A Traffic-Aware Scheduling for Bluetooth Scatternets[1]

Jang-Ping Sheu†, Chao-Hsun Cheng†, Kuei-Ping Shih‡, and Shin-Chih Tu†

†Dept. of Computer Science and Information Engineering, National Central University, Chungli, Taiwan
‡Dept. of Computer Science and Information Engineering, Tamkang University, Tamshui, Taiwan
e-mail: †sheujp@csie.ncu.edu.tw, ‡kpshih@mail.tku.edu.tw

Abstract: **Bluetooth scatternet is a set of piconets interconnected via bridge devices. A good inter-piconet scheduling is necessary for the bridge devices to switch among the piconets they participated. This paper proposes an inter-piconet scheduling algorithm, Traffic-Aware Scatternet Scheduling (TASS), for bridges in bluetooth scatternets. According to the traffic information of all masters that the bridge is connected, TASS can adaptively switch the bridge to the masters with high traffic loads and increase the usage of the bridge. In addition, TASS can reduce the number of failed unsniffs and the overhead of the bridge switch wastes to further increase the overall system performance. Simulation results show that TASS outperforms the existing inter-piconet scheduling in network throughput and adaptability for various traffic loads.**

1. Introduction

Bluetooth is a low cost, low power, and short-range radio technology used for wireless personal area networks (PANs) [1]. A *piconet* is a basic structure in Bluetooth, which is constructed in an ad-hoc fashion by one master and up to 7 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. The slave connecting to two or more masters is termed as 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 piconets it is connected in a time-sharing fashion. For ease of explanation, the master that the bridge is serving is called the *serving master* and the other masters that the bridge is connected but not in service are called *waiting masters*.

The scheduling of a bridge switching among piconets is also referred as *inter-piconet scheduling*. Obviously, an ill-considered scheduling may cause severe system performance degradation. However, inter-piconet scheduling is not specified in Bluetooth specification. Thus, the inter-piconet scheduling should be developed and be well designed to help the bridge efficiently switch among piconets. On the other hand, the *intra-piconet scheduling* is referred as the scheduling of a master serving the slaves which the master connects. *Polling* is a general scheme adopted for intra-piconet scheduling. There have been existed a lots of researches on intra-piconet scheduling in the literature [5, 6, 7, 8, 9, 10, 11]. Intra-piconet scheduling is out of the scope of the paper.

Recently, a number of researches on inter-piconet scheduling have been proposed [12, 13, 14, 15, 16]. In [12] an Adaptive Presence Point Density scheme (called APPD) for inter-piconet scheduling is proposed. In APPD, a credit value is attached to each connection of the bridge that is established. According to the credit values, the bridge can decide whether to switch to another piconet or not. However, the decision of the switching is controlled by the bridge, without negotiating with the serving master. It may result from one or more packet losses. On the other hand, to preserve the fairness among the connections, in APPD, the probabilities of masters getting the usage of the bridge are the same. Nevertheless, it may lead to a bottleneck since the master with 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 APPD as well.

A novel inter-piconet scheduling protocol, Traffic-Aware Scatternet Scheduling (TASS), is presented in the paper. In TASS, a bridge will switch to another piconet only when the current serving master notices it to switch off. As a result, there will not be any packet loss when the bridge switches to another piconet. Each master will maintain a scheduling table. The table records all masters' traffic information and their bridge usage statuses, such as how long the master has waited for the usage of the bridge, how long the master may 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. 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 can be much reduced. 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 piconets that it is connected according to the master's traffic loads and waiting time. Simulation results demonstrate that TASS outperforms from the APPD with higher network throughput and better flexibility in the various traffic loads environment.

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

2. Challenges of the Inter-Piconet 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, *sniff*, *hold*, and *park* modes. In general, sniff mode is used for a bridge to switch among piconets regularly and periodically. A device in sniff mode only wakes up periodically in pre-arranged sniff

[1]This work was supported by the Ministry of Education, the Republic of China, under Grant A-92-H-FA07-1-4 (Learning Technology).

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 pre-scheduled 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 one packet loss. Hence, too many failed unsniff will significantly degrade the performance of the piconet, even that of a scatternet.

A bridge will switch among piconets that it is connected in a time-sharing fashion. An inter-piconet scheduling is needed for a bridge to switch among piconets. However, the scheduling is not straightforward. The followings are the challenges to be considered by an inter-piconet scheduling.

- **The variance of traffic loads.** Since the traffic of a network must be variable, therefore, an inter-piconet scheduling must have the flexibility to dynamically adjust the scheduling to meet the variant traffic loads [12].
- **The tradeoff between the throughput and the transmission delay.** To achieve the maximum throughput of a scatternet, we would like to allocate more bridge service time to the master with high traffic loads. However, it may increase the transmission delay of the masters with low traffic loads [17]. Therefore, in addition to increase the throughput, to reduce transmission delay is also needed to be considered.
- **The frequency of a bridge switching among piconets.** When a bridge switches to a new piconet, the bridge may not match to the timing of the new piconet. 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 reduce the guard time waste as well [18].

In general, the communication between two devices always lasts for a certain time. The traffic is with the temporal locality. Consequently, inter-piconet scheduling of a bridge can utilize the historical information to enhance the efficiency of the scheduling.

3. Traffic-Aware Scatternet Scheduling (TASS) Protocol

In this section, the system model of the paper is presented in Section 3.1 and the TASS protocol is described in Section 3.2.

3.1. System Model

TASS is operated on a constructed scatternet. The connection between a master and a slave is ACL link only. 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 will maintain a *scheduling table*, which indicates the traffic information of all masters that the bridge is connected. 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. A scheduling table is shown in Table 1, where MID represents the identity of the master and the other fields are described as follows.

- QCT (Queue Consuming Time): the time that a link still needs the bridge to serve,
- LT (Lost Time): the 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, and
- α: the historical information that, in average, the traffic generation rate per slot between the master and the bridge.

The unit of the time described above is slot.

QCT is defined as the time that a link still needs the bridge to serve. It means the time that all the data packets in the queues of the master and the bridge need to be transmitted completely. 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. Therefore, the master can compute the QCT.

LT is defined as the time that a master can not get the usage of the bridge. Since the scheduling table has the QCTs of all masters that the bridge is connected, when the serving master has to release the usage of the bridge, according to QCTs, it could predict the time that it may lose the bridge after it releases the usage of the bridge. The time is called LT. LT could 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 could reduce the number of the failed unsniffs.

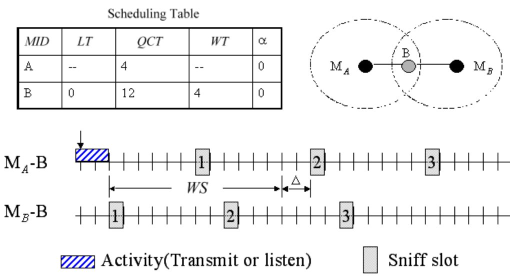
WT is the time that the master has been waiting for the usage of the bridge since it released the usage of the bridge.

α represents the history of the traffic loads. Since the decision of the master releasing the bridge depends much on the value of QCT, the precision of QCT will influence the performance of TASS. Therefore, to obtain QCT precisely, the history of traffic loads is counted to evaluate QCT due to the temporal locality of the traffic. Let ρ be the increment of the traffic in queue during a fixed time period, say T. The queue agent responds to maintain ρ. Thus, α can be obtained as $\alpha = \frac{\rho}{T}$. α will be computed for every T time period. After α is obtained, the queue agent will reset ρ to zero. For example, suppose $T = 20$. For some master and the bridge, they generate 2 DH5 packets in T. So $\rho = 10$ and $\alpha = 10/20 = 0.5$. It means that the QCT of the master-bridge link will increase 0.5 packets per slot in average. When the serving master has to release the usage of the bridge, it would record the α in the scheduling table. Hence, when the new serving master gets the usage of the bridge, it could evaluate the QCT more precisely for some waiting master.

When the serving master i has to release the usage of the bridge, it will find a candidate to be the new serving master,

MID	LT	QCT	WT	α
⋮	⋮	⋮	⋮	⋮

Table 1: Scheduling Table.



MID	*LT*	*QCT*	*WT*	α
A	--	4	--	0
B	0	12	4	0

Figure 1: An example of LT.

say j, and update the LT_i. The serving master i first finds the minimum LT_j from the scheduling table, for some j. If there are more than one minimum LT, then it selects the one with the maximum WT. It 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 out-of-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 the master j. As a result, we can use α_j to estimate the QCT_j roughly. Therefore, at least, the time that the serving master i will not get the usage of the bridge can be obtained as below, say WS.

$$WS = QCT_j + \alpha_j * WT_j$$

On the other hand, to avoid the excessive transmission delay of the waiting masters, a waiting threshold (W_{thold}) is used to limit the transmission delay. If the WS is bigger than W_{thold}, then WS is set to W_{thold}.

Due to a master can communicate with a bridge only on the sniff slots, WS may not coincide with the sniff slots. So we have to add an offset, Δ, to match the sniff slot exactly. Δ can be calculated as follows.

$$\Delta = T_{Sniff} - ((WS + 2) \bmod T_{Sniff}).$$

Therefore, the time that the serving master i will not get the bridge after it releases the usage of the bridge is

$$LT_i = WS + \Delta.$$

Take Figure 1 as an example, where master A is the serving master and $T_{Sniff} = 8$. Assume that master A is going to release the usage of the bridge on the first time slot. $WS = QCT_B + \alpha_B * WT_B = 12$ ($QCT_B = 12$, $\alpha_B = 0$, and $WT_B = 4$). $\Delta = 8 - ((12 + 2) \bmod 8) = 2$. Hence, $LT_A = WS + \Delta = 12 + 2 = 14$.

3.2. The Protocol

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

Bridge Phase

If a serving master i gets the usage of the bridge, it first finds the minimum LT_j from the scheduling table, for some j. According to this information, master i can realize how much time it can use the bridge freely. Besides, master i should be responsible for the maintenance of the scheduling table. That is, the serving master i should add 1 to each WT and subtract 1 from each LT in the scheduling table per slot. When $LT_j = 0$, master i must check if it has to release the usage of the bridge to the waiting master j. When the released condition is satisfied, the serving master i has to release the usage of the bridge to the waiting master j. The serving master i then performs the serving master part of the *Bridge Release Procedure*. As described above, once the serving master i intends to release the usage of the bridge, the serving master i will calculate LT_i by means of the scheduling table. After the LT_i is calculated, master i updates the LT_i into 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 the master i is turned from the serving master to the waiting master. Thus, master i then performs the *bridgeless phase* afterward. The bridge receiving the scheduling table will perform the *Bridge Release Procedure* as well, but the bridge part. The bridge then waits for being unsniffed by the new serving master j and maintains the scheduling table during this waiting period. The maintenance is that the bridge would record the time slot count (sc) since it got the scheduling table to receive the unsniff message from the new serving master. When the bridge is unsniffed by the new serving master, it would subtract sc from each LT, add sc to each WT in the scheduling table, and then transmit the scheduling table to the new serving master. The *bridge phase* and *bridge release procedure* are demonstrated in Algorithms 1 and 2, respectively.

There are two conditions that the serving master i has to release the usage of the bridge to the waiting master j.

1. $WT_j > W_{thold}$, (TIME event),
2. $QCT_j + \alpha_j * WT_j > QCT_i + QC_{thold}$, (QUEUE event).

The first condition implies that the master j has been waiting for the usage of the bridge over the W_{thold}. The second condition implies that all the data needed to be transmitted completely between master j and the bridge is bigger 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 close to each other.

The first condition is to avoid the excessive transmission delay of the waiting master. The released event triggered by this condition is termed as TIME event. The second condition is to allocate more service time to the link with high traffic loads. The released event triggered by this condition is termed as QUEUE event. If none of the two conditions is satisfied, then the serving master i could keep using the bridge. This is termed as EXTEND event. It is worth mentioning that an EXTEND event will result in a failed unsniff at the waiting master which has the highest possibility to get the usage of the bridge in the near future (here implies the waiting master j). However, EXTEND event implies that the traffic loads between the serving master i and the bridge is higher than those between the waiting master j and the bridge.

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 is expired. So master j will try to unsniff the bridge on the sniff slots in the future. Therefore, the LT_j in the scheduling table of the serving master i must reset to T_{Sniff}.

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
Step 4:
  Choose a waiting master $j$ with $LT_j = 0$.
  if there are more than one waiting master with $LT = 0$ then
    Select the waiting master $j$ with the largest $WT$ and the other $LT$s 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.
```

Bridgeless Phase

The waiting masters that do not get the usage of the bridge will perform the *bridgeless phase*. For some waiting master, say j, according to the LT_j that was calculated when the master j had released the usage of the bridge, the master j could realize the time it might not get the usage of the bridge. Therefore, the waiting master j won't unsniff the bridge during LT_j. Hence, it could reduce the number of failed unsniffs. Algorithm 3 is the operations that all waiting masters have to perform per slot.

Algorithm 2 Bridge Release Procedure.

```
The part to be executed by the serving master.
  {The serving master $i$ deciding to release the usage of the bridge will perform the following operations.}
  Step 1:
    Calculate $LT_i$.
  Step 2:
    Update $LT_i$ and reset $WT_i$ to zero in the scheduling table.
  Step 3:
    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:
    Accumulate the time slot count $sc$ since the bridge gets the scheduling table from the serving master $i$ to receive the unsniff message from the new serving master $j$.
  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 $j$.
```

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 unsniff the bridge on the following sniff slots.}
end if

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

4. Bridge Switch Problem

If a bridge switches to a piconet, but the serving master has no data to the bridge. A Poll-Null sequence event happens. It is a *bridge switch waste*. However, if the Poll-Null sequence event is happened in the sniff slot between the master and the bridge, it implies that this switch of the bridge is waste once. It would lead the bridge to go back to sleep and the other waiting masters would be unable to unsniff the bridge successfully. This will reduce the usage of the bridge. In TASS, in order to reduce the number of bridge switch wastes, LT will be increased as long as the bridge switch waste happens. 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 from the bridge a DH1 ACK packet that no data is included, except the scheduling table, the serving master will regard the ACK packet as a Null packet. Thus, LT of the serving master will be increased accordingly. An additional time (PNT, Poll-Null time) is added to the LT_i. As a result, the LT_i will be lengthened after the Poll-Null sequence event. Thus, it could reduce the number of the bridge switch wastes. A Poll-Null event counter is used to record the times of successive bridge switch wastes, and the 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 as below.

$$LT_i = LT_i + PNT.$$

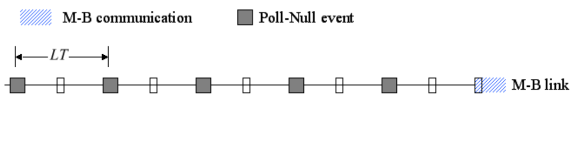
LT is getting larger while the bridge switch wastes happen often. To avoid excessive transmission delay while master i has data to the bridge, an upper bound, MaxLT, is to limit the increasing of LT.

In the following, we will give an example to demonstrate the procedure to reduce the bridge switch wastes. Figure 2(a) shows the result of no improvement of bridge switch wastes and Figure 2(b) shows the result with the improvement. In Figure 2(b), the numbers below the gray squares indicate the values of the Poll-Null event counter, and zero implies 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 Figure 2(b), the LT will be lengthened after each bridge switch waste. The more the successive bridge switch wastes are, the longer the LT is. Consequently, in Figure 2, we can obviously observe that it will reduce 3 times of bridge switch wastes in the result with the improvement of bridge switch wastes comparing with the result without the improvement.

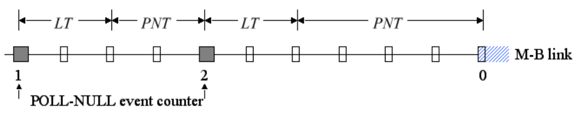
5. Simulation Results

CSIM is the simulator we use to verify the feasibility of the proposed protocol. In the experiment, $T_{Sniff} = 20$, $QC_{thold} = 10$, $W_{thold} = 50$, and Max$LT = 100$, where the unit is slot. 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 topology on which the experiment performed is shown in Figure 3. In the topology, there are 7 piconets sharing a bridge. To do so is because we are interested in the influence of the inter-piconet scheduling on the scatternet performance. On the same conditions, the comparisons between TASS and



(a) No improvement.



(b) With improvement.

Figure 2: The comparison of the results without and with the improvement of bridge switch waste.

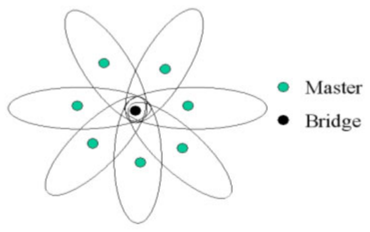


Figure 3: Simulation topology.

APPD (Adaptive Presence Point Density [12]) on throughput, activity ratio, packet delay, and the number of failed unsniffs are presented as well. The packet generation rates of the masters are in a constant bit rate (CBR). Among these masters, the packet generation rate of one master is fixed on 300kbps and those for the others are fixed to 60kbps. A high packet generation rate implies that the master would need more bridge service time. The bridge does not generate any packets at all and the destinations of all packets are to the bridge. The simulation time is 200 seconds.

Figures 4 and 5 show the impact of the degree of the bridge on the activity ratio and the throughput of the master whose packet generation rate equals 300kbps, respectively. The activity ratio means the ratio of the total bridge service time of the master whose packet generation rate equals 300kbps to the total simulation time. The throughput is evaluated by the data packets received by the bridge per second. Obviously, TASS can allocate more bridge service time to the master with high traffic loads. The master with high traffic loads can almost obtain the maximum throughput. On the contrary, in APPD, the bridge service time allocated to the master with high traffic loads decreases seriously as the degree of the bridge increases. Accordingly, the throughput of the master with high traffic loads will decrease when the the degree of the bridge increases as well. It is because that, in APPD, the bridge service time allocated to the masters is based on the link level fairness. That is, the chances of the masters getting the usage of the bridge are the same, no mater how heavy the traffic load of the master is. Therefore, the bridge service time of the master with high traffic loads will decrease seriously as the bridge degree increases. Contrarily, in TASS, the master with high traffic loads will have higher probability to obtain the usage of

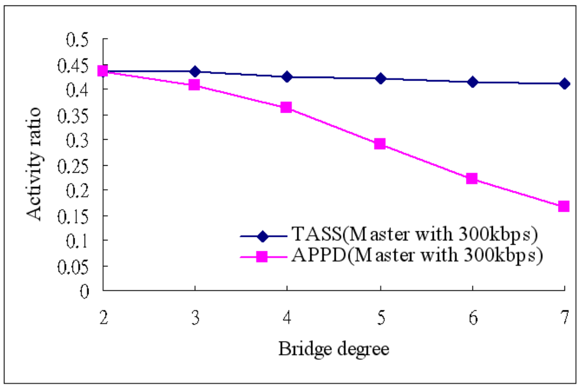


Figure 4: The impact of the degree of the bridge on the bridge activity ratio of the master with high traffic loads.

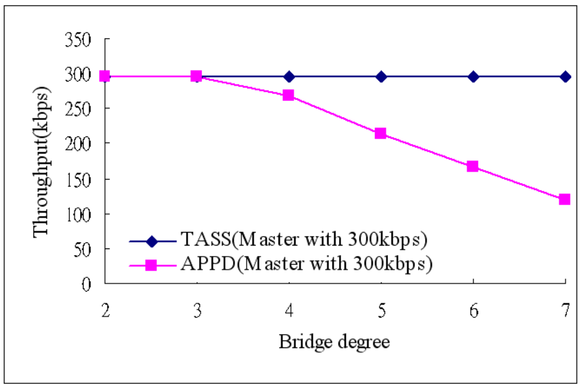


Figure 5: The impact of the degree of the bridge on the throughput of the master with high traffic loads.

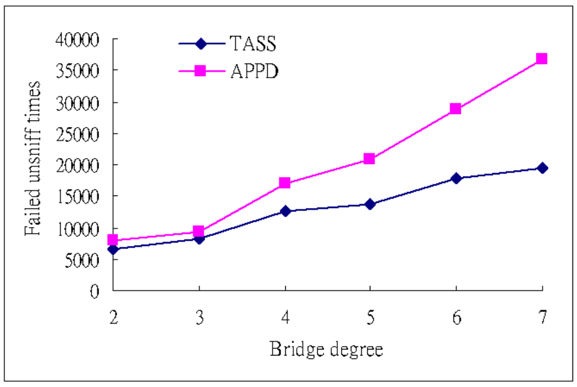


Figure 6: The impact of the bridge degree on the number of failed unsniffs.

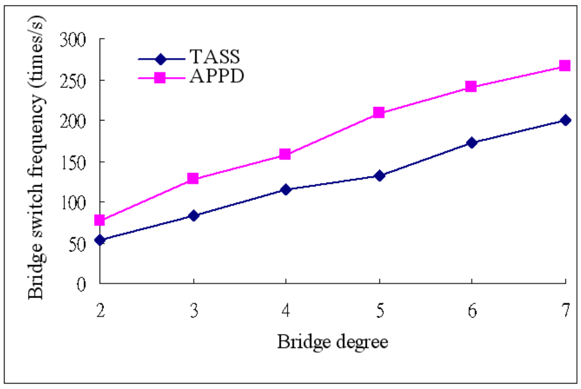


Figure 7: The impact of the bridge degree on the frequency of bridge switches.

the bridge due to QUEUE event. On the other hand, TASS will not cause the master with low traffic load to starve since the master with low traffic load can obtain the usage of the bridge by TIME event.

Figure 6 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. Thus, the number of failed unsniffs of APPD would be high. However, in TASS, the waiting masters know how long it can not get the usage of the bridge. Therefore, the waiting master would 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 resulting in the guard time waste. The higher the frequency of the bridge switch is, the more the guard time wastes. Figure 7 shows the bridge switch frequencies of TASS and APPD for various bridge degrees. Due to the lacks of the traffic information and the same probabilities of the master getting the usage of the bridge in APPD, the bridge may switch to a master with no packet to send and the bridge switch waste would be high. On the contrary, TASS takes traffic information into consideration. The bridge service time for a master would be different and depend upon the traffic loads of the master. The bridge would not switch among the piconets blindly. In TASS, the average bridge service time of the master that has data packets to send will be longer than that in APPD. This implies that the frequency of bridge switch in TASS will be lower than that in APPD, as shown in Figure 7.

In the following, we will investigate the impact of the various traffic loads on the total throughput when a bridge connects with 3 masters. Among the three masters, one master will vary its packet generation rate from 100kbps, 400kbps, to 20kbps every 20 seconds and the other two masters will fix their packet generation rates to 100kbps. Initially, the packet generation rate of each master is 100kbps. Figure 8 illustrates the total throughputs of TASS and APPD, which are obtained from every 1600 slots (i.e. 1 second). As shown in Figure 8, 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 can not adjust the switch scheduling according to different traffic loads of masters. Thus, TASS can still keep the maximum total throughput, but APPD can not. 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 rapidly, but APPD still needs some time to adapt to the real traffic loads. Since there are still a lot of data packets queued at the previous 20 seconds in APPD, hence, it needs additional time to consume the queued packets. Therefore, the adaptability of TASS is superior to that of APPD.

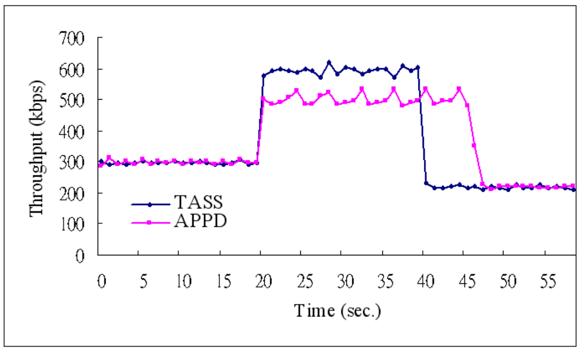


Figure 8: The impact of the various traffic loads on the total throughput when a bridge connects to three masters.

6. Conclusions

In this paper, we presented an inter-piconet scheduling, Traffic-Aware Scatternet Scheduling (TASS), which can dynamically adjust the bridge service time according to the master's traffic loads, reduce the number of failed unsniffs, and further increase the system throughput. The primary idea of TASS is to allocate the bridge service time to the master which needs the most. That is, TASS would allocate enough bridge service time to the master with high traffic loads and reduce the bridge switch wastes. On the other hand, to avoid excessive transmission delay of the master with low traffic loads, TASS would allocate the bridge service time to a master once the master has waited for a period of time, no longer than W_{thold}. Moreover, the masters in *bridgeless phase* will reduce the number of failed unsniffs because of the LT. To improve the bridge switch problem, TASS will lengthen the LT after a bridge switch waste and hence can reduce the number of the bridge switch wastes.

Simulation results reveal that when the traffic loads of the masters are various, the bridge switch scheduling of TASS is more efficient than that of APPD. In addition, TASS is better than APPD in adaptability. The number of failed unsniffs of TASS is also fewer than that of APPD. Comprehensively, the performance of TASS outperforms against APPD, especially under the environment with various traffic loads.

REFERENCES

[1] *Specifications of the Bluetooth System, ver. 1.1*, Bluetooth Special Interest Group, Feb. 2001, http://www.Bluetooth.com.

[2] J. Bray and C. F. Sturman, *Bluetooth Connect Without Cables*. Prentice Hall PTR, 2001.

[3] B. A. Miller and C. Bisdikian, *Bluetooth Revealed*. Prentice Hall PTR, 2001.

[4] N. J. Muller, *Bluetooth Demystified*. McGraw-Hill Companies Inc., 2001.

[5] A. Capone, M. Gerla, and R. Kapoor, "Efficient polling schemes for bluetooth picocells," in *Proceedings of the IEEE International Conference on Communications (ICC)*, June 2001, pp. 1990–1994.

[6] S. Chawla, H. Saran, and M. Singh, "QoS based scheduling for incorporating variable rate coded voice in bluetooth," in *Proceedings of the IEEE International Conference on Communications (ICC)*, vol. 4, June 2001, pp. 1232 –1237.

[7] A. Das, A. Ghose, A. Razdan, H. Saran, and R. Shorey, "Enhancing performance of asynchronous data traffic over the bluetooth wireless ad-hoc network," in *Proceedings of the IEEE INFOCOM, the Annual Joint Conference of the IEEE Computer and Communications Societies*, Apr. 2001, pp. 591–600.

[8] N. Glomie, N. Chevrollier, and I. ElBakkouri, "Intereference aware bluetooth packet scheduling," in *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM)*, Nov. 2001, pp. 2857–2863.

[9] M. Kalia, D. Bansal, and R. Shorey, "MAC scheduling and SAR policies for bluetooth: A master driven TDD pico-cellular wireless system," in *Proceedings of MoMuC*, 1999, pp. 384–388.

[10] ——, "Data scheduling and SAR for bluetooth MAC," in *Proceedings of IEEE Vehicular Technology Conference*, vol. 2, 2000, pp. 15–18.

[11] Y. Liu, Q. Zhang, and W. Zhu, "A priority-based MAC scheduling algorithm for enhancing QoS support in bluetooth piconet," in *IEEE International Conference on Communications, Circuits and Systems and West Sino Expositions*, vol. 1, July 2002, pp. 544 –548.

[12] S. Baatz, M. Frank, C. Kuhl, P. Martini, and C. Scholz, "Bluetooth scatternets: An enhanced adaptive scheduling scheme," in *Proceedings of the IEEE INFOCOM, the Annual Joint Conference of the IEEE Computer and Communications Societies*, June 2002, pp. 782–790.

[13] A. Rácz, G. Miklós, F. Kubinszky, and A. Valkó, "A pseudo random coordinated scheduling algorithm for bluetooth scatternets," in *Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2001, pp. 193–203.

[14] N. Johansson, F. Alriksson, and U. Jonsson, "JUMP mode - a dynamic window-based scheduling framework for bluetooth scatternets," in *Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2001, pp. 204–211.

[15] N. Johansson, U. Korner, and L. Tassiulas, "A distributed scheduling algorithm for a bluetooth scatternet," in *Proceedings of the 17th International Teletraffic Congress*, Sept. 2001.

[16] W. Zhang and G. Cao, "A flexible scatternet-wide scheduling algorithm for bluetooth networks," in *Proceedings of the 21st IEEE International Performance, Computing, and Communications Conference*, 2002, pp. 291–298.

[17] J. Kim, Y. Lim, Y. Kim, and J. S. Ma, "An adaptive segmentation scheme for the bluetooth-based wireless channel," in *Proceedings of the 10th IEEE International Conference on Computer Communications and Networks*, 2001, pp. 440–445.

[18] S. Baatz, M. Frank, C. Kuhl, P. Martini, and C. Scholz, "Adaptive scatternet support for bluetooth using sniff mode," in *Proceedings of the 26th Annual Conference on Local Computer Networks*, Nov. 2001, pp. 112–120.