Design and Implementation of QoS-driven Dynamic Slot Assignment and Piconet Partitioning Algorithms over Bluetooth WPANs

Carlos Cordeiro, Sachin Abhyankar, and Dharma P. Agrawal
OBR Research Center for Distributed and Mobile Computing
Department of ECECS, University of Cincinnati – Cincinnati, OH 45221-0030 (cordeicm, sabhyank, dpa)@ececs.uc.edu

Abstract — Bluetooth is a promising wireless technology aiming at supporting electronic devices to be instantly interconnected into short-range ad hoc networks. The Bluetooth medium access control protocol is based on the Master/Slave concept wherein any communication between slave devices has to go through the Master. While this model is simple, it incurs longer delay between any two slave devices due to the use of non-optimal packet forwarding scheme and the use of double the bandwidth at the Master. Moreover, if more than two devices want to communicate as a group, this can only be achieved by either multiple unicast transmissions or a piconet-wide broadcast from the master. To handle these issues efficiently, we propose a novel combination of Dynamic Slot Assignment (DSA) and piconet partitioning. With DSA, the piconet Master dynamically assigns slots to Slaves so as to allow them to communicate directly with each other without any intervention from the Master. Such proposed communication architecture provides for Quality of Service (QoS) requests, admission control, and multi-device conversation by which a multicast-like communication is feasible within a piconet. To widen the scope of DSA, we propose QoS-driven Enhanced DSA (EDSA) where dynamic piconet partitioning and scatternet support comes into picture. Devices are grouped in piconets according to their connection endpoints in EDSA, enabling it to be employed over a scatternet. We have performed extensive simulations and observe that these schemes drastically enhance Bluetooth performance in terms of delay and throughput, while significantly reducing network power consumption.

I. INTRODUCTION

Bluetooth [1, 22, 24] is a wireless communication technology that provides short-range, semi-autonomous radio network connections in the 2.4GHz Industrial-Scientific-Medical (ISM) band, and can establish a class of ad hoc networks, called piconets. It was also chosen to serve as the baseline of the IEEE 802.15.1 standard for Wireless Personal Area Networks (WPANs) [2], which can support both baseline of the IEEE 802.15.1 standard for Wireless Personal networks, called piconets. Bluetooth operates in a Master/Slave concept wherein the Master periodically polls the Slave devices and only after receiving such a poll is a Slave allowed to transmit. Multiple piconets can be linked together through some of the devices acting as bridges to form what is called a scatternet.

So far, we can identify three waves of Bluetooth-based applications. In the first wave, Bluetooth was intended as a cable replacement technology so as to enable a wide range of devices such as laptops, PDAs, mobile phones, and headsets, to form ad hoc networks in a semi-autonomous fashion [1]. With the advent of access points (with functionality similar to IEEE 802.11 access points) enabling hundreds of Bluetooth units to access the wired network in places such as theaters, stadiums, conferences, pavilions, and so on [28, 29]. However, in the third wave of applications, the low cost, low power, and effortless and instant connection provided by Bluetooth technology has also become attractive for automatically forming an ad hoc network of a large number of low-power sensor nodes [30, 31]. These sensor network applications are characterized by thousands of nodes embedded in the physical world, and a Bluetooth RF link to enable them to form a network by only bridging these sensor nodes within their radio range [22, 32, 33]. This calls for solutions to be developed that are equally applicable to both small scale and large scale Bluetooth networks, and that aims at keeping interference at the minimum level [6, 21, 23]. As it is shown later, our proposed schemes satisfy such scalability requirements as opposed to existing solutions.

By virtue of the Master/Slave communication model, the Bluetooth medium access provides for simplicity, low power (as compared to other standards), and low-cost, and these have become major forces driving the usefulness of the technology. However, this design choice also introduces major shortcomings, such as inability for the slaves to communicate directly with each other since their packets must be forwarded through the master device [1, 22, 24]. Moreover, in Bluetooth there is no built-in support for group communication and its many applications [17, 27], which can only be achieved by either multiple unicast packets or by a piconet-wide broadcast from the master. As a result of these limitations, the packet forwarding among slaves in Bluetooth becomes sub-optimal, bandwidth is wasted by forwarding through the master, end-to-end packet delay increases, and power consumption is significantly increased at the master unit due to its frequent medium access for transmission and reception. Therefore, the adoption of the Master/Slave paradigm in its present form does not seem to be the most appropriate solution.

To overcome these limitations and address the shortcomings of the current Bluetooth Master/Slave communication model, we initially propose a novel Dynamic Slot Assignment (DSA) scheme to be coordinated by the master of a piconet. Based on the piconet traffic patterns, the master device dynamically allocates slots for direct communication between slaves. This
way, packets do not need to be forwarded by the master while it periodically re-evaluates slot assignments and changes it accordingly. This novel communication architecture enables for not only direct slave-to-slave communication but also serves as multi-slave communication, hence emulating a group (i.e., multicast-like) communication within the piconet. In order to enable DSA to be employed over a scatternet as well as to support a higher number of simultaneous connections with their required QoS, we propose the use of a QoS-enabled dynamic piconet partitioning scheme into DSA in what we refer to as Enhanced DSA (EDSA). We note that one of the major features encouraging the use of these schemes is the implementation of QoS mechanisms such as resource request and release, and admission control. We have carried out extensive simulations of DSA and EDSA and observe a significant enhancement in current Bluetooth performance. The design of EDSA is such that it is inherently scalable, hence could efficiently support all the three waves of Bluetooth applications.

The rest of this paper is organized as follows. Section II gives an overview and motivation for our work, while Section III describes our proposed DSA and EDSA schemes. Next, Sections IV, V, and VI present the simulation methodology and results of extensive runs, along with comparisons with other existing solutions. Related work is then given in Section VII. Finally, the paper is concluded in Section VIII.

II. BLUETOOTH OVERVIEW AND MOTIVATION

The details of the Bluetooth system, architecture and protocols are defined in [1]. Bluetooth is a short-range (up to 10m) wireless link technology. It operates in the 2.4 GHz ISM frequency band, where the raw data rate is 1 Mbit/s. A Time Division Duplex (TDD) technique divides the channel into 625µs slots, where a slot can carry up to 625 bits. Transmission occurs in packets, where DHx (without FEC) and DMx (with FEC) packets are usually employed for data traffic, while HVx packets are used for voice applications. Here, x represents the number of slots and can be equal to 1, 3 or 5.

![Figure 1. Packet transmission in Bluetooth](image)

The Bluetooth specification [1] defines two different types of links for data and voice applications. The first link type Synchronous Connection Oriented (SCO) is treated as circuit-switched, point-to-point traffic, whereas the second link type Asynchronous Connectionless Link (ACL) acts as a packet-switched, point-to-point data traffic. Usually, SCO links are used for audio applications with strict quality of service requirements where packets are transmitted at predefined regular intervals, while ACL links are often used in data applications where there is no such strict requirement on end-to-end delay. At most three SCO connections can be supported within a piconet and the polling cycle varies when connections of such a type are present, ranging from 6 for HV3 packets to 3 for HV1 packets. Stating that SCO packets from a given connection have a polling cycle of 6 slots means that one SCO packet has to be sent in every 6 slots, so as to achieve the 64 Kbps bandwidth needed for voice applications. On the other hand, ACL connections do not have such requirements and the polling cycle can be expanded or shrunk according to the number of slaves in a piconet and their traffic. For simplicity, we assume that every SCO connection uses HV3 packets which have a polling cycle of 6 slots, even though our scheme applies to any SCO packet type. Figure 1 illustrates a full polling cycle highlighting the use of 1, 3 and 5-slot packets.

Figure 2 depicts a Bluetooth piconet comprised of five devices. Figure 2(a) illustrates the case where the slave device S1 communicates with another slave S3, and all packets have to be forwarded through the master device M. Here, the packet forwarding between S1 and S3 is clearly sub-optimal, bandwidth is wasted by forwarding through the master M, end-to-end packet delay increases, and, as shown later, power consumption is significantly increased at the master unit due to its frequent medium access for transmission and reception. Therefore, we believe that for existing devices with limited battery power and for application efficiency, the Master/Slave paradigm of Bluetooth needs to be changed from its current form.

![Figure 2. Master/Slave communication in Bluetooth](image)

In case slave S1 wants to send packets (e.g., a business card) to both S3 and S4, it has to send two unicast packets as illustrated in Figure 2(b). On top of that, these multi-slave packets may have to be kept in a device’s buffer for as many polling cycles as is the number of recipients, since a device can only address one other unit in a given polling cycle. Note that a piconet-wide broadcast may be a feasible option, however as the master is the only device capable of sending a broadcast, packets will still have to be forwarded through it. Additionally, the trade-off between when to employ multiple unicasts or a broadcast is not clear since slaves within a piconet are stateless. Hence, given the importance and applicability of multicasting, a basic support for group communication ought to be provided.

Finally, one major limitation arises when we consider the limitations of current Bluetooth, namely, the support of a very limited number of audio connections (e.g., at most one duplex audio connection), no delay or throughput guarantees to data connections, and the lack of end-to-end QoS guarantees [1]. Given that, we believe that simple QoS primitives ought to be devised in order to support basic application QoS demands.

III. THE QOS-DRIVEN DYNAMIC SLOT ASSIGNMENT (DSA) AND PICONET PARTITIONING ALGORITHMS

We propose a novel QoS-driven Enhanced Dynamic Slot Assignment (EDSA) scheme so that we can address these major shortcomings in the Bluetooth design while keeping the
simplicity of the Master/Slave paradigm. Our basic strategy is to utilize both and combine the QoS-driven Dynamic Slot Assignment (DSA) and the dynamic piconet partitioning. Here, DSA-only is employed at the piconet level while dynamic partitioning opens up the scope of DSA to the scatternet level. The basic idea behind DSA is to appropriately manage the polling cycle conducted by the master of the piconet given the connection QoS requirements. As devices initiate or terminate communication with each other within the piconet, we restructure the piconet polling cycle (expanding it with a new connection or shrinking it upon termination), build the transmission schedule for each device, and then propagate the resulting schedule to the members of the piconet. This way, slaves know exactly in which slot to transmit and/or listen. Therefore, not only is direct communication between slave devices supported, but also a multicast-like communication by having several destination slaves listen to the same slot is achieved. As detailed later on, we temporarily use the broadcast address in order to implement multicasting. If we assume a uniform distribution of connection requests within a piconet, we can conclude that slave-to-slave communication will be present in approximately 75% of all connections, thereby stressing the need for supporting and optimizing such cases.

In order to widen the scope of DSA to scatternets, to effectively support application QoS demands, and to provide effective scalability, a controlled form of dynamic partitioning has been developed to interoperate with DSA. This new scheme is referred to as Enhanced DSA (EDSA). EDSA dynamically partitions piconets when application QoS demands cannot be satisfied by the current slot allocation. The partitioning is guided by the connection endpoints as we try to keep communicating devices within the same piconet. If this cannot be achieved, EDSA carefully synchronizes slot allocations of neighboring piconets so that data can be transferred from one piconet to another over the scatternet, hence providing uninterrupted communication. As we will show, the application of EDSA provides increased application and system performance, effective QoS guarantees, and scatternet support.

Next, we describe the DSA scheme while the dynamic piconet partitioning algorithm within EDSA is described later.

A. Connection Request, Release and Admission Control

In order for the piconet master to optimally assign and reserve slots for piconet devices, it needs to know the QoS requirements for each participating slave device. In DSA, whenever a slave device wishes to establish a connection with another device (whether another slave or the master itself), it sends a CONNECTION_REQ message to the piconet master, specifying in its payload: i) the destination (if unicast) or destinations (if multicast) address to which the device wants to establish a connection; ii) the baseband packet type to be used in its transmission; iii) desired transmission rate; and iv) acceptable transmission rate. For instance, when slave S1 in Figure 2(a) wants to establish an FTP connection with slave S2, it could send a connection request to its master M, as CONNECTION_REQ(S3, DH5, 30, 50), while it would send a CONNECTION_REQ(S3, S4, DH5, 30, 50) for a multicast FTP connection to slaves S3 and S4. The transmission rate defines the frequency (in number of slots) a device desires to transmit. In our last example, slave S1 notifies the master that it desires to transmit one DH5 packet every 30 slots, but it accepts to transmit in every 50 slots if the desired transmission rate cannot be supported.

When a connection request arrives at the master, it takes the requesting slave address (contained in the packet header), the packet type – which identifies the type of connection (voice or data) requested –, the destination address contained in the payload, and the QoS requirement (e.g., transmission rates). If the master can grant this request, it allocates a unique identifier to the connection, recalculates the appropriate schedule (detailed in the next subsection), and broadcasts the scheduling information to all active slaves. If the connection cannot be supported with this QoS level (there exists no intra-piconet slot scheduling to accommodate the request), the master enters the partitioning algorithm (see section III.F) where it considers dividing the piconet in such a way that the new connection request can be accepted. If possible, the piconet partitioning algorithm will group devices in such a way that all existing data connections remain with their endpoints in the same piconet in order to avoid scatternet-wide connections. In the event that scatternet-wide connections cannot be avoided, the piconet partitioning scheme will interact with DSA in order to design a joint and synchronized inter-piconet slot scheduling between the partitioned piconets, so as to enable scatternet-wide connections not only to remain active but to perform at their best. If and only if no such partitioning exists (e.g., a conflict in finding an appropriate device grouping given the existing connections amongst them), the master will return a CONNECTION_REJ message back to the source. Back to our earlier example, having requested a DH5 packet does not guarantee that the slave will be granted its request. As explained later, based on the current traffic and schedule, the master decides which packet is best and propagates this information to all slaves with a broadcast. Also, we shall see that the traffic type influences the slot assignment, as SCO packets have to be scheduled periodically due to their time constraints, while no such restriction exists for ACL.

An important issue arises in case of bi-directional flows (e.g., TCP traffic), as slots have to be reserved for the reverse traffic as well. In this case, it is up to the destination device to detect the presence of reverse traffic and similarly make a slot reservation through a CONNECTION_REQ message sent back to the source. However, as this message is sent in response to a bi-directional flow, it is handled differently by the master who immediately allocates the required slots specified within the request (e.g., DH3 for TCP ACKs) to be used in the reverse direction. If the master cannot satisfy this reverse connection request due to the absence of enough resources, it returns a CONNECTION_REJ message back to the source and also drops the associated forward connection, along with its reserved slots, which generated this bi-directional flow.

Similar to connection requests, slaves also send connection termination messages to the master device. A slave transmits a CONNECTION_REL message, specifying the connection identifier as originally allocated by the master. Returning to the previous example of Figure 2(a) where we assume C1,3 as being
the connection identifier, upon termination of the communication between $S_i$ and $S_j$, the slave $S_i$ would simply send a CONNECTION_REL($C_{i,j}$) to the master device M. The master M would free the slot allocated previously to this connection, calculate the new schedule for the remaining slaves and redistribute the new schedule within the piconet. Also, the master associates an idle timer ($T_{IDLE}$) with each connection to handle the case when a device moves out of range or the connection is idle for too long. Upon expiration of $T_{IDLE}$, the connection is terminated and its resources freed up. Note that the master keeps track of all the connections within the piconet in order to assign slots to the devices, however it is not generally believed [21] that piconets, which are capable of having at most eight devices [1], will have a large number of connections. Therefore, in our design, we assume that at most $\text{CONN} \_{\text{THRES}} = 16$ connections can be simultaneously present in a piconet, which turns out to be a very reasonable number.

B. Slot Scheduling and Assignment

Every time the master of a piconet receives a connection request, release, or the idle timer of a connection expires, it computes a new schedule for its associated devices. This schedule contains information about which slot(s) belong to which device, and in which slot(s) a given slave needs to listen. With this mechanism, devices are able to directly talk to each other in either a unicast or multicast-like communication.

The schedule has to be transmitted to all the slave devices so that each one of them can determine when to transmit and when to listen. For that, we have defined a new broadcast message called SLOT_SCHED, which has a format depicted in Figure 3 and is described in detail in section III.C. For simplicity, we assume that broadcast messages are reliable. Several approaches could be used to achieve reliability such as broadcasting the same message more than once. In our implementation, we assume reliability can be achieved by retransmission of the same broadcast message three times.

C. Slot Scheduling Message Format

Figure 3 illustrates a proposed format of the scheduling message employed in our schemes, with the size of each field in bits as indicated. With this single message, we can provide both unicast and multicast-like communication within a piconet. As we can see from Figure 3, the scheduling message consists of three parts: the header, the information about the new connection request which caused the scheduling of this new message, and information about the slot assignment in the new and existing ACL connections. As we shall see, SCO connections are treated differently due to their periodicity.

The header part of the message is composed of three fields. The first field indicates whether the connection request, that caused the transmission of this new scheduling broadcast, was ACL or SCO connection. Alternatively, this field determines whether the information about the new connection contained in the second part of the message is related to an SCO or ACL connection. This makes a difference in the slot scheduling as ACL connections can be scheduled at any time, while SCO connections have to appear at every polling cycle (see section II). The second field of the header indicates the number of destinations addressed by this new connection request. In case of a slave-to-slave communication, this field would be equal to one, whereas it would be equal to the group size in a multi-device communication. If this broadcast message is the result of a connection termination, this field contains zero. The third and final field in the header indicates the number of ACL connections contained in the third part of the scheduling message (to be soon described). In other words, this field is used to determine the boundaries of the message.

The second part of the scheduling message provides information about the connection request that has caused the generation of this scheduling broadcast, and absence of this part implies a connection termination. As mentioned earlier, both unicast and multicast-like communications are supported by this scheduling scheme. With this in mind, the first field indicates the source of data for the connection (i.e., the device which generated the connection request), followed by as many destination devices as specified in the number of destinations field contained in the header. Since each Bluetooth active member address is 3 bits [1], the size of this field is a multiple of 3. Next comes the field which indicates how many slots have been allocated by the master for transmission by the source. Recall from section III.A that together with the connection request message, the requesting device also specifies the type of packet it wishes to use in its transmissions. Depending upon the traffic pattern and the presence or absence of SCO connections, the master may or may not meet the device’s request. Finally, the last field of this message part is devoted to assigning a unique connection identifier to each connection within the piconet. The allocation of this identifier is managed by the master and communicated to the slaves in this broadcast message. This is the number that is used by the source slave when sending a connection release message to the master, and is also employed in the third part of this message.

The third and last part of the scheduling message contains the information about the schedule itself, that is, when each device is supposed to transmit and/or receive. The first field(s) contains lists of all the connection identifiers currently present in the piconet, and the order in which a connection identifier appears in this field determines the order in which devices associated with this identifier (either as source or destinations) have their slots assigned. To indicate the starting slot for transmission to the slaves (in case of the source of a connection) or reception (in case of the destination(s)), we use the field called $TxSlot$ (transmit slot). The values permitted for $TxSlot$ are 1, 3, or 5, depending upon the type of packet a connection uses. Lastly, as we have already mentioned, we
employ a scheme in DSA where we expand or shrink the slot-cyle according to the number of ACL connections in the piconet. Therefore, conveying the slot cycle information to all the slaves is the objective of the last field, called Slot-Cycle, as shown in Figure 3. This field is always a multiple of 6, as this value is the required periodicity of SCO connections. Based on that, a device determines the slot – slots = \((polling\_cycle \times index\_in\_list) + TxSlot\) – in which it is supposed to transmit and/or receive. Here, polling_cycle depends upon the SCO packet in use and is equal to 6 in case of HV3 packets, index_in_list is the index where the device’s associated connection identifier appears in the first field of the third message part, and TxSlot is as described earlier.

D. Example Scenario

To better illustrate the dynamics of DSA, consider again Figure 2(a) where we have a piconet composed of 5 devices, one master M and four slaves labeled S_i for 0 < i < 5. For simplicity purposes, we do not include QoS admission control in our discussion here, while this feature is incorporated into our DSA implementation. Assume that no traffic currently exists in the piconet, at which time slave S_1 sends a connection request to the master M in order to establish an ACL connection with slave S_2, specifying DH5 as the desired packet for its transmissions. Upon receipt of the connection request, the master M allocates a connection identifier, say C_1,2, to the connection between S_1 and S_2, and calculates the scheduling message to be broadcast as follows.

Initially, when there is no traffic, the slot cycle in the piconet is equal to 6 as polling is the only activity and is carried out by the master M. Figure 4(a) illustrates this scenario, where slots are numbered from 1 to 6. Upon receipt of the aforementioned ACL connection request, the master M increases slot cycle by 6 – making it 12 – and assigns the first slot cycle for direct communication between the requesting device S_1 and destination S_2, while it always keeps the last slot cycle for polling (see Figure 4(b)). Therefore, we see that the slot cycle is directly proportional to the number of ACL connections, being always a multiple of 6. The master also has to indicate the source and destination slaves (S_1 and S_2 in our example) the index within the assigned slot cycle where source (S_1) is supposed to transmit and the destination (S_2) is supposed to listen. This is accomplished by the field TxSlot where, in our example, the master M would set TxSlot to 1 as to allow slave S_1 to use the entire slot cycle for the transmission of its DH5 packets. The scheduling message is then broadcast to all slaves. If, for instance, another ACL connection request to employ DH3 packets from slave S_1 to S_2 arrives at the master M, it proceeds in a similar manner by assigning a connection identifier, say C_3,4, increasing the slot cycle from 12 to 18 to accommodate the new connection, building the new schedule, and finally broadcasting it throughout the piconet. This new schedule is illustrated in Figure 4(c).

Assume now that a simplex SCO connection request from device S_1 to S_2 arrives at the master M. Given the periodicity of SCO traffic which has to be present in each and every slot cycle, the master M has to reorganize the slot assignments. In our example, the connection C_1,2 will no longer be allowed to use DH5 packets as this would prevent SCO packets from being sent in every slot cycle, while the connection C_3,4 could continue employing the same packet type. To incorporate the SCO traffic within the schedule, the master changes the value of TxSlot from 1 to 3, indicating to all ACL connections that they will, from now on, transmit starting at slot 3, while the SCO connection is assigned the first two slots of every slot cycle. As a matter of fact, in DSA, SCO connections always use the first slots of a cycle while ACL connections follow next. Furthermore note that, contrary to ACL connections which have their polling cycle expanded according to the number of existing data connections, the slot cycle of SCO connections is always 6, implying that SCO packets have to be transmitted in every slot cycle. This is the reason behind not having the SCO slot cycle value specified in the scheduling message. This way, we adhere to the requirements of Bluetooth [1]. This new schedule is illustrated in Figure 4(d).

Note that in our scheme, we give a higher priority to SCO and connections that initiate early. This could be further enhanced to enable existing connections to be dropped or even have their resource allocations decreased in order to make room for new, higher priority connections.

E. Comments and Observations

One very important issue remains to be discussed, namely, how the multicast addressing is performed in DSA given that every Bluetooth packet contains a single three bit active member address (AM_ADDR), where seven of these addresses are used by the slaves and the remaining one is employed for broadcast. In the design and implementation of DSA, we considered two options to tackle this problem: we could increase the AM_ADDR field in the Bluetooth packet from three to four bits and use one out of these four bit addresses for multicasting, or we could temporarily allocate one of the three bit addresses for this purpose. The first option would make DSA incompatible with current Bluetooth. Therefore, we have selected the second option, where we temporarily allocate the broadcast address for the purpose of multicasting. However,
note that by allocating the broadcast address for multicasting does not imply that all devices will receive such a packet since the packet is discarded once the device realizes it is not supposed to listen at that particular slot as specified in the current slot scheduling message. Hence, only the actual multicast members (specified in the slot scheduling message) will accept the corresponding packet. On the other hand if the packet need to be broadcast throughout the piconet, then the given slot carrying the broadcast packet will not be present in the slot scheduling message, resulting in this broadcast packet being accepted by all piconet devices.

In the previous section, we have only provided a scenario where unicast communication takes place, however note that multicast-like propagation does not require any more changes than to specify the entire list of destinations in the second part of the scheduling message format. In this case, all devices belonging to the list of destinations would record the slot cycle and TxsSlot, resulting in all of them tuning to the same slots and hence receiving the same message. Additionally, our scheme also supports more than one multicast source. It is important to note that we have chosen a sender-initiated approach to support multicasting over Bluetooth, as it is optimal for small groups [18] where the sender has information about all the destinations. It contrasts with receiver-oriented approaches [17, 27] where receivers initiate group membership, and are best applicable for medium-to-large groups.

F. Dynamic Piconet Partitioning and the Enhanced DSA (EDSA) Scheme

We now describe our QoS-driven dynamic piconet partitioning algorithm which, together with DSA, results in the EDSA scheme. Dynamic piconet partitioning fulfills an important task of supporting a higher number of connections with their desired QoS. Here, we do not employ partitioning for the sole purpose of augmenting network capacity (throughput) [14, 28], while we see it as an important technique to satisfy application QoS needs if carefully and correctly used.

In EDSA, partitioning is triggered by a new connection request whose QoS requirements could not be accommodated by DSA, as it could not find an appropriate intra-piconet scheduling to satisfy the existing connections plus the new connection request demands. Here, the master tries to partition the current piconet such that the connection endpoints remain within the same piconet, that is, it tries to group the devices according to the connections amongst them. This way, we seek to minimize inter-piconet communication. In situations where inter-piconet communication cannot be avoided due to the connection topology among the devices, EDSA builds a joint and synchronized inter-piconet slot scheduling such that inter-piconet scheduling can be optimized. This scheduling is defined similar to the mechanisms already described earlier and broadcast to all devices right before partitioning itself. This way, any node acting as bridge between the two piconets would have its inter-piconet scheduling beforehand, hence optimizing and reducing the latency of inter-piconet traffic. Here, we do not tackle the issue of new scatternet-wide connections being established after partitioning, while we believe that existing proposals for scatternet scheduling could be employed [7, 34].

To clarify the QoS-driven partitioning scheme let us assume that, in the example of Figure 4, a request for a duplex SCO connection between S1 and S4 arrives at the master M. We believe this to be a realistic scenario where two devices initiate conversation by some data exchange through an ACL connection, and subsequently establish an audio connection through a duplex SCO traffic. Upon receiving the connection request, the master M of Figure 5(a) would try to figure out a new intra-piconet slot assignment to satisfy all application QoS demands. Since no intra-piconet schedule would be possible given the bandwidth requirements of the duplex SCO traffic, the master M would initiate the partitioning algorithm to find an appropriate grouping of devices such that partitioning could be optimal and inter-piconet communication minimized. After analyzing all connections, their QoS requirements, and their endpoints, a possible outcome of EDSA is illustrated in Figure 5(b) where the piconet master M has divided the original piconet into two: piconet 1 and piconet 2. Piconet 1 comprises slaves S1, S2, and the original master M (from now on referred to as M1), while piconet 2 is formed by slaves S3 and S4, where S4 has been assigned the role of master of its piconet by the original master M. As of now, the master of the new formed piconet is selected to be the device which is the endpoint of the majority of connections, hence reducing delay, overhead and power consumption, and saving piconet bandwidth. Returning to our example, in this new configuration piconet 1 would carry their initial ACL and simplex SCO connections between S1 and S2, while piconet 2 would possess the original ACL connection between S2 and M2 plus the new admitted duplex SCO connection between the same devices. By doing this, we ensure that the new connection request is not rejected and that the QoS requirements of existing connections are still satisfied.

Moreover, note from Figure 5(b) that M1 also acts as a bridge node between the neighboring piconets even if there is no current inter-piconet traffic between them (note that this bridge node does not necessarily need to be active in the absence of inter-piconet traffic). We do this for future inter-piconet connections where bridge nodes would be already in place, as well as to extend this concept for dynamic rejoining of piconets. To illustrate this, consider the case where the duplex SCO connection between devices S3 and M2 is terminated. In this case, instead of having two piconets and an idle bridge node between them, it could be better to rejoin piconets 1 and 2 into a single piconet as in its original configuration. By having the bridge node in place, we can extend our proposal to include rejoining through an appropriate exchange of messages between
masters $M_1$ and $M_2$. While this has not been included in our present implementation, its evaluation is underway.

While in our example of Figure 5 there have been no inter-piconet connections, it might so happen that the partitioning algorithm cannot find a grouping of devices such that this type of connections can be avoided. In these cases, we give higher priority to the SCO connections by having its endpoints confined within the same piconet, while ACL connections are allowed to span more than one piconet. This design decision is due to the fact that SCO traffic has strict QoS requirements (e.g., throughput), while no such strictness exist in the case of ACL traffic. Next, EDSA helps to find an appropriate scheduling for the inter-piconet connections. The algorithm basically works the same way as previously described, with the difference that EDSA now treats the bridge node as a pseudo-master responsible for generating and conveying the inter-piconet scheduling to the actual piconet masters. Finally, as briefly mentioned earlier in section III.A, the only situation where a CONNECTION_REJ message is returned back to the source is if and only if no device partitioning exists to accommodate a new connection request.

IV. SIMULATION ENVIRONMENT AND METHODOLOGY

To evaluate DSA-only and EDSA schemes, we have implemented them using the Network Simulator (ns-2) [19] and BlueHoc [20]. Since BlueHoc only provides the basic functionality of Bluetooth, we have made considerable extensions to this simulator in order to conduct our experiments including the support for direct slave-to-slave and multicast-like communications, dynamic partitioning, and the aforementioned QoS mechanisms.

The simulation experiments share the same network topology. It uses a single piconet comprised of 8 devices placed randomly within a 17m x 17m region. Radio propagation range is 10m, and nominal channel capacity is set to 1Mbps. We have thoroughly compared our schemes with existing Bluetooth under the following metrics:

- **Average Aggregate Throughput** – This is the number of data bytes correctly received by all piconet devices per unit time;
- **Average Delay** – Measures the delay per bit in transferring data from source to destination;
- **Overhead** – Measures the efficiency of the schemes. It relates the total number of data and control bytes received to the total number of data bytes sent;
- **Average Power Consumption** – With this metric, expressed in Megabytes/Joule (Mbytes/J), we evaluate the power efficiency of Bluetooth, DSA and EDSA. Bluetooth radios operate at 2.7 V and 30 mA, resulting in a 115 nJ/bit for transmission [1, 25]. Moreover, measurements have shown [26] that receive:send ratio is usually 1:1.4, while power consumption in idle times is negligible. In our simulations, we have used these relations to compute power consumption for the Bluetooth devices.

We have considered both slave-to-slave and multi-slave communications by varying the number of connections and the type of traffic. In order to cover a broad range of scenarios, we have run our simulations with FTP, Telnet, and SCO (voice) traffic, where FTP and Telnet connections use DHx and DMx packets while SCO traffic employs HV3 packets. Each traffic connection uses pair-wise distinct nodes, and we consider up to three connections of each type. To achieve reliability in the broadcast of the DSA scheduling message and at the same time determine an upper bound on the overhead of DSA, for each schedule change we perform the broadcast of a slot-scheduling message three times in all simulation runs. We believe it to be a reasonable number of broadcasts in comparing DSA with existing Bluetooth, or serving as a worst-case analysis.

V. DSA-ONLY SIMULATION RESULTS

In this section we investigate the performance of the DSA scheme only, while the evaluation of EDSA is given in the next section. In other words, here we assume that all connections have their QoS requirements satisfied by the piconet master. Also, results for unicast communication and multicast-like communications are given separately.

A. Unicast Communication

In what follows, we study the behavior of our selected metrics under three different traffic scenarios:

- **Scenario A** – A total of 3 FTP connections are considered where each connection is successively initiated at different points in time;
- **Scenario B** – This scenario analyzes voice traffic performance by employing a total of 2 SCO connections, also initiated at different points in time;
- **Scenario C** – This is a mixed scenario with a total of 3 connections initiated in succession, where the first one is a SCO connection, the second one a Telnet, and the third one an FTP connection.

**Aggregate Throughput:** Figures 6(a), 6(b), and 6(c) compare the aggregate throughput for DSA and existing Bluetooth for the three scenarios. In general, we can observe that DSA always achieves higher throughput than existing Bluetooth. More specifically, at some points DSA experiences an average of up to 300% improvement. The reason is simple: while in Bluetooth all traffic has to go through the master, path becomes non-optimal and bandwidth is wasted, hence compromising the number of data bytes received per unit time. On the other hand, DSA employs direct slave-to-slave communication, thus avoiding additional delay due to packet transmission, propagation time, and queuing, and hence boosting the number of data packets received per unit time.

In Figure 6(a), we see that with one FTP connection Bluetooth can still provide a performance comparable with that of DSA as the delay effects are minimized. However, when the number of connections increases to two and three, Bluetooth can no longer sustain the same performance whereas DSA scales approximately linearly.

**Delay:** The delays experienced in each of the three scenarios are illustrated in Figure 7(a), 7(b), and 7(c) respectively. Due to direct communication between slaves, and hence an optimal communication path, DSA provides reduced delay in all scenarios, with delays being as low as one-third of the current delay in Bluetooth. Note that in scenarios A and C
where we have traffic connections involving TCP, the delay increases as we increase the number of connections. This is due to the burstiness of TCP traffic which increases the average queue length with every burst. On the other hand, this is not the case in scenario B where only SCO connections are considered and the traffic pattern follows a constant bit rate (CBR) stream. Here, the delay for DSA is constant due to our direct slave-to-slave communication, while in Bluetooth the delay initially increases rapidly and then tends to stabilize. Note that in all scenarios, the delay experienced by the DSA scheme is dramatically lower than that of the Bluetooth.

**Overhead:** Figures 8(a), 8(b), and 8(c) compare the overhead of Bluetooth and DSA for scenarios A, B, and C respectively. Here, we see that although DSA transmits three broadcasts for each message schedule, the associated overhead is always approximately half of that of Bluetooth in all the three scenarios. The reason for this is that the Bluetooth’s overhead is present for every packet since all of them have to be forwarded by and routed through the master, hence incurring an additional transmission. On the other hand, in DSA the overhead is only tied to the broadcast message while all data packets are directly sent to the intended destination.

In Figure 8(b), we can see that the overhead of Bluetooth for scenario B remains nearly the same as in the DSA while only one SCO connection is present in the piconet, whereas it increases rapidly when the second SCO connection arrives while DSA maintains a proportional grade of service. There is an important reason behind that. Current Bluetooth can support at most three simplex SCO connections, that is, having the master as either a source or a destination of SCO traffic. Therefore, when it comes to duplex connections, that is, from slave-to-slave, Bluetooth can support only one SCO connection. With our DSA scheme employing direct slave-to-slave communication, we are able to overcome this restriction and enable up to three duplex SCO connections in a piconet.

**Power Consumption:** Figure 9 shows the power consumption comparison of DSA and Bluetooth for the scenarios under consideration. In all the curves, we can clearly notice a dramatic reduction in power consumption at the master device in the DSA scheme. As we can see from Figure 9, as with DSA the master does not get involved in the communication between slaves, this results in a significant increase in the energy efficiency (Mbytes/Joule). Table I illustrates the average power consumption improvement of DSA both at the master as well as in the slaves for.

As we can see from Table I, the average energy saved at the slaves is mainly due to the reduction in the number of polls by the master device, as these slaves are involved in the data communication. Hence, power consumption at the slaves is much less as compared to the master device where energy conservation is immense as a result of DSA (see Table I). Finally, it is important to note the energy-per-byte improvement ratio of DSA over Bluetooth in Figure 9. In other words, with the adoption of DSA, a significantly larger amount of data can be transmitted with the same energy. As a result, the lifetime of a piconet is substantially increased.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Master</th>
<th>Slaves</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>83.06%</td>
<td>2.12%</td>
</tr>
<tr>
<td>B</td>
<td>43.62%</td>
<td>8.95%</td>
</tr>
<tr>
<td>C</td>
<td>54.36%</td>
<td>6.60%</td>
</tr>
</tbody>
</table>

**B. Multicast Communication**

In this section we study the multicast support in DSA and compare it with the existing way of implementing group communication in Bluetooth by multiple unicasts, while we cannot compare DSA with a piconet-wide broadcast given the lack of multicast group information in Bluetooth. For this experiment, we consider two multicast sessions wherein each of the two multicast sources has exactly three other devices as group members. Here, the first multicast session is initiated at simulation startup, while the second is initiated 100 seconds later. Exponential traffic sources are considered.

**Aggregate Throughput:** Figure 10 compares the aggregate throughput of DSA and Bluetooth. As expected, DSA outperforms Bluetooth by employing a single transmission for all group members, while Bluetooth transmits a packet per destination. As a matter of fact, most of the well-known benefits of multicasting [17, 27] over multiple unicasts are embodied in the DSA, making it more efficient than Bluetooth. At some points, DSA improves throughput by up to 500%.

**Delay:** Delay is another aspect in which the multicasting mechanism built in DSA, brings it to a minimum level. As we can see from Figure 11, delay in DSA is practically constant due to its direct one-to-many communication, whereas it is approximately 30 times higher in Bluetooth.

**Overhead:** As Bluetooth employs multiple unicast in order to provide a group-like communication mechanism, it is expected to have a much higher overhead than DSA. Figure 12 confirms this assertion by revealing that DSA experiences around one-seventh the overhead of Bluetooth, despite the control messages broadcasted in DSA.

**Power Consumption:** Finally, Figure 13 compares the power consumption in Bluetooth and DSA. Similar to the unicase, multicasting with DSA also significantly reduces the power consumed at the master device as well as at all slaves. On an average, the master experiences a 54.17% reduction in energy consumption, while the slaves have a 33.05% reduction. This, in turn, prolongs the lifetime of the overall network by enhancing the energy efficiency (Mbyte/Joule) ratio of DSA over Bluetooth (Figure 13).

VI. EDSA SIMULATION RESULTS

In this section we evaluate the EDSA scheme. For simplicity we do not show multicast communication simulation results, as they would not provide much more insight than the ones already presented in the previous section and the ones to be described next. Tables 2, 3 and 4 list the connection patterns employed in three scenarios D, E, and F, respectively, being considered in this evaluation, while the connections are listed in the order in which they are initiated in the piconet. These three scenarios are as follows:
Figure 6. Aggregate throughput for the three scenarios considered

Figure 7. Delay for the three scenarios considered

Figure 8. Overhead for the three scenarios considered

Figure 9. Power consumption comparison of DSA and Bluetooth

Figure 10. Aggregate throughput comparison of DSA and Bluetooth

Figure 11. Delay comparison of DSA and Bluetooth

Figure 12. Overhead comparison of DSA and Bluetooth

Figure 13. Power consumption of DSA and Bluetooth
• Scenario D – Here, we consider a total of five FTP connections with their connection pattern as illustrated in Table II. As we can see from the connections QoS requirements, the piconet master will only be able to satisfy the QoS for the first three FTP connections, while it will have to partition the system upon arrival of the fourth one as all the slots would be reserved. Moreover, further expanding the slot cycle would violate the QoS demands of the existing connections. Therefore, the piconet has to be partitioned in order to satisfy the QoS of all the connections.

• Scenario E – A total of four slave-to-slave SCO connections are considered and their connection patterns are depicted in Table III. This is to say that EDSA will only partition the system upon arrival of the third connection, as it can support the QoS demands of up to two simplex slave-to-slave connections by employing direct slave-to-slave communication, whereas current Bluetooth can support at most one simplex slave-to-slave SCO connection [1] rejecting all others. Therefore, in EDSA the final topology will consist of two piconets running two SCO connections each, hence satisfying the applications QoS demands.

• Scenario F – In this scenario we consider a total of five connections, where three of them are FTP and the other two are SCO. Table IV presents such connections from which we can conclude (from the packet types specified) that the third and fifth connections are SCO, while the others are the FTPs. From the QoS specified in each of these connections requests, we see that the system will be partitioned only once when the second, and last, SCO connection arrives. This is due to the fact that the master is able to properly expand the slot cycle to accommodate the existing FTP connections together with the SCO connection without the need to go for partition. However when the second SCO connection arrives, EDSA is faced with a similar situation as in scenario E and hence initiates the partitioning algorithm.

It is important to note that EDSA performance is expected to be similar to DSA performance before a partition takes place. That is, the gain of EDSA over DSA will become apparent once a partition is performed given that a higher number of connections can be admitted into the network with their required QoS, while this not the case in DSA. Similar to all previous graphs, in the plots that follow we explicitly indicate the exact instant at which a connection request is issued, thus enhancing their readability and comprehension.

**Aggregate Throughput:** Figures 14(a), 14(b), and 14(c) depict the aggregate throughput for the scenarios. From these figures we can see that, similar to DSA, EDSA performance is always superior to Bluetooth. However when the partition occurs (according to the scenarios previously described), EDSA average aggregate throughput is further improved over DSA given that it is able to accept the new connection request while this would not be the case in DSA and Bluetooth as they reject the new connection request due to lack of resources. This trend is observed in all graphs under consideration.

<table>
<thead>
<tr>
<th>TABLE II. Scenario D connection pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection</td>
</tr>
<tr>
<td>S_i ↔ S_j</td>
</tr>
<tr>
<td>S_j ↔ S_k</td>
</tr>
<tr>
<td>S_i ↔ S_k</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLE III. Scenario F connection pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection</td>
</tr>
<tr>
<td>S_i ↔ S_j</td>
</tr>
<tr>
<td>S_j ↔ S_k</td>
</tr>
<tr>
<td>S_i ↔ S_k</td>
</tr>
</tbody>
</table>

**Delay:** The delay performance of EDSA is another metric in which EDSA is expected to show improvement over DSA and Bluetooth, especially when partitioning takes place. Figures 15(a), 15(b), and 15(c) present the average delay for the three scenarios. Let us analyze the scenario of Figure 15(b) where we see that, upon partitioning, EDSA can significantly cut down the average delay and can effectively support all four SCO connections, whereas DSA can support only the two first connections within a single piconet while Bluetooth stops in a single slave-to-slave connection. A similar conclusion is reached when we study the results of the scenarios in Figures 15(a) and 15(c) which show that EDSA surpasses DSA upon partitioning, while DSA is shown to exceed Bluetooth.

**Overhead:** Figures 16(a), 16(b) and 16(c) show the overhead of Bluetooth, DSA, and EDSA for the scenarios being analyzed. Although a higher number of piconets are likely to be generated when using EDSA in order to accommodate a larger number of connections, we see that EDSA overhead is only slightly higher than DSA in a multiple piconets environment. This is due to the fact that partitioning in EDSA takes very few bytes, which makes EDSA and DSA overhead to be practically the same in the long run. At the same time, EDSA overhead is seen to be less than Bluetooth only up to a given number of piconets when the overhead generated in EDSA by broadcasting the scheduling message in all piconets surpasses that of a single piconet Bluetooth network. Also, for similar reasons to those already discussed, DSA overhead is continuously lower than Bluetooth.

**Power Consumption:** Finally, Figures 17(a), 17(b) and 17(c) compare the average power consumption of the selected scenarios. From Figure 17(a) we see that power consumption at the master for EDSA is less than in Bluetooth despite the fact that EDSA masters spends extra power partitioning piconets so as to accommodate additional connections. However, once the piconet partitioning is concluded, direct communication takes place amongst the devices whereas Bluetooth rejects any new
connection request exceeding its capacity and keeps on concentrating packet forwarding at the master device. On the other hand, at the slave side, Bluetooth is observed to consume less power than EDSA given that only very few connections can be admitted into the network, hence limiting the slaves’ power consumption. As for DSA, power consumption at the slaves is superior to Bluetooth and is less than EDSA since the master in DSA is able to achieve optimal resource allocation within the boundaries of a piconet, hence outperforming Bluetooth by supporting a larger number of connections amongst the slaves, but cannot go beyond a piconet as EDSA. A similar behavior is seen in the results of scenarios E and F depicted in Figures 17(b) and 17(c) respectively.

VII. RELATED WORK

Recent research studies have addressed the issue of Bluetooth piconet performance improvement from different angles. From the scheduling perspective, [3, 4, 5] pointed out the drawbacks of existing scheduling techniques. However, the issue of packet forwarding through the master is not addressed.

[8, 9, 10, 11, 12] proposes algorithms to generate the scatternet topology with properties such as limiting the maximum roles and degrees of any node, evenly distributing topology generation, limiting the number of piconets and the maximum number of hops between any pair of devices, and so on. However, the major shortcoming of these approaches is the disregard of traffic characteristics when building the topology. That is, even if any two slaves need to communicate frequently, there is no guarantee that they will be within the same piconet in the resulting scatternet.

Another solution would be piconet partitioning for every new connection, wherein a pair of slaves forms a piconet by themselves if frequent communication is taking place between them [14]. However, this approach incurs additional problems such as scatternet scheduling and, more importantly, this cannot be done indefinitely as interference levels may become unacceptable [6, 16, 21, 23]. Additionally, this solution makes a very unrealistic assumption that all connections are pair-wise distinct nodes in order to piconet partitioning to be successful. In our proposed schemes we do partition piconets, but we do it in a much less greedy fashion taking into consideration other factors such as number, type and topology of connections.

Finally, [15] proposes a time-slot leasing approach where slots are allocated to slaves for direct communication. While this scheme has a few points in common with our proposal, there are fundamental differences. First, SCO connections are not evaluated. While this greatly simplifies design, it is unrealistic to assume that SCO traffic will not be present [2]. Second, the mechanism in [15] does not provide for multicast-like communication. Third, this scheme does not provide any mechanism to adjust bandwidth allocation as traffic demand increases or decreases, while our proposed mechanisms dynamically adjust the slot cycle so as to meet traffic needs.

VIII. CONCLUSIONS

By virtue of the Master/Slave communication model, Bluetooth medium access provides for simplicity, low power, and low-cost, these being the major forces driving the technology. However, this medium access scheme brings with it several problems such as inefficient packet forwarding and wastage of bandwidth by forwarding through the master, increased end-to-end packet delay, and additional power consumption at the master unit due to frequent medium access. Moreover, multicast-like communication and effective QoS guarantees to satisfy application demands are not supported.

With that in mind, we have proposed a dynamic scheduling scheme with three novel approaches, namely, the implementation of direct communication between any two slaves within a piconet, the support for multicast-like group communication, and a dynamic QoS-driven partitioning scheme applicable to scatternets. Through extensive simulation, we have shown our schemes to outperform existing Bluetooth implementation by providing better bandwidth usage, lower delay, lower overhead, and improved energy efficiency. Also, the EDSA scheme provides for scalability by supporting a larger number of connections with their desired QoS.

REFERENCES

Figure 14. Aggregate throughput for the three scenarios considered

Figure 15. Delay for the three scenarios considered

Figure 16. Overhead for the three scenarios considered

Figure 17. Power consumption for the three scenarios considered