masters that do not get usage of the bridge will perform the *bridgeless phase*. For some waiting master, say j, according to the  $LT_j$  that was calculated when master j released the usage of the bridge, it can realize the time  $(LT_j)$  it might not get the usage of the bridge. Therefore, it won't unsniff the bridge during  $LT_j$ . Hence, this can reduce the number of failed unsniffs. All waiting masters will perform Algorithm 3 in the *bridgeless phase*.

Algorithm 3 Bridgeless Phase.

{The waiting master should execute the algorithm *per slot*. Suppose the waiting master is master j, for some j.}

if 
$$LT_j > 0$$
 then

$$LT_j = LT_j - 1$$

else

Back to normal operation of sniff mode.

{It implies that the master j will try to unsinff the bridge on the following sniff slots.}

end if

if master *j* unsniffs the bridge successfully **then** Go to the *Bridge phase*.

# end if

# 3.3 An Example

Fig. 2 shows an example of TASS for a bridge with three links each to masters *A*, *B*, and *C*, respectively. Assume that  $T_{Sniff} = 8$ ,  $W_{thold} = 20$ , and  $QC_{thold} = 5$ . Master *A* is the serving master.





(0)							
1ID	LT	QCT	WT	α			
Α	18	2	4	0.2			

QCI

2

20

4

/1- \

WT α

> 3 0.2

\_ 1

13 0.5

13 0.5

1

LT

19

1

8

A

В

C

В

C

(c)
· · ·

,		ī.		
1	C	ł	١	

19

4

MID	LT	QCT	WT	α	MID	LT	QCT	WT	α
Α	10	2	12	0.2	Α	7	2	15	0.2
В	20	11	0	1	В	17	11	3	1
С	0	4	21	0.5	С	—	15	_	0.5
		(e)					(f)		

Fig. 3. The scheduling tables of the example shown in Fig. 2. (a) The scheduling table that master A transmits to the bridge by the QUEUE event. (b) The scheduling table that master B receives from the bridge by the QUEUE event. (c) The scheduling table of master B before the EXTEND event. (d) The scheduling table of master B after the EXTEND event. (e) The scheduling table that master B transmits to the bridge by the TIME event. (f) The scheduling table that master C receives from the bridge by the TIME event.

#### 3.4 Implementation Issue

TASS is a practical and implementable protocol. We analyzed TASS from the viewpoints of execution time as well as memory space to verify its practicability.

Suppose a bridge participates in n piconets. The scheduling table is an  $n \times 5$  table. For Algorithm 1, Steps 1, 3, and 4 take O(n) time and the rest take O(1) time. In total, the time complexity of Algorithm 1 is O(n). Similarly, in Algorithm 2, the part to be executed by the serving master takes O(1) time and the part to be executed by the bridge takes O(n) time. Moreover, Algorithm 3 takes O(1) time. Only one of the three algorithms at a time will be performed by a bridge or a master. Moreover, all the operations are easy operations, both in logic and arithmetic. In general, n is very small, about 2 to 3, and little time is required to execute the TASS protocol. As a result, a Bluetooth device can finish a procedure within a time slot.

On the other hand, from the storage point of view, the size of a scheduling table is small as well, ranging from only several bytes to tens of bytes. For a one-slot packet, the payload can be 17 or 27 bytes (DM1/DH1). This is enough to enclose a scheduling table in the payload of a packet.

In short, TASS is an easily implemented and costeffective (in both time and storage) protocol.

#### BRIDGE SWITCHING PROBLEM 4

If a bridge switches to a piconet, but the serving master has no data going to the bridge, a Poll-Null sequence event happens. It is called a bridge switch waste. In Bluetooth, the

Fig. 2. A TASS protocol example.

Suppose the scheduling table of the serving master A is shown in Fig. 3a. In the scheduling table,  $LT_B = 0$ . Serving master A will check if it should release the bridge to waiting master B. The second released condition (QUEUE event) is satisfied since  $QCT_B + \alpha_B * WT_B > QCT_A + QC_{thold}$ . Therefore, serving master A should release the bridge by the QUEUE event. Serving master A will calculate  $LT_A$  to see how long it may not get the usage of the bridge after it releases the usage of the bridge. Master A will reset  $WT_A$  to zero as well. In addition, master A updates  $LT_A$ ,  $QCT_A$ ,  $WT_A$  and  $\alpha_A$  in the scheduling table, as shown in Fig. 3a. Serving master A then transmits the scheduling table to the bridge and informs the bridge to switch to new serving master B. After releasing the usage of the bridge, the role of master A is changed from being a serving master to being a waiting master and, therefore, will be in the bridgeless phase afterward. During the  $LT_A$ , waiting master A will not unsniff the bridge (i.e., skip the first and the second sniff slots of  $M_A$ -bridge link).

On the other hand, master B can unsniff the bridge successfully on the first sniff slot of the  $M_B$ -bridge link and get the scheduling table transferred from the bridge, as shown in Fig. 3b. Master B then becomes the new serving master and will perform the bridge phase.

When  $LT_C = 0$ , the scheduling table of the serving master B is shown in Fig. 3c. Since master B checks that none of the two release conditions are satisfied (i.e.,  $WT_C$  <  $W_{thold}$  and  $QCT_C + \alpha_C * WT_C < QCT_B + QC_{thold}$ , it can keep the usage of the bridge by EXTEND event. The scheduling table updated by serving master B is shown in Fig. 3d. At the same time, the  $LT_C$  for waiting master C is expired, and it will try to unsniff the bridge on its first sniff slot of the  $M_C$ -bridge link. This results in one failed unsniff.

When serving master B detects that  $LT_C = 0$  again, which is two slots before the second sniff slot of the  $M_C$ -bridge link, the first condition of the two release conditions is satisfied ( $WT_C > W_{thold}$ ), and serving master B will release the bridge by the TIME event. Thus, waiting master C will get the usage of the bridge by successfully unsniffing the bridge on the following sniff slot. The scheduling tables that the old serving master B transfers to the bridge and which the new serving master C receives from the bridge, are shown in Fig. 3e and Fig. 3f, respectively.

access to the medium is based on a Time Division Duplex (TDD) scheme controlled by the master. So, even if there is no data in the queue from a master to a slave, the master still has to poll the slave by means of a poll packet to see if there are data that the slave wants to transmit to the master. If there is no data in the slave's queue to the master, the bridge will ACK the master with a null packet on the following odd slot. Therefore, if no data is to be transmitted between a master and a bridge, the Poll-Null sequence event could happen frequently. However, if the Poll-Null sequence event happens in the sniff slot between the master and the bridge, it implies that this switch of the bridge was wasted. It leads the bridge to go back to sleep, and the other waiting masters will be unable to unsniff the bridge successfully, thereby reducing the usage of the bridge.

In TASS, in order to reduce the number of bridge switch wastes, the LT will be increased as long as the bridge switch waste occurs. When a bridge switches to a serving master, the bridge will transfer the scheduling table to the serving master on the first odd slot. If the serving master receives a DH1 ACK packet from the bridge with no data included except the scheduling table, the serving master will regard the ACK packet as a Null packet. Thus, the LT of the serving master will be increased accordingly. The details of the solution are described as follows: When serving master *i* gets the usage of the bridge, it will check whether a Poll-Null sequence event is happening in the sniff slot. If a Poll-Null sequence event has happened, then it implies that the serving master *i* and the bridge have no data for each other. Therefore, an additional time (PNT, Poll-Null time) is added to the  $LT_i$ . As a result, the  $LT_i$  will be lengthened after a Poll-Null sequence event has happened. Consequently, it can reduce the number of the bridge switch wastes. In this paper, a Poll-Null event counter is used to record the times of successive bridge switch wastes, and the PNT will be lengthened twice for each Poll-Null sequence event. Hence, *PNT* can be obtained as follows:

$$PNT = T_{Sniff} * 2^{(\text{Poll-Null event counter})}$$

The  $LT_i$  will be lengthened after a bridge switch waste by:

$$LT_i = LT_i + PNT.$$

The LT increases if bridge switch wastes happen often. To avoid excessive transmission delay while master *i* has data to the bridge, an upper bound, MaxLT, is implemented to limit the increase of LT.

At the same time, because the  $LT_i$  may exceed the  $W_{thold}$ ,  $WT_i = -LT_i$  and is recorded in the scheduling table so that  $WT_i$  will not be constrained by  $W_{thold}$ . Once there are data to be transmitted between the master and the bridge, the Poll-Null event counter will be reset to zero. This method to relieve the bridge switch wastes is summarized in Algorithm 4:

Algorithm 4 Bridge Switch Wastes Reducing Procedure.

{The master getting the usage of the bridge will perform the procedure.}

## Step 1:

if master *i* gets the usage of the bridge and the Poll-Null sequence event is happened in the sniff slot **then** 



Fig. 4. Comparison of the results with and without the improvement of bridge switch waste. (a) No improvement. (b) With improvement.

Go to Step 2. else Reset the Poll-Null event counter to zero. Go to the *Bridge phase*. end if Step 2: Add 1 to Poll-Null event counter. Compute the  $LT_i$  and PNT. Step 3: if  $LT_i + PNT \leq Max LT$  then  $LT_i = LT_i + PNT$ end if Step 4:  $WT_i = -LT_i$ Execute the *Bridge Relagee Procedure* 

Execute the *Bridge Release Procedure*.

In the following, an example is given to demonstrate the procedure for reducing bridge switch wastes. Fig. 4a shows the result when there is no improvement of the bridge switch wastes and Fig. 4b shows the result with an improvement. In Fig. 4b, the numbers below the gray squares indicate the values of the Poll-Null event counter and zero implies that the Poll-Null event counter is reset to 0 due to data transmission between the master and the bridge. Assume that LT will be the same after each bridge switch waste. In Fig. 4b, the LT will be lengthened after each bridge switch waste. The larger the number of successive bridge switch wastes, the longer the LT. It is evident in Fig. 4 that this procedure reduces the number of bridge switch wastes three-fold.

#### **5** SIMULATION RESULTS

In this section, we use the CSIM simulator [23] to verify the feasibility of the proposed protocol. CSIM is a general purpose discrete-event simulator, which can be used for modeling wireless networks, traffic models, communication protocols, and so on.

We have made experiments to analyze the impact of the two thresholds,  $W_{thold}$  and  $QC_{thold}$ , on the activity ratio and network throughput. In comparison with the related work, two scenarios are performed in the simulations. The first scenario is to evaluate the performance of TASS in comparison with the related work when seven masters



Fig. 5. Simulation topology.

share with a bridge and the traffic load of a master is much higher than the others. The second scenario is to illustrate the detailed behavior of TASS against the related work when three masters share with a bridge and the traffic load of a master varies form high to low. The specific simulation settings are respectively described at each scenario.

The topology on which the experiment is performed is shown in Fig. 5. In this topology, there are at most seven piconets sharing a bridge. This is done because we are interested in the influence of the interpiconet scheduling on the scatternet performance. The packet generation rates of the masters have a constant bit rate (CBR). Among these masters, the packet generation rate of one master is fixed at 300kbps and those for the others are fixed at 60kbps. A high packet generation rate implies that the master will need more bridge service time. The bridge does not generate any packets of its own, and the destinations of all packets are to the bridge. The period to collect the historical information is 160 slots. The data queue size is 32KB. Initially, all WTs in the scheduling table are set to  $W_{thold}$ . The simulation time is 200 seconds.

In TASS, two thresholds are utilized for a serving master to decide whether to release the usage of a bridge or not. One is  $W_{thold}$ , which is used for the TIME event in order to avoid excessive delay time of waiting masters. The other one is  $QC_{thold}$ , which is used for the QUEUE event to prevent the ping-pong effect. Therefore, we carried out some experiments to observe how these thresholds effect the performance of TASS. In the following experiments, the bridge connects to seven masters,  $T_{sniff} = 20$ ,  $W_{thold}$  varies from 20 to 100 in steps of 10, and  $QC_{thold} = 5, 10, 15$ , and 20.



Fig. 7. The effect of the  $\mathit{W}_{thold}$  and  $\mathit{QC}_{thold}$  on the network throughput.

Fig. 6a and Fig. 6b illustrate the effects of the  $W_{thold}$  and  $QC_{thold}$  on the activity ratios of the bridge between the master with a high traffic load and between all seven masters, respectively. Since the increase of  $W_{thold}$  will decrease the switch frequency of the bridge and, therefore, the serving master with a high traffic load can have plenty of time to send, the activity ratio of the master with a high traffic load will increase with the increase of  $W_{thold}$ . The same is true for the total activity ratio. On the other hand, if the  $QC_{thold}$  is too small, the waiting master with a low traffic load has a high possibility to obtain the usage of the bridge. Thus, the activity ratio for the low  $QC_{thold}$  value is worse than those for the high  $QC_{thold}$  values.

Fig. 7 illustrates the effects of  $W_{thold}$  and  $QC_{thold}$  on the total network throughput. Since the total activity ratio will increase with the increase of the  $W_{thold}$ , the total network throughput also increases with the increase in the  $W_{thold}$ . Similarly, the throughput for the low  $QC_{thold}$  value is worse than those for the high  $QC_{thold}$  values. However, if the  $QC_{thold}$  is too large, it will cause the serving master with a low traffic load not to release the bridge. Therefore, the best performance is at  $QC_{thold} = 10$  when the  $W_{thold}$  is large enough.

From the above experimental results, we find that when  $W_{thold}$  and  $QC_{thold}$  are large enough (in the above experiment,  $W_{thold} > 40$  and  $QC_{thold} > 5$ ), the results are very close to each other and varying both parameters would not affect the performance significantly.



Fig. 6. The effect of the  $W_{thold}$  and  $QC_{thold}$  on the activity ratio. (a) The activity ratio of the master with a packet generation rate of 300 kbps. (b) The total activity ratio.



Fig. 8. The impact of the degree of the bridge on the bridge activity ratio of the master with a high traffic load.

Basically, the settings of  $W_{thold}$  and  $QC_{thold}$  are application-dependent.  $W_{thold}$  depends on the delay time that the application can tolerate, and  $QC_{thold}$  depends on the queue size and the traffic generation rate in the system. From our simulation, threshold values that are too large and too small result in poor performance. Otherwise, the performance of each combination of the two threshold values is very close to each other. Consequently, the two thresholds,  $W_{thold}$  and  $QC_{thold}$ , are set to 50 and 10, respectively, for the following experiments.

Currently, APPD (Adaptive Presence Point Density) [11] is a well-known interpiconet scheduling method. Moreover, APPD also adopts the *sniff* mode for a bridge to switch among the connected piconets. Therefore, the comparisons between TASS and APPD (Adaptive Presence Point Density [11]) on throughput, activity ratio, packet delay, and the number of failed unsniffs are presented here as well. In the following experiments,  $T_{Sniff} = 20$ ,  $QC_{thold} = 10$ ,  $W_{thold} = 50$ , and Max LT = 100, and the unit measurement is a slot.

Fig. 8 and Fig. 9 show the impact of the degree of the bridge on the activity ratio and the throughput of a master with a packet generation rate of 300kbps, respectively. The bridge degree represents the number of piconets connected to the bridge. The activity ratio means the ratio of the total bridge service time of the master with a packet generation rate of 300kbps for the total duration of the simulation. The throughput is evaluated by the number of data packets received per second by the bridge. Obviously, TASS can allocate more bridge service time to the master with a high traffic load. The master with a high traffic load can almost



Fig. 10. The impact of the bridge degree on the total activity ratio.

obtain the maximum throughput. On the contrary, with the APPD method, the bridge service time allocated to the master with a high traffic load decreases substantially as the degree of the bridge increases. Accordingly, the throughput of a master with a high traffic load will decrease with the increase in the degree of the bridge. This is because, in the APPD method, the bridge service time allocated to the masters is based on link level fairness. That is to say, the chances of the masters getting the usage of the bridge are the same, no matter how heavy the traffic load of the master is. Therefore, the bridge service time of the master with a high traffic load will decrease substantially as the bridge degree increases. Contrarily, in TASS, the master with a high traffic load will have a higher probability to obtain the usage of the bridge due to the QUEUE event. At the same time, TASS will not cause the master with low traffic load to starve, since the master with a low traffic load can obtain the usage of the bridge by the TIME event.

Fig. 10 and Fig. 11 show the impact of the bridge degree on the total activity ratio and the total throughput of the bridge, respectively. The total activity ratio means the ratio of the total bridge service time for all masters to the total simulation time, and the total throughput means the sum of all data packets received per second by the bridge from all masters. Note that the increase of the bridge degree implies an increase of the total traffic load. Thus, the activity ratio and the throughput will increase when the bridge degree increases. As described above, TASS can dynamically switch the bridge to a master according to the traffic load of a master. The master with a high traffic load can have plenty of bridge service time without starving the master



Fig. 9. The impact of the degree of the bridge on the throughput of the master with a high traffic load.



Fig. 11. The impact of the bridge degree on the total throughput.



Fig. 12. The impact of the bridge degree on the average packets delay for a master with a high traffic load (300kbps).

with a low traffic load. The bridge always keeps busy. Thus, the total activity ratio of TASS is superior to that of APPD. As a result, the total throughput of TASS is superior to that of APPD as well. However, in APPD, it can happen that a bridge switches to a master with no packet to send. This will reduce the bridge service time since the bridge may go back to sleep after the sniff slot and the other masters with a high traffic load can not use the bridge, resulting in a low activity ratio as well as a low throughput.

In the following, the impact of the bridge on the average packet delay is investigated. The packet size is 384 bytes in this experiment. Fig. 12 illustrates the average queue packet delay of a master with a high traffic load (300kbps). In comparison with TASS, the packet delay of the APPD method is much higher, especially when the degree of the bridge is larger than 3. It is evident that this is because the master with a high traffic load in APPD cannot obtain enough bridge service time to handle all of its data packets. Thus, the average packet delay will increase substantially as the bridge degree increases. In contrast to APPD, TASS can allocate more bridge service time to the master with a high traffic load and can thereby reduce the packet delay time even further. The average packet delay of TASS rises slightly when the bridge degree is larger than 6. This is because the allocated bridge service time is insufficient to consume all queued data packets.

Fig. 13 shows the impact of the bridge degree on the average queued packets delay of the masters with a low traffic load (60kbps). Since the probabilities of these masters getting usage of the bridge are the same, the packet delay



Fig. 14. The impact of the bridge degree on the number of failed unsniffs.

times of the masters with low traffic loads are almost the same when the bridge degrees vary from 2 to 7. However, in TASS, the packet delay time will increase as the bridge degree increases. This is because the master with a high traffic load can get more bridge service time. This will increase the packet delay time of the masters with low traffic load. Nevertheless, TASS will not cause starvation of the masters with a low traffic load due to the implementation of the TIME event. Although the packet delay time of TASS in the case of a master with a low traffic load is higher than that of APPD, as a whole, the total packet delay time of TASS is still lower than that of APPD.

Fig. 14 illustrates the impact of the bridge degree on the number of failed unsniffs. Due to the lack of the traffic information in APPD, the number of unsniffs in APPD is higher than that in TASS. In APPD, the waiting master that has packets to transmit will try to unsniff the bridge on each sniff slot until it successfully unsniffs the bridge. Consequently, the number of failed unsniffs of APPD will be high. However, in TASS, the waiting masters know how long they can not get the usage of the bridge. Therefore, the waiting master will not unsniff the bridge until its *LT* expires. As a result, the number of failed unsniffs of TASS is reduced accordingly.

The switching of the bridge among piconets is the main reason for guard time waste. The higher the frequency of the bridge switch is, the higher the guard time wastes. Fig. 15 shows the bridge switch frequencies of TASS and APPD for various bridge degrees. Due to the lack of traffic information and the fact that all masters have the same



Fig. 13. The impact of the bridge degree on the average packet delay of masters with a low traffic load (60kbps).



Fig. 15. The impact of the bridge degree on the frequency of bridge switches.



Fig. 16. The impact of the various traffic loads on the total throughput when a bridge connects to three masters.

probabilities of getting usage of the bridge in APPD, the bridge may switch to a master with no packet to send and the bridge switch waste will be high. On the contrary, TASS takes traffic information into consideration. The bridge service time for a master is different depending upon the traffic load of the master. The bridge will not switch blindly among the piconets. In TASS, the average bridge service time of a master with data packets to send will be longer than that in APPD. This implies that the frequency of bridge switch in TASS is lower than that in APPD, as shown in Fig. 15.

In the following, we investigate the impact of the various traffic loads on the total throughput when a bridge connects with three masters. Among the three masters, one master will vary its packet generation rates from 100kbps to 400kbps to 20kbps every 20 seconds and the other two masters will fix their packet generation rates at 100kbps. Initially, the packet generation rate of each master is 100kbps. Fig. 16 illustrates the total throughputs of TASS and APPD, which are obtained from every 1,600 slots (i.e., 1 second). As shown in Fig. 16, both TASS and APPD can reach the maximum throughput in the first 20 seconds since the packet generation rates of the three masters are the same. In the following 20 seconds, the packet generation rate of one master rises to 400kbps. Since APPD does not take traffic information into consideration, it cannot adjust the switch scheduling according to the different traffic load of the master. Consequently, TASS can maintain the maximum total throughput, but APPD cannot. At the last 20 seconds, the packet generation rate of one master is reduced to 20kbps. As the figure shows, TASS can adapt to the real traffic quite rapidly, but APPD requires additional time to adapt to the real traffic load situation. This comes because there are still a lot of data packets queued for the previous 20 seconds in APPD; hence, it needs additional time to consume the queued packets. It is evident that the adaptability of TASS is superior to that of APPD.

#### **CONCLUSIONS** 6

In this paper, we presented an interpiconet scheduling, Traffic-Aware Scatternet Scheduling (TASS) scheme, which can dynamically adjust the bridge service time according to a master's traffic load, reduce the number of failed unsniffs, and further increase the system's throughput. The primary idea of TASS is to allocate the bridge service time to that

master which needs it the most. That is, TASS allocates enough bridge service time to the master with a high traffic load and reduces the bridge switch wastes. At the same time, to avoid excessive transmission delay of the master with a low traffic load, TASS will allocate the bridge service time to a master once that master has waited for a period of time, but no longer than  $W_{thold}$ . Moreover, the masters in the bridgeless phase will reduce their number of failed unsniffs because of the LT. To improve the bridge switch problem, TASS will lengthen the LT after each bridge switch waste and, hence, reduces the number of the bridge switch wastes.

Simulation results revealed that, when the traffic loads of the masters vary, the bridge switch scheduling of TASS is more efficient than that of APPD. In addition, TASS is superior to APPD in adaptability. The number of failed unsniffs of TASS is also fewer than that of APPD. All in all, the performance of TASS is superior to that of APPD, especially under an environment with various traffic loads.

In this paper, TASS has been shown that it can perform well in the case of a bridge shared by multiple piconets. However, it is possible that two or more masters share more than one bridge. Therefore, for the sake of completeness, a comprehensive investigation should be made in the future to find out how TASS performs in all cases, including as a bridge shared by multiple piconets, multiple bridges shared by multiple piconets, or both.

#### **ACKNOWLEDGMENTS**

This work was supported by the National Science Council of the Republic of China under grants NSC 93-2213-E-008-001 and NSC 94-2524-S-032-003.

#### REFERENCES

- Specifications of the Bluetooth System, version 1.1, Bluetooth Special [1] Interest Group, Feb. 2001, http://www.Bluetooth.com.
- [2] J. Bray and C.F. Sturman, Bluetooth: Connect without Cables. Prentice Hall, 2001.
- B.A. Miller and C. Bisdikian, Bluetooth Revealed. Prentice Hall, [3] 2001
- N.J. Muller, Bluetooth Demystified. McGraw-Hill, 2001. A. Capone, M. Gerla, and R. Kapoor, "Efficient Polling Schemes [5] for Bluetooth Picocells," Proc. IEEE Int'l Conf. Comm. (ICC), pp. 1990-1994, June 2001.
- S. Chawla, H. Saran, and M. Singh, "QoS Based Scheduling for [6] Incorporating Variable Rate Coded Voice in Bluetooth," Proc. IEEE Int'l Conf. Comm. (ICC), vol. 4, pp. 1232-1237, June 2001.
- A. Das, A. Ghose, A. Razdan, H. Saran, and R. Shorey, "Enhancing [7] Performance of Asynchronous Data Traffic over the Bluetooth Wireless Ad Hoc Network," Proc. IEEE INFOCOM, Ann. Joint Conf. IEEE Computer and Comm. Soc., pp. 591-600, Apr. 2001.
- N. Glomie, N. Chevrollier, and I. ElBakkouri, "Intereference [8] Aware Bluetooth Packet Scheduling," Proc. IEEE Global TeleComm. Conf. (GLOBECOM), pp. 2857-2863, Nov. 2001.
- [9] M. Kalia, D. Bansal, and R. Shorey, "MAC Scheduling and SAR Policies for Bluetooth: A Master Driven TDD Pico-Cellular Wireless System," Proc. Int'l Workshop Mobile Multimedia Comm., pp. 384-388, 1999.
- M. Kalia, D. Bansal, and R. Shorey, "Data Scheduling and SAR for [10] Bluetooth MAC," Proc. IEEE Vehicular Technology Conf., vol. 2, pp. 15-18, 2000. S. Baatz, M. Frank, C. Kuhl, P. Martini, and C. Scholz, "Bluetooth
- [11] Scatternets: An Enhanced Adaptive Scheduling Scheme," Proc. IEEE INFOCOM, Ann. Joint Conf. IEEE Computer and Comm. Soc., pp. 782-790, June 2002.

- [12] L. Har-Shai, R. Kofman, G. Zussman, and A. Segall, "Interpicnoet Scheduling in Bluetooth Scatternets," Proc. OPNETWORK Conf., Aug. 2002.
- [13] L. Har-Shai, R. Kofman, A. Segall, and G. Zussman, "Load Adaptive Interpiconet Scheduling in Small-Scale Bluetooth Scatternets," *IEEE Comm. Magazine*, vol. 42, no. 7, pp. 136-142, July 2004.
- [14] N. Johansson, F. Alriksson, and U. Jonsson, "JUMP Mode—A Dynamic Window-Based Scheduling Framework for Bluetooth Scatternets," Proc. ACM Int'l Symp. Mobile Ad Hoc Networking and Computing, pp. 204-211, 2001.
- [15] N. Johansson, U. Korner, and L. Tassiulas, "A Distributed Scheduling Algorithm for a Bluetooth Scatternet," Proc. 17th Int'l Teletraffic Congress, Sept. 2001.
- [16] R. Kapoor, A. Zanella, and M. Gerla, "A Fair and Traffic Dependent Scheduling Algorithm for Bluetooth Scatternets," ACM Mobile Networks and Applications, vol. 9, no. 1, pp. 9-20, Feb. 2004.
- [17] M. Kazantzidis and M. Gerla, "On the Impact of Interpiconet Scheduling in Bluetooth Scatternets," Proc. Int'l Conf. Internet Computing (IC2002), vol. 1, pp. 37-43, June 2002.
- [18] V.B. Mišić and J. Mišić, "Adaptive Interpiconet Scheduling in Small Scatternet," ACM SIGMOBILE Mobile Computing and Comm. Rev., vol. 7, no. 2, pp. 45-58, Apr. 2003.
- [19] A. Rácz, G. Miklós, F. Kubinszky, and A. Valkó, "A Pseudo Random Coordinated Scheduling Algorithm for Bluetooth Scatternets," Proc. ACM Int'l Symp. Mobile Ad Hoc Networking and Computing, pp. 193-203, 2001.
- [20] W. Zhang and G. Cao, "A Flexible Scatternet-Wide Scheduling Algorithm for Bluetooth Networks," Proc. 21st IEEE Int'l Performance, Computing, and Comm. Conf., pp. 291-298, 2002.
- [21] J. Kim, Y. Lim, Y. Kim, and J.S. Ma, "An Adaptive Segmentation Scheme for the Bluetooth-Based Wireless Channel," *Proc. 10th IEEE Int'l Conf. Computer Comm. and Networks*, pp. 440-445, 2001.
- IEEE Int'l Conf. Computer Comm. and Networks, pp. 440-445, 2001.
  S. Baatz, M. Frank, C. Kuhl, P. Martini, and C. Scholz, "Adaptive Scatternet Support for Bluetooth Using Sniff Mode," Proc. 26th Ann. Conf. Local Computer Networks, pp. 112-120, Nov. 2001.
- [23] H. Schwetman, "CSIM: A C-Based Process-Oriented Simulation Language," Proc. 1986 Winter Simulation Conf., pp. 387-396, 1986.



Jang-Ping Sheu received the BS degree in computer science from Tamkang University, Taiwan, Republic of China, in 1981 and the MS and PhD degrees in computer science from National Tsing Hua University, Taiwan, Republic of China, in 1983 and 1987, respectively. He joined the faculty of the Department of Electrical Engineering, National Central University, Taiwan, Republic of China, as an associate professor in 1987. He is currently a professor

in the Department of Computer Science and Information Engineering and director of the computer center at the National Central University. He was a chair of the Department of Computer Science and Information Engineering, National Central University, from 1997 to 1999. He was a visiting professor at the Department of Electrical and Computer Engineering, University of California, Irvine, from July 1999 to April 2000. His current research interests include wireless communications, mobile computing, and parallel processing. He was an associate editor of the Journal of the Chinese Institute of Electrical Engineering from 1996 to 2000, the Journal of Information Science and Engineering from 1996 to 2002, and the Journal of the Chinese Institute of Engineers from 1998 to 2004. He is currently an associate editor of the IEEE Transactions on Parallel and Distributed Systems and was a guest editor of the special issue for the Wireless Communications and Mobile Computing journal. He was a program chair of IEEE ICPADS 2002 and a vice program chair of ICPP 2003. He received the Distinguished Research Award of the National Science Council of the Republic of China in 1993-1994, 1995-1996, and 1997-1998 and was the Specially Granted Researcher, National Science Council, from 1999 to 2005. He received the Distinguished Engineering Professor Award of the Chinese Institute of Engineers in 2003. He is a senior member of the IEEE and a member of the ACM and the Phi Tau Phi Society.



**Kuei-Ping Shih** received the BS degree in mathematics from Fu-Jen Catholic University, Taiwan, Republic of China, in June 1991 and the PhD degree in computer science and information engineering from the National Central University, Taiwan, Republic of China, in June 1998. After two years of military obligation, he joined the faculty of the Department of Computer Science and Information Engineering, Tamkang University, Taiwan, Republic of China, as an

assistant professor in 2000. Currently, he is an associate professor in the Department of Computer Science and Information Engineering, Tamkang University. Dr. Shih served as a guest editor for the *International Journal of Wireless and Mobile Computing (IJWMC)* special issue on mobile systems, e-commerce, and agent technology, a program area chair for the *IEEE International Conference on Advanced Information Networking and Applications (AINA)*, 2005, a technical Track Chair for the *IEEE International Conference on Information Technology: Research and Education (ITRE)*, 2005, and a program committee member for several international conferences. Dr. Shih's current research interests include wireless networks, sensor networks, mobile computing, and network protocols design. He has received the research award of National Science Council, Taiwan, 2000. He is a member of the IEEE Computer Society, the IEEE Communication Society.



Shin-Chih Tu received the BS degree in information management in 1999 and the MS degree in mathematic science in 2001, both from Aletheia University, Taiwan. Since September 2001, he has been working toward the PhD degree in the Department of Computer Science and Information Engineering, National Central University, Taiwan. His current research interests include mobile computing, bluetooth radio systems, and ad hoc wireless networks.



**Chao-Hsun Cheng** received the BS degree in computer science and information engineering from National Taiwan Ocean University, in 2001 and the MS degree in computer science and information engineering from National Central University, Taiwan, in 2003. His current research interests include wireless networking, bluetooth radio systems, and ad hoc wireless networks.

▷ For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.