ASP: An Adaptive Energy-Efficient Polling Algorithm for Bluetooth Piconets

Mark Perillo, Wendi B. Heinzelman
Department of Electrical and Computer Engineering
University of Rochester
Rochester, NY 14627
{*perillo, wheinzel*}*@ece.rochester.edu*

Abstract

Among the various wireless technologies available today, Bluetooth stands out as the most promising for use in low-power sensor networks. This is especially true for large networks of sensors with low data rates, where Bluetooth's low power Hold, Sniff, and Park modes could be utilized during periods of low activity to reduce power consumption. One way that these modes can be exploited to reduce energy consumption is by using an intelligent polling scheme to control intra-piconet traffic. We introduce a novel polling algorithm called Adaptive Share Polling (ASP) that is designed to perform well when the network consists of sources sending short data packets at constant rate that may fluctuate over time. In ASP, the scheduler at the master implicitly learns the share of the bandwidth that needs to be allocated to each of the slaves in order to meet the latency and/or power requirements of the application. Our simulation results show that significant power savings can be achieved in slaves whose data rates do not approach capacity of a Bluetooth piconet, and in masters when the composite data rate of the piconet does not approach capacity.

1 Introduction

Sensor networks have recently attracted a great deal of attention as a means for monitoring an environment for a variety of applications. Since many sensors typically derive their power from either batteries or their surrounding environment, these networks are often characterized by very tight energy constraints. When designing a communications protocol for a sensor network, such energy constraints should be considered so that the lifetime of the network can be maximized. Bluetooth is an emerging wireless communications standard that was originally designed for cable replacements, but whose low power consumption makes it ideally suited for sensor networks as well. The Bluetooth specification leaves a number of areas open, including the manner in which a master polls the slaves in its piconet. However, the choice of the polling method is critical in minimizing power consumption by the sensors.

We introduce a new polling algorithm called Adaptive Share Polling (ASP) that is designed to reduce power consumption when traffic patterns consist of short packets from a constant bit rate application. The ideal polling scheme for such traffic patterns would require the master to poll a slave only if there is data queued at the slave and immediately after that data has arrived at the slave's queue, while allowing the nodes to sleep for the remainder of the time. However, packets seldom arrive with exactly constant or predictable inter-arrival times, so even if the exact packet rate and the most recent packet's arrival time were known, it would be difficult to predict precisely when the next packet will arrive at the slave's queue. Another problem can be observed when considering an increase in packet arrival rate. If the master polls only at the exact necessary rate, it will observe successful polls (defined as data sent in response to a poll) all of the time both before and after the rate increase, making it impossible to detect fluctuations in offered traffic load. Instead, the master needs to poll at a rate slightly above the necessary rate in order to detect these fluctuations. In this case, the polling success rate remains below, but close to, 100%. If the success rate becomes too high, large latency will be observed on the network, since there will be longer delays between polls. If the success rate becomes too low, the slave will be polled more often than it needs to be and energy will be wasted. This energy versus latency tradeoff is very common in polling networks.

ASP balances this tradeoff by ensuring that poll success rate remains within a pre-specified range. In ASP, the scheduler at the master implicitly learns the share of the bandwidth that needs to be allocated to each of the slaves in order to meet the latency and/or power requirements of the application. ASP is capable of adaptively changing the allocated shares based on observations of polling successes

and failures. This algorithm is especially beneficial in networks consisting of multi-rate sensors, where power can be saved in both the master and the slaves by decreasing the number of times the master polls the nodes, without requiring explicit cross-layer communication. ASP requires a slight modification to the Bluetooth standard - an LMP (Link Manager Protocol) message called Conditional Hold.

The system model that we have designed for is one in which a number of multi-rate sensors send information at a constant bit rate with small packet sizes to a single data collector acting as piconet master. On a larger scale, many of these networks could be connected in a hierarchical way to create a large network with a tree topology. A local data collector may be able to aggregate data before relaying to another level. With varying correlation of the collected data, this data collector would act as a multi-rate data source itself.

The rest of this paper is organized as follows. Section 2 addresses related work. Section 3 offers a brief summary of the Bluetooth standard necessary for understanding this paper. Section 4 explains the ASP scheduling algorithm. Section 5 offers predictions of the power consumption and average packet delay in simple networks. Section 6 presents our simulation results with some analysis. Section 7 concludes and details plans for future work.

2 Related Work

A number of polling algorithms for controlling intra-piconet traffic in Bluetooth networks have already been proposed. Many of these have focused on efficient use of the available bandwidth of a piconet, while not considering energy efficiency as a metric [3] [4] [7] [10] [11]. Recently, more algorithms have been designed with power constraints considered. Garg *et al.* took advantage of the Sniff mode to propose several power-saving polling schemes [9]. These schemes consist of varying the sniff interval and serving time based on the master-slave pair slot utilization. Chakraborty *et al.* proposed several scheduling algorithms, among them Adaptive Probability based Polling Interval (APPI) with Fixed Resolution and Adaptive Resolution [6]. APPI uses the Sniff mode by periodically sleeping after a burst of traffic has ended. While this algorithm performs well for bursty traffic, we suspect that it is not as well-suited for more constant rate traffic patterns such as those that we are considering. Predictive Fair Polling, proposed by Yaiz *et al.*, is similar to ASP in that it contains a mechanism that estimates traffic demand for each slave [15]. PFP uses this estimation to determine the probability that a slave has data queued. PFP also estimates a fair share of the bandwidth that each slave should be given. At each polling opportunity, the master calculates an urgency metric for each slave based on the probability that it has queued data and the fraction of its fair share that it has been given. The slave with the highest urgency metric is then polled.

Zhu *et al.* recently proposed a polling algorithm called APCB (Adaptive Power Conserving service discipline for Bluetooth) that we were unaware of at the time ASP was designed [16]. Like ASP, APCB estimates traffic rates based on observations and adapts polling intervals accordingly, while putting slaves to sleep in Hold mode during periods when flows involving the slaves are not anticipated to be active. In APCB, power tuning is performed by varying a parameter whose value determines how rapidly a flow's rate changes. In ASP, the power tuning is performed by the choice of a target success range which determines how much more frequently the slave is polled than what is necessary.

Another common polling method that is used to schedule intra-piconet traffic is Deficit Round Robin (DRR), proposed by Shreedhar *et al.* [14]. DRR works similar to a simple round robin scheduler except that each node is limited to a given time quantum during a given round. If a queued packet is larger than this quantum, the queue is passed over until the next round, when its quantum is added to the quantum that it was forced to sacrifice during the previous round. DRR was implemented as the intra-piconet traffic scheduler in the BlueHoc software releases that were used in our simulations [1] and is used in this paper for comparison with ASP.

3 Background

Bluetooth is a fast frequency hopping time-division duplex standard that was originally designed to act as a cable replacement in Personal Area Networks (PANs) [2]. The simplest Bluetooth network, a piconet, consists of one master device and up to seven slave devices. Bluetooth time slots have a length of 625 μsec, which is also the period of the frequency hop pattern. All traffic in a piconet is directed by the master, which may begin transmission only on even slots. Slaves may begin transmission only on the odd slots immediately following slots during which they have been addressed by the master. Transmissions by the master (or slave) may last for 1, 3, or 5 slots. When multi-slot packets are used, the slave (or master) must wait until the end of this packet before responding to the poll (or polling the next slave). The Bluetooth standard does not specify or even suggest the manner in which a master should poll slaves.

One of the benefits of the Bluetooth standard is that it allows devices that do not anticipate having data to go inactive and turn off their radios to save energy in a low-power state. The Bluetooth standard specifies three modes that can be used to reduce power consumption - Sniff, Park,

and Hold. In Hold mode, which is used in ASP, the master and slave agree that the slave will not be addressed for a given number of slots. The slave enters Hold mode after exchanging LMP messages with its master. A slave can enter Hold mode by being forced by the master, forcing itself, or negotiating with the master. In ASP, we propose another way to enter Hold mode - a "Conditional Hold" message from the master forcing the slave into Hold mode for one of three possible intervals.

4 Adaptive Share Polling (ASP)

The motivation behind Adaptive Share Polling is that we would like to eliminate two situations that are unavoidable in any persistent polling method. The first situation occurs when there are a large amount of exchanges where the master polls a slave and the slave replies with a Null packet. For the remainder of this paper, we will refer to this situation as an unsuccessful poll. ASP solves this problem by requiring the master to track the share of the bandwidth needed by each slave in order to keep unsuccessful polls at a predetermined level. The second situation that we would like to avoid is when slaves receive many access codes and headers for packets not destined for them. ASP solves this problem by requiring the master to send a "Conditional Hold" message, which tells a slave to sleep for an amount of time which is dependent on the length of the slave's reply.

4.1 Adaptive Share Allocation

We define success ratio as the ratio of slots used for successful polls (exchanges that result in successful slave-to-master data transmission) to total slots used for polls for a certain slave. For a slave using single slot packets, it is simply the ratio of successful polls to total polls. In the case of a slave using multi-slot packets, this is different since a successful poll might not occupy the same number of slots as an unsuccessful poll. Target success range is defined as the range of success ratios that a slave should remain within for the duration of its connection to the master. A master implementing ASP scheduling maintains a target success range for each slave. The master will schedule traffic so that all slaves' success ratios always remain within this range. The choice of target success range is critical. If the range is chosen to be very high, it may be hard to recognize fluctuations in offered load. If the range is chosen to be very low, the master will poll the slave more often than what is necessary, resulting in a waste of power and bandwidth resources at both the master and the slave.

The piconet master constantly tracks the current share - the fraction of available slots that are used for communication - of each active slave. A slave's share is chosen so that its success ratio remains within its target success range. We choose to assign share to each slave rather than polling interval because it is easy to set the combined share of the piconet to a value less than one, ensuring that we do not over allocate resources. Also, this is helpful in maintaining fairness since different slaves can use different packet sizes to send data.

During an observation window of N_{normal} polls (or slot pairs if multi-slot packets are used), the master tracks the success ratio for a slave. This ratio will be referred to as the recent success ratio (RSR). This short-term success ratio is smoothed over time and stored as the long-term success ratio (LTSR). The LTSR is updated after each observation window through a smoothing function

$$\begin{aligned} LTSR_i &= \alpha_{LTSR} LTSR_{i-1} + (1 - \alpha_{LTSR}) RSR_i \\ &= (1 - \alpha_{LTSR}) \sum_{k=0}^{\infty} \alpha_{LTSR}^k RSR_{i-k} \end{aligned} \quad (1)$$

Here, a_{LTSR} acts as a smoothing parameter and its value should depend on the expected burstiness of the traffic. The choice of this smoothing method is arbitrary and the use of other methods may in fact result in enhanced performance. When the RSR and LTSR deviate from the designated success range, the share for a node may be adjusted in order to bring the RSR and LTSR back inside the target range. To provide stability in our system, we include a factor $alpha_{share}$ (< 1), which is the maximum factor by which the master can increase or decrease a slave's share during a single observation window. This parameter is included in order to avoid adjusting share by too much when an abnormally high or low RSR is the result of jitter in the inter-packet arrival times. All possible ranges for RSR and LTSR and the actions taken for each situation are given in Table 1. Ranges are given with respect to the target success range (TSR = $\{TSR_{min}, TSR_{max}\}$). TSR_{avg} is defined as the algebraic mean of TSR_{min} and TSR_{max}.

A plot of share versus time for a typical multi-rate sensor is given in Figure 2. In this simulation, a constant bit rate sensor suddenly increases its data rate from 100 kbps to 200 kbps and later decreases its rate to 10 kbps.

4.2 Fast Recovery Mechanism

It was observed that for sudden extreme increases in a slave's data rate, it may require a long time for the master to increase the slave's share to the appropriate value. The solid line in the bottom plot of Figure 3 shows a plot of share versus time for a slave sending constant bit rate traffic at 10 kbps and suddenly increasing its rate to 400 kbps. For a single observation window, the most that the master can increase the share, even if the RSR for that window is 100%, is $\frac{1}{TSR_{avg}}$. Also, if the slave is given a low share, the delay between the ends of consecutive observation windows will be large, so it may take a long time before the

	$RSR > TSR_{max}$	$RSR < TSR_{max}$ $RSR > TSR_{min}$	$RSR < TSR_{min}$
$LTSR > TSR_{max}$	Share is too low. $share = share \times min(alpha_{share}, \frac{min(LTSR,RSR)}{TSR_{avg}})$	LTSR may be settling. Take no action.	Abnormal situation. Take no action.
$LTSR < TSR_{max}$ $LTSR > TSR_{min}$	Possible jitter. Take no action.	Ideal. Take no action.	Possible jitter. Take no action.
$LTSR < TSR_{min}$	Abnormal situation. Take no action.	LTSR may be settling. Take no action.	Share is too high. $share = share \times max(\frac{1}{alpha_{share}}, \frac{max(LTSR,RSR)}{TSR_{avg}})$

Figure 1: Scenarios for RSR and LTSR.

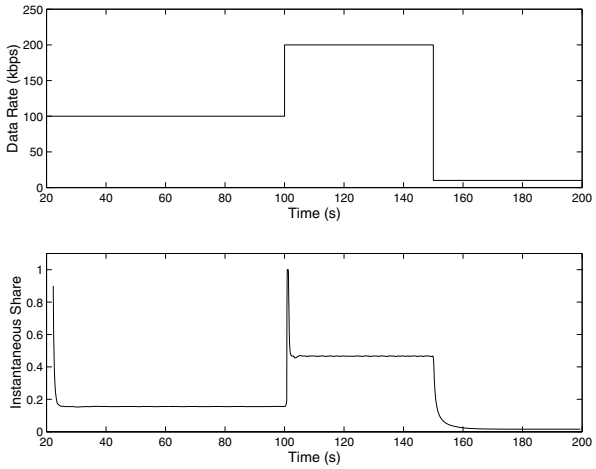

Figure 2: Typical plot of share vs. time for a multi-rate sensor.

master becomes aware of a queue backlog. To take care of this problem, ASP includes a "Fast Recovery" mechanism.

If a slave's RSR for an observation window reaches 100%, this may be an indication that there has been a sudden increase in the slave's offered load and severe action should be taken to avoid large queue buildups. However, it would not be appropriate to take severe action whenever a 100% RSR is observed since this could also be a sign that there is some jitter in the inter-packet arrival times or that the traffic has becomes burstier. Considering this, the fast recovery mechanism was designed so that if the number of consecutive 100% RSRs reaches a treshhold, the share is multiplied by a factor K_{fr} ($> \frac{1}{TSR_{avg}}$) rather than going through the typical adjustment. The threshhold for beginning fast recovery depends on the value of LTSR. Also, to minimize the delay between the ends of observation windows for queues with low share, the observation window size is decreased to N_{fr} ($< N_{normal}$) whenever a 100% RSR is observed. The effect of the Fast Recovery mechanism can be seen in Figure 3.

4.3 Conditional Hold

In ASP, the master maintains a timer for each slave in its piconet which represents the next time (a value of the master's native clock) at which the slave should be polled in order to maintain its share. When the master is given the opportunity to poll a slave, it consults its timer table and determines if one of the slaves's timer value has expired. If so, it chooses the slave with the most outdated timer and polls it either by sending a baseband Poll packet or by sending an LMP "Conditional Hold" message. The Conditional Hold message is similar to the LMP message in which a master forces a slave into Hold mode, except that it contains three possible intervals for which the slave can enter Hold mode. The interval chosen by the slave depends on the number of slots used in the slave's reply. The master initially assumes that the slave will reply with a single-slot packet and updates the slave's timer accordingly. When the slave replies, the master will update the value of the slave's timer if a multi-slot packet is used. In some instances, the slave's timer will have a value small enough so that the slave will never actually be able to sleep, in which case the master simply sends a baseband Poll packet.

4.4 Fairness

If a master decides that one of its slaves requires an increase in share greater than what is available, it first attempts to satisfy the node that requires the least share. The master allocates this share to the slave if it is the case that every slave receiving this share would result in the combined share remaining under capacity. If this is not the case, the master gives the remaining slaves an equal division of the remaining share. This process is repeated until all slaves have share allocated to them. This method for allocation of resources has previously been has been documented [13] and was used in Predictive Share Polling as well [15].

4.5 Extension to Bidirectional Traffic Patterns

In the design of ASP, only slave-to-master traffic was considered in the master's allocation of bandwidth to the

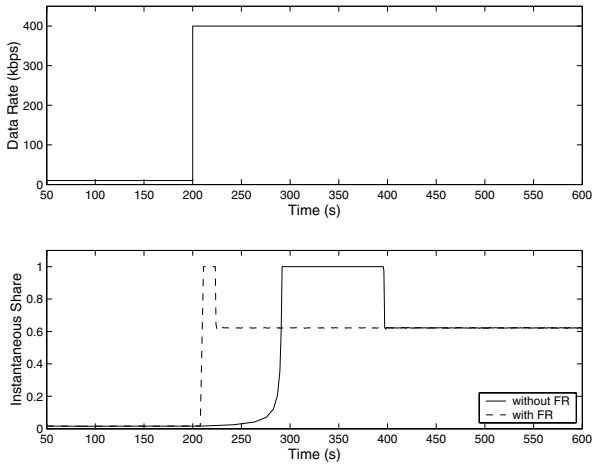

Figure 3: Plot of share vs. time for a sensor with a sharp rate increase.

slaves. However, it would not be difficult to modify the algorithm so that observed traffic patterns in both directions affect the master's polling decisions. Success ratios would simply need to be calculated based on the packets transmitted by the master as well as those transmitted by the slaves. The only difficulty with the modification is that it necessitates the sending of either a separate Conditional Hold packet after the master's data packet or a hybrid packet containing data and the hold interval information.

5 Predictions

In the following predictions of power consumption and average packet delay, we assume that the success ratio is maintained at a precise value rather than within a range throughout the course of the connection. In this section, target success range is given as a single value that represents both TSR_{min} and TSR_{max}, i.e., $TSR = TSR_{min} = TSR_{max}$. In these predictions, as well as in the simulations, we used the product specification of Ericsson's ERC41 Bluetooth Radio Core module to derive typical values for transmit, receive, idle, and low power oscillator mode power [8]. These values are:
P_t = 100 mW, P_r = 78 mW, P_i = 486 μW, P_{lpo} = 54 μW

In the single slave case, we can predict the average power dissipation in the radio core of the master (P_m) and slave (P_s) based on these powers and the fraction of time spent by the devices performing each of the possible tasks.

$$\begin{aligned} P_m = \quad & (share_{data} + share_{Null})P_r + share_{i,m}P_{idle} + \\ & (share_{CH} + share_{Poll})P_t + share_{lpo,m}\ P_{lpo} \end{aligned} \tag{2}$$

$$\begin{aligned} P_s = \quad & (share_{data} + share_{Null})P_t + share_{i,s}P_{idle} + \\ & (share_{CH} + share_{Poll})P_r + share_{lpo,s}\ P_{lpo} \end{aligned} \tag{3}$$

where $share_{data}$, $share_{Null}$, $share_{CH}$, and $share_{Poll}$ represent the share of time being spent on data packet transmission, Null packet transmission, Conditional Hold packet transmission, and Poll packet transmission, respectively. $share_{i,m/s}$ and $share_{lpo,m/s}$ represent the share of time spent by the master/slave in idle mode and low power oscillator, respectively. For a given bit rate R and application packet size of $PacketSize$ Bytes, data packets will arrive at the slave's queue at a rate of

$$AR_{data} = \frac{R}{8 \times PacketSize} \tag{4}$$

where AR_{data} has units packets/second. Given this data arrival rate, we can calculate the share that the master must allocate to this slave in order to satisfy a given target success range TSR as

$$share = \frac{AR_{data}(1 + NumSlots) \times 625\mu s}{TSR} \tag{5}$$

where $NumSlots$ represents the length of the slave's replies (1, 3, or 5) normalized to a 625 μs slot. The share of the bandwidth that is used for successful packet transmission can be calculated as

$$share_{success} = AR_{data}(1 + NumSlots) \times 625\mu s \tag{6}$$

The total share of the bandwidth allocated to a slave consists of the share of bandwidth used for successful polls, which arrive at a rate of AR_{data}, and the share of bandwidth used for unsuccessful polls, which arrive at a rate of AR_{Null}. From the fact that unsuccessful polls have a temporal length of 1.25 ms, we can calculate AR_{Null} as

$$\begin{aligned} AR_{Null} &= \frac{share - share_{success}}{1.25ms} \\ &= \frac{share}{1.25ms} - \frac{AR_{data}(1+NumSlots)}{2} \end{aligned} \tag{7}$$

Since each data and Null packet must be preceded by a master's poll, we know that

$$AR_{CH} + AR_{Poll} = AR_{data} + AR_{Null} \tag{8}$$

where AR_{CH} represents the Conditional Hold arrival rate and AR_{Poll} represents the baseband Poll arrival rate. In ASP, the master sends a baseband Poll packet when there is no chance that the slave will actually be able to sleep. We

can calculate the fraction of polls that consist of a baseband Poll packet as

$$\frac{AR_{Poll}}{AR_{Poll}+AR_{CH}} = \begin{cases} 4-\frac{3}{share} & share > \frac{3}{4} \\ 0 & else \end{cases} \quad (9)$$

We can now calculate AR_{Poll} from Equations 8 and 9 as

$$AR_{Poll} = \begin{cases} (4-\frac{3}{share})(AR_{data}+AR_{Null}) & share > \frac{3}{4} \\ 0 & else \end{cases} \quad (10)$$

AR_{CH} can be found easily from Equations 4, 7, 8, and 10. Knowing the arrival rates and lengths of each of the four types of packets used in ASP, we can now calculate the share of time spent on transmission for each. Since the master must keep a stable clock, no share of its time can be spent in LPO mode. The remainder of its time must be spent in idle mode.

$$share_{data} = AR_{data}(PSize + HSize + LHSize) \times 8\mu s \quad (11)$$

$$share_{Null} = AR_{Null}(HSize) \times 8\mu s \quad (12)$$

$$share_{CH} = AR_{CH}(CHSize) \times 8\mu s \quad (13)$$

$$share_{Poll} = AR_{Poll}(HSize) \times 8\mu s \quad (14)$$

$$share_{i,m} = 1 - share_{data} - share_{Null} - share_{CH} - share_{Poll} \quad (15)$$

We assume that packet size $PSize$ ranges from 0 - 335 Bytes, while Hold message size $HSize$, L2CAP header size $LHSize$, and Conditional Hold message size $CHSize$ are fixed at 20 Bytes, 4 Bytes, and 23 Bytes, respectively. $8\mu s$ represents the transmission time for a single Byte. The slave will be idle during a poll slot for the remainder of the slot period not occupied by the poll packet (baseband Poll or Conditional Hold). The remainder of its time may be spent in LPO mode.

$$share_{i,s} = AR_{CH} \times (625\mu s - CHSize \times 8\mu s) + AR_{Poll} \times (625\mu s - PollSize \times 8\mu s) \quad (16)$$

$$share_{lpo,s} = 1 - share_{data} - share_{Null} - share_{CH} - share_{Poll} - share_{i,s} \quad (17)$$

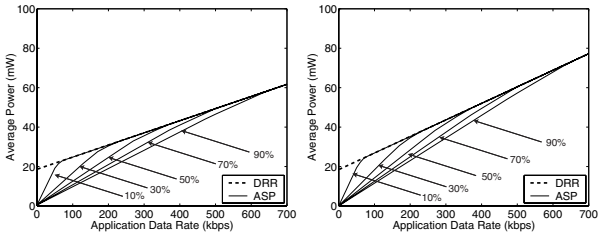

Figure 4: Predicted average power dissipation by the master (a) and the slave (b) in a single-slave piconet.

Using this analysis and assuming 335 Byte five-slot packets, we have plotted the expected power dissipation by the master and the slave as a function of application data rate R for ASP with multiple target success ranges as well as for DRR in Figure 4.

The average packet delay for ASP can also be predicted quite easily. The transmission time of a packet is

$$T_r = (NumSlots) \times 625\mu s \quad (18)$$

Packets are generated with arrival times that are uniformly distributed on a constant sized interval whose limits vary from 0 to $\frac{1.25}{share}$ ms before the successful poll in which the packet is transmitted. The means of these distributions average out to $\frac{625\mu s}{share}$ and the average packet delay can be easily calculated as

$$T_{delay} = \frac{625\mu s}{share} + T_r = (\frac{1}{share} + NumSlots) \times 625\mu s \quad (19)$$

Using this analysis and assuming 335 Byte five-slot packets, we have plotted the expected average delay as a function of application data rate in Figure 5.

6 Simulations and Analysis

We used Network Simulator [12] and IBM's BlueHoc extension [1] to simulate ASP and analyze power dissipation and packet delay in various Bluetooth piconet models. In all simulations, we assume high quality links (no lost packets), slave-to-master traffic only, and negligible overhead in initial link establishment and settlement to the proper share. In all simulations, constant bit rate traffic patterns with maximum size five-slot packets and inter-packet arrival time jitter are used.

The first network considered is a single-slave piconet. In these simulations, we calculated power dissipation by the master and the slave and average packet delay for various

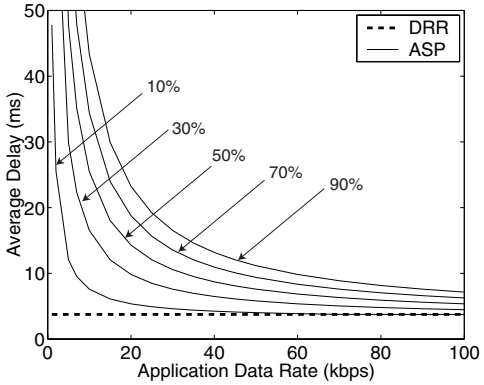

Figure 5: Predicted average packet delay in a single-slave piconet.

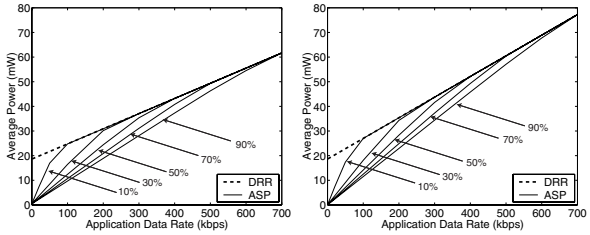

Figure 6: Average power dissipation by the master (a) and the slave (b) in a single-slave piconet.

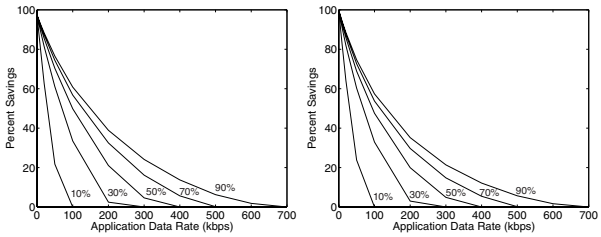

Figure 7: Percent savings of ASP in power at the master (a) and the slave (b) in a single-slave piconet.

application data rates. We ran these simulations for ASP with various target success ranges and for DRR. In addition, we ran simulations for two-slave piconets with both symmetric and asymmetric rates among the slaves. In the symmetric simulations, we varied each slave's application data rate, always keeping them equal. We calculated the power dissipation by the master and the average among the slaves and average packet delay. In the asymmetric simulations, we kept one slave's rate constant at 10kbps while varying the other slave's rate. For these simulations, we calculated the power dissipation at the master, the low data rate slave, and the variable data rate slave. We also calculated the average packet delay for the flow involving the low data rate slave and that involving the variable data rate slave. We ran all two-slave piconet simulations for ASP with target success range of $\{0.85, 0.85\}$ and for DRR. Finally, we ran simulations for seven-slave piconets, again using slaves with both symmetric and asymmetric rates. In the asymmetric simulations, we kept one slave's rate constant at 10kbps while varying the other slaves' rates. We ran all seven-slave piconet simulations for ASP with target success range of $\{0.85, 0.85\}$ and for DRR.

6.1 Power Dissipation

Figure 6 shows the average power dissipation of the master and slave in a single slave piconet for various target success ranges (10%, 30%, 50%, 70%, 90%) and for DRR. The curves for ASP with all target success ranges eventually converge to DRR when the target success ranges and offered load require that a share of 1.0 should be allocated to the slave. Our simulation results match our predictions in Figure 4 very closely. Figure 7 shows the percent savings in average power at the master and the slave that ASP can achieve over DRR. As expected, savings are very high at very low data rates (96.8% for 1 kbps) when using a high (90%) target success range. Even when using a low target success range (10%), the power savings at low rates are very high (95.6% for 1 kbps). Savings become insignificant at high data rates.

Figure 8 shows the average power dissipated by the master and each slave for a symmetric two-slave piconet for ASP and for DRR. As shown in Figure 8a, at the master, DRR scheduling is actually an improvement over ASP in terms of power dissipation for high slave data rates. In this case, the master using ASP has allocated 100% of the share to the slaves. Since the master is polling at every opportunity, its average receive power should be the same as if it were using DRR. The difference is that with ASP, the master may sometimes send Conditional Hold messages in place of Poll messages. Consequently, its average transmit power is greater than it would be if using DRR. The power inefficiency at the master is justified by the savings at the slaves, which can be seen in Figure 8b.

Figure 9 shows the average power dissipated by the master, the low (constant) data rate slave, and the variable data rate slave for an asymmetric two-slave piconet for ASP and for DRR. Using ASP, significant power savings over DRR can be seen for all devices at low data rates. As expected, the power savings at the master and the variable data rate slave become much less when the variable data rate slave

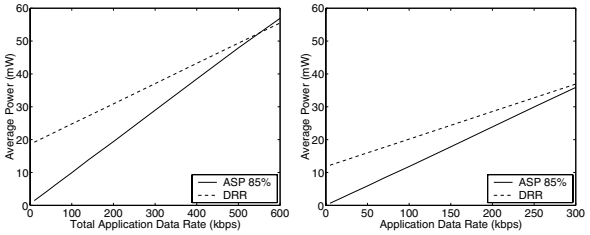

Figure 8: Average power dissipation by the master (a) and each slave (b) in a symmetric two-slave piconet.

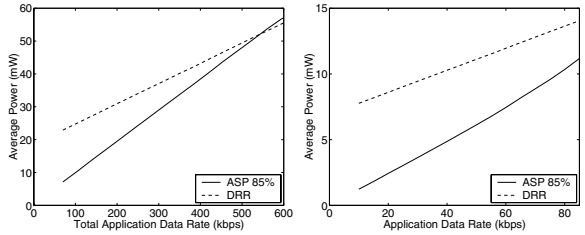

Figure 10: Average power dissipation by the master (a) and each slave (b) in a symmetric seven-slave piconet.

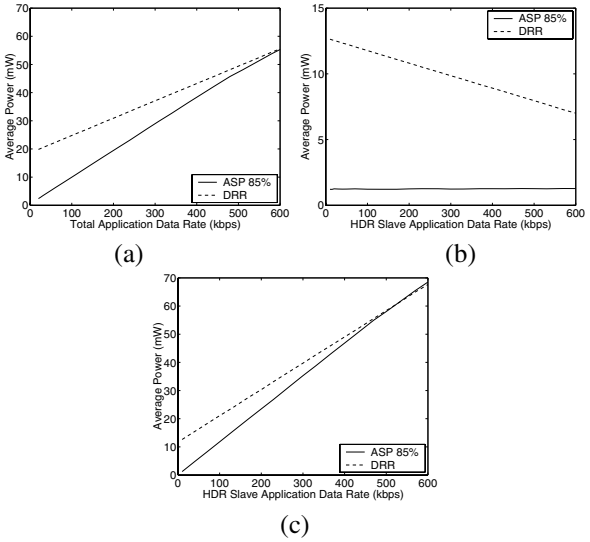

Figure 9: Average power dissipation by the master (a), the low data rate slave (b), and the variable data rate slave (c) in an asymmetric two-slave piconet.

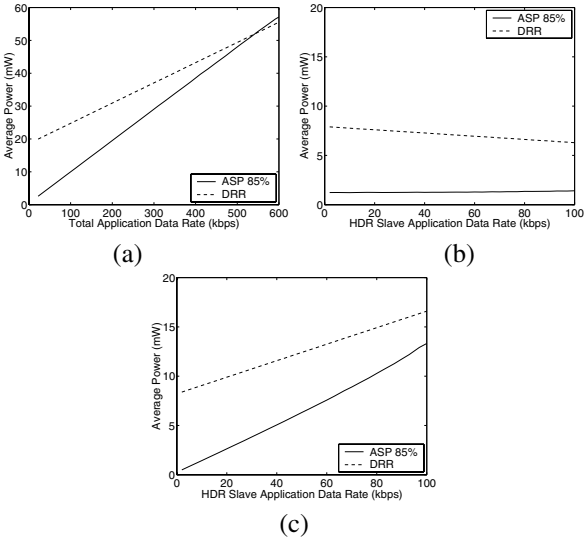

Figure 11: Average power dissipation by the master (a), the low data rate slave (b), and each of the variable data rate slaves (c) in an asymmetric seven-slave piconet.

increases its rate. The average power at the low data rate slave remains approximately constant for ASP as a function of the variable data rate slave's rate. For the DRR case, as the variable rate slave's data rate increases, the master polls the variable rate slave more and the low data rate slave less. Thus the power dissipated by the low data rate slave will decrease as the variable rate slave's rate increases for DRR.

Figure 10 shows the average power dissipated by the master and each of the slaves for a symmetric seven-slave piconet for ASP and for DRR. Similar to the two-slave piconet case, the master dissipates more power with ASP than with DRR for high data rates, and again this is justified by the savings at the slaves.

Figure 11 shows the average power dissipated by the master, the low data rate slave, and each of the variable data rate slaves for an asymmetric seven-slave piconet for ASP and for DRR. Similar to the asymmetric two-slave piconet case, significant power savings can be seen for the master and all slaves at low data rates and the savings become much less when the variable data rate slaves increase their rates. When data rates among the variable rate slaves become very high, although the low data rate slave's rate remains constant, its power consumption increases. This is because when the low data rate slave returns from sleeping, the timer of another slave will often expire simultaneously. When this happens, the master may poll one of the other slaves first and the low data rate slave will receive access codes and headers despite the fact that it is not being addressed by the master.

6.2 Average Packet Delay

In the previous section, we showed that ASP can achieve significant power savings over a persistent polling method

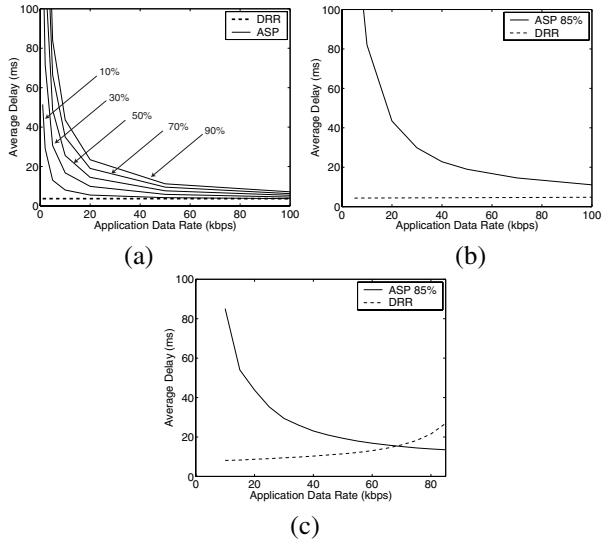

Figure 12: Average packet delay for the slave in a single-slave piconet (a), each slave in a symmetric two-slave piconet (b), and each slave in a symmetric seven-slave piconet (c).

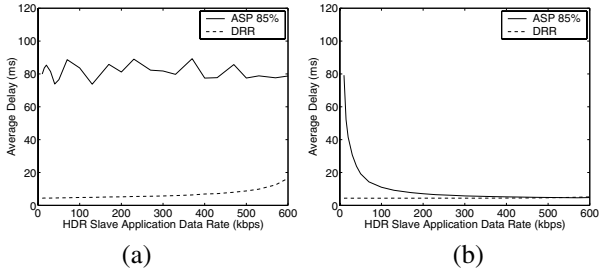

Figure 13: Average packet delay for the low data rate slave (a) and the variable data rate slave (b) in an asymmetric two-slave piconet.

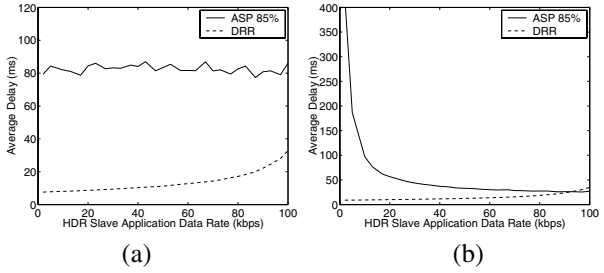

Figure 14: Average packet delay for the low data rate slave (a) and the variable data rate slave (b) in an asymmetric seven-slave piconet.

when slave data rates are low. The cost for these power savings comes in the form of increased packet delay. In this section, we present simulation results that quantify this increase.

Figure 12a shows the average packet delay for a single-slave piconet for ASP with various target success ranges and for DRR. As the target success range approaches 0%, ASP becomes a persistent poller and the curve approaches that of DRR. The curves for ASP match our predictions in Figure 5 very closely and converge to the constant average packet delay seen in DRR for high data rates. Figures 12b and 12c show the average packet delay for a symmetric two-slave piconet and a symmetric seven-slave piconet, respectively, for ASP and for DRR. It is interesting that ASP actually outperforms DRR in terms of average packet delay for high data rates. The reason is that a master implementing DRR polls its slaves in a burstier manner than in ASP. In fact, the average packet delay in a piconet using ASP will converge to that for a piconet using a simple round robin approach, where slaves are polled in a strictly cyclic manner, for very high data rates.

Figure 13 shows the average packet delay at the low data rate slave and the variable rate slave for an asymmetric two-slave piconet for ASP and for DRR. Figure 14 shows similar results for the average packet delay at the low data rate slave and each of the variable rate slaves for an asymmetric seven-slave piconet. In both the two-slave and seven-slave case, with ASP the low data rate slave's average packet delay remains fairly constant, as we would expect. Since this slave's rate does not change, its share also remains constant and so it is being polled at a rate independent of the other traffic on the piconet. As the rates of the variable data rate slaves become low, DRR acts like simple round robin, polling the slaves cyclically and usually utilizing one time slot pair for each slave since most slave replies consist of Null packets rather than data. As these rates increase, slaves will have data to send more frequently. Since five-slot packets are being used, this means that three time slot pairs will be occupied by a poll and there will be longer delay between polls. Therefore, even the low data rate slave, whose rate remains constant, will be subjected to higher average packet delay as piconet throughput increases.

6.3 Power-Latency Tradeoff

It is evident from our results that the choice of a target success range should be made to balance a tradeoff between average packet delay and power-efficiency, a tradeoff typical of polling networks. Figure 15 shows a plot of total power (master and slaved combined) versus average packet delay for a single-slave piconet. In the plot, a solid line connects points of the same application data rate. Movement along a single curve from high power consumption

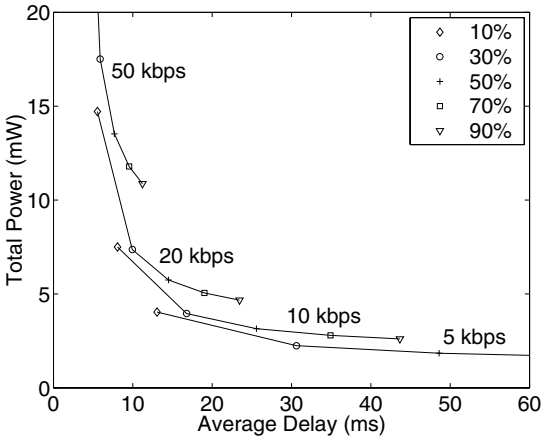

Figure 15: Power versus delay tradeoff in a single-slave piconet.

and low average packet delay toward low power consumption and high average packet delay can be interpreted as an increase in target success range. A system designer (or network middleware) forced to operate on a single curve determined by the slave's data rate is given the flexibility to choose where along that curve to operate so that power constraints and/or network performance goals are met.

7 Conclusions and Future Work

We have proposed a novel scheduling algorithm for Bluetooth to control intra-piconet traffic. This algorithm allows the system designer to balance the tradeoff between power-efficiency and packet latency based on the requirements of the application. Furthermore, this tradeoff can be made on a node by node basis, providing a great deal of flexibility. We are currently working to develop an extension of ASP in which the master empties the slave's queue once a successful poll is observed, a strategy similar to that of many of the polling schemes that have been proposed to date. We expect that this algorithm will sacrifice some short-term fairness but will dramatically improve average packet delay when sensors use large L2CAP packets that must be segmented into multiple baseband packets.

In the near future, our research will be directed toward the design of a middleware for use in sensor network applications such as those being developed for a Smart Medical Home at the University of Rochester's Center for Future Health [5]. The goal of this middleware is to allow the network to adapt itself to the requirements of an application so that a minimum level of reliability is maintained in a network whose topology is potentially very dynamic. We believe that this middleware will be able to use ASP by dynamically adjusting such design parameters as target success range to meet time-varying application requirements.

References

[1] BlueHoc: Bluetooth performance evaluation tool. http://oss.software.ibm.com/developerworks/ opensource/bluehoc/, 2002.

[2] Bluetooth. http://www.bluetooth.com/, 2002.

[3] R. Bruno, M. Conti, and E. Gregori. Wireless Access to Internet via Bluetooth: Performance of the EDC Scheduling Algorithm. In *Proceedings of the First Workshop on Wireless Mobile Internet*, 2001.

[4] A. Capone, M. Gerla, and R. Kapoor. Efficient Polling Schemes for Bluetooth Picocells. In *IEEE International Conference on Communications*, 2001.

[5] University of Rochester Center For Future Health. http://www.futurehealth.rochester.edu/, 2002.

[6] I. Chakraborty, A. Kashyasp, A. Kumar, A. Rastogi, H. Saran, and R. Shorey. MAC Scheduling Policies with Reduced Power Consumption and Bounded Packet Delays for Centrally Controlled TDD Wireless Networks. In *IEEE International Conference on Communications*, 2001.

[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 *IEEE Infocom*, 2001.

[8] Ericsson. http://www.ericsson.com/bluetooth, 2002.

[9] S. Garg, M. Kalia, and R. Shorey. MAC Scheduling Policies for Power Optimization in Bluetooth: A Master Driven TDD Wireless System. In *IEEE Vehicular Technology Conference*, 2001.

[10] N. Johansson, U. Kolmer, and P. Johansson. *Performance Evaluation of Scheduling Algorithms for Bluetooth*, pages 139–150. Kluwer Academic Publishers, 2000.

[11] D. Kalia, M.and Bansal and R. Shorey. Data Scheduling and SAR for Bluetooth MAC. In *IEEE Vehicular Technology Conference*, 2001.

[12] Network Simulator - ns. http://www.isi.edu/vint/nsnam/, 2000.

[13] K. Ramakrishnan, R. Jain, and D. Chiu. Congestion Avoidance in Computer Networks with a Connectionless Network Layer, Part IV: A Selective Feedback Scheme for General Topologies. In *Technical Report DEC-TR-510, Digital Equipment Corporation*, Aug. 1984.

[14] M. Shreedhar and G. Varghese. Efficient Fair Queueing using Deficit Round Robin. In *Proceedings of ACM SIGCOMM*, 1995.

[15] R. Yaiz and G. Heijenk. Polling Best Effort Traffic in Bluetooth. In *The Fourth International Symposium on Wireless Personal Multimedia Communications*, 2001.

[16] H. Zhu, G. Cao, G. Kesidis, and C. Das. An Adaptive Power-Conserving Service Discipline for Bluetooth. In *International Conference on Communications*, 2002.