TECHNICAL RESEARCH REPORT

Performance issues of Bluetooth scatternets and other asynchronous TDMA ad hoc networks

by Theodoros Salonidis, Leandros Tassiulas

CSHCN TR 2002-23
(ISR TR 2002-52)

Performance issues of Bluetooth scatternets and other asynchronous TDMA ad hoc networks

Theodoros Salonidis and Leandros Tassiulas
Institute for Systems Research
University of Maryland at College Park

***Abstract*—**
In this paper we address a practical performance issue arising in wireless ad hoc networks using time division multiple access (TDMA). This issue relates to the degradation of the network ability to allocate bandwidth when the usual assumption of system-wide slot synchronicity does not hold.

The problem is investigated for the case of Bluetooth, a new promising TDMA wireless technology that enables the formation of ad hoc networks called scatternets. A scatternet does not support a global slot synchronization mechanism. Instead, nodes are grouped in multiple channels called piconets and each piconet uses a different time slot reference provided by a node designated as master. Traffic forwarding between piconets is performed in a time division fashion by bridge nodes that are aware of the time references of the piconets they participate in. Due to the inherent lack of global synchronicity, slots are wasted when bridges switch piconet time references. This translates to a bandwidth loss compared to an ideally synchronized scatternet. While the existence of this asynchronicity overhead has been reported in previous work, there has been no effort to minimize it or even evaluate its possible extent. Nevertheless, these issues are very important for the deployment of practical TDMA-based ad hoc networks.

We consider a scatternet that allocates bandwidth to its links using an asynchronous periodic conflict-free TDMA schedule. In this setting the asynchronicity overhead is manifested as an increase in the minimum period required to realize a demand allocation when compared to a perfectly synchronized system.

We first derive a general upper bound on the overhead of any scatternet and demand allocation. Then we consider the problem of minimizing the overhead given a scatternet configuration and demand allocation on its links. We cast this as a combinatorial optimization problem and propose two algorithms for its solution. The first algorithm reaches the optimal solution using exhaustive search. However this approach can be computationally prohibitive for large problem sizes. To this end, we introduce a second practical heuristic algorithm of polynomial complexity. Using simulations, the heuristic is shown to perform very well for problem sizes where the optimal can be computed. For large problem sizes we use the heuristic to investigate the effect of the various system parameters on the generated overhead and evaluate its performance with respect to the derived general upper bound.

I. Introduction

A wireless ad hoc network is a collection of nodes equipped with radio interfaces and form a multi-hop wireless infrastructure without the aid of any centralized administration. Time division multiple access (TDMA) is a well known medium access scheme for deterministic bandwidth allocation and quality of service provision in ad hoc networks. According to TDMA, bandwidth can be allocated to the network links using a schedule of period T slots. At every slot of such a schedule, several links are activated for transmission such that no conflicts occur at the intended receivers. Then the amount of conflict-free slots a link gets within a period T determines its allocated bandwidth.

A central performance issue arising in a TDMA-based ad hoc network is the determination of the set of allocations it can achieve. A demand link rate allocation $\boldsymbol{r} = [r_l]$ $(0 \leq r_l \leq 1)$ is *feasible* if the network can allocate $\tau_l = \lfloor r_l \cdot T \rfloor$ conflict-free slots to every link l without exceeding the system period T. This decision problem is intrinsically coupled with an optimization one: Find a link schedule of minimum period that realizes slot allocation $\boldsymbol{\tau} = [\tau_l]$. If the solution of this problem is less that the system period then the allocation is feasible, otherwise it is not. The link schedule optimization problem has been studied for ad hoc networks using a single broadcast channel [2] and ones using multiple point-to-point channels at the physical layer [3]. In the first case the problem has been shown to be NP-complete while in the second it can be solved by a polynomial algorithm of high complexity. The works in [7][6] provide heuristic techniques of lower complexity that compute suboptimal link schedules.

The above performance studies as well as most proposed centralized or distributed TDMA protocols for slotted ad hoc networks assume that the time slot boundaries are provided by a global system clock. This system-wide synchronization mechanism is not always possible to achieve in the distributed ad hoc network setting. This is the main reason that some current standardized technologies for ad hoc networks such as WLAN use asynchronous random access schemes (802.11 DFWMAC). Nevertheless, random access cannot provide with any bandwidth allocation guarantees to the network.

Bluetooth [1] is a new TDMA wireless technology that enables the formation of ad hoc networks called scatternets. While being a slotted system, Bluetooth has the interesting feature of not supporting a global slot synchronization mechanism. Instead, time reference is provided locally for each link by one of the node endpoints acting as ”master”. A node that acts as slave to more than one links switches time reference when it needs to talk to a new master. When this happens, a slot must be wasted by the slave for tuning to the time reference of the link master. This phenomenon has been reported in works related to scatternet scheduling [9] [10] [11] [12] [13] [15] as a source of overhead. However there has been no attempt to formally study its effect in the ability of the system to allocate bandwidth. This ability is linked to the determination of the feasible allocations region, or equivalently, the solution of the related link schedule optimization problem.

Given a demand allocation, the minimum period achieved by an asynchronous TDMA system is expected to be greater than the one achieved by a perfectly synchronized one. This is because the various time reference switches over time can have a cumulative additive effect on the overall minimum period required by the asynchronous system. The increase in the minimum period is essentially the overhead introduced by the system asynchronicity.

Based on the above observation, we can use a two step approach to address the link schedule optimization problem for the asynchronous TDMA scattternet setting. The first step assumes perfect synchronization and finds a synchronized link schedule of minimum period that realizes the demand allocation. Bluetooth falls in the category of multi-channel systems studied in [3] and the algorithms and results therein can be used for this purpose.

The second step (which is the contribution of this paper), uses the optimal synchronized schedule as a reference to find an asynchronous schedule of minimum overhead. It turns out that the overhead depends on the order of link activations in the reference synchronized schedule. We introduce two algorithms for addressing this problem. The first algorithm derives a minimum overhead asynchronous schedule for a specific link activation ordering of the synchronized schedule. It also has an upper bound for the overhead it generates for any possible input ordering or scatternet configuration. Using this algorithm it is possible to reach the optimal solution by executing it over all possible orderings. This leads to a problem of combinatorial nature that prohibits exhaustive search for large problem sizes. To this end we introduce a second heuristic algorithm of polynomial complexity. The heuristic is shown to have excellent performance for problem sizes where the optimal can be computed. For large problem sizes we investigate the effect of the various system parameters to the generated overhead and compare the heuristic performance to the derived upper bound.

The rest of the paper is organized as follows. Section II is an introduction to the architecture of Bluetooth scatternets and related work on their scheduling. Section III introduces a scheduling framework for allocating bandwidth in the asynchronous scatternet setting by means of periodic conflict-free link schedules. Section IV and V provide the formulation of the asynchronicity overhead problem and the algorithms used for minimizing it. Section VI evaluates the algorithm performance and investigates the effect of various factors affecting the asynchronicity overhead. Finally, section VII concludes the paper.

II. Piconets and Scatternets

Every Bluetooth unit has an internal ”native” system clock that determines the timing and hopping of the radio transceiver. The native clocks of different Bluetooth nodes are not synchronized and differ by a phase. Clock synchronization happens only locally when nodes are grouped in multiple communication channels called ”piconets”. In each piconet, one unit assumes the role of master while the others act as slaves. The master defines the piconet frequency hopping sequence and provides its native clock as the piconet time reference.

Within a piconet, a master controls access to the channel by polling slaves according to a slotted Time Division Duplex (TDD) protocol. According to this protocol, each piconet slot consists of a master-slave pair of half-duplex mini-slots and therefore supports full duplex communication. All slaves know the slot start time and the frequency hop of a piconet slot and listen passively for a poll by the master. The master then selects one of the slaves by polling it during the first half of the piconet slot and then the slave can respond in the second half.

Piconets can be interconnected via bridge nodes to form a bigger ad hoc network known as a ”scatternet”. Bridge nodes can timeshare between multiple piconets, receiving data from one piconet and forwarding it to another. There is no restriction on the role a bridge node can play in each piconet it participates in. A bridge node can be a master in at most one piconet and slave in another (termed as M/S bridge) or a slave in all piconets (termed as S/S bridge).

The Bluetooth technology standard [1] has not yet specified the way bridges should schedule their visits in different piconets and there is currently an intense research effort on this topic. The emphasis is on distributed scheduling schemes and the approaches can be categorized according to the degree of coordination they offer. According to ”hard coordination” schemes [13][15], the link scheduling is performed in such a way that when a master polls a slave on a Bluetooth link, this slave is guaranteed to be listening on this piconet. Since no transmission conflicts exist, these schemes can potentially achieve strict bandwidth allocation guarantees. However, there is an associated implementation and communication complexity for maintaining the conflict-free property, especially when the scatternet becomes highly dynamic. Soft coordination schemes [11][10][12] trade-off perfectly conflict-free transmissions for lower complexity. The downside here is that this comes to a loss of the ability to provide bandwidth guarantees.

While there is still a simplicity vs performance debate between the two approaches, the bandwidth loss due to piconet switching always exists due to the asynchronous nature of Bluetooth. In the next section we introduce a hard coordination scheduling framework for overhead minimization. There are mainly two reasons for doing this. First, in this case the overhead is naturally linked to the ability of the system to allocate bandwidth. Second and most important is that a coordinated scheduling approach is the best we can do for minimizing the overhead and therefore provides a useful point of reference.

III. Scatternet Communication model

The scatternet is represented as a directed graph $G(N, E)$. A directed edge $(i, j) \in E$ signifies that nodes i and j are within wireless range and they have established a Bluetooth link where i is the master and j the slave.

Each full-duplex piconet slot supports bidirectional communication initiated by the master node: During the first half of the slot the master polls and during the second half the slave responds if polled by the master. Each Bluetooth node has the following operating restrictions:

- **(R.1):** A slave cannot be listening as slave to more than one piconet at the same slot.
- **(R.2):** A node cannot poll as master and be listening to another piconet as slave at the same slot.
- **(R.3):** A master must not poll more than one slave at the same slot.

The first two constraints are due to the single radio transceiver in Bluetooth units, while the third is due to the requirement of conflict-free communication in both directions of a full-duplex piconet slot[1]. These constraints imply that during a slot, a Bluetooth node can use *at most one* of its adjacent links either as master (transmit a poll during the first half of the slot and listen for a response in the second half) or slave (start listening for a poll and respond during the second half of the slot if a poll is received).

A. *Conflict-free bandwidth allocation in scatternets*

Each node i allocates slots to its adjacent links by maintaining a local link schedule $\boldsymbol{S}_i$ of period T_{system}. Each slot entry in $\boldsymbol{S}_i$ corresponds to a full-duplex piconet slot of duration of 1.25ms [2]. The local schedule is with respect to the node's own native clock tick and the node uses it to determine its communication action for the duration of every slot entry: it can either be active on a link (acting as master or slave) or remain idle.

The slot boundaries of different local schedules are not aligned in time. Link endpoint nodes maintain a relative phase-with respect to each other in order to know which slot positions overlap in their local schedules[3]. If node i maintains a relative phase $\phi_{ij} = -1$ with respect to j, then slot position p in $\boldsymbol{S}_i$ overlaps with slot positions $p-1, p$ in $\boldsymbol{S}_j$. If $\phi_{ij} = 1$ then p in $\boldsymbol{S}_i$ overlaps with p and $p+1$ in $\boldsymbol{S}_j$. A $\phi_{ij} = 0$ indicates that the nodes happen to be perfectly synchronized. The relative phase maintained at the other link endpoint j is $\phi_{ji} = -\phi_{ij}$.

Communication is successful on a link (i, j) only if both sides assign time-overlapping slots in their local schedules. The assignment must be such that when the master i starts polling in slot p of $\boldsymbol{S}_i$, the slave j must have assigned slots $p + \frac{\phi_{ij}(1+\phi_{ij})}{2} - 1$ and $p + \frac{\phi_{ij}(1+\phi_{ij})}{2}$ in $\boldsymbol{S}_j$ for listening to this master. In general, for successful conflict-free communication on x consecutive full-duplex piconet slots, the master must allocate x slots in its local schedule for polling, while the slave must allocate at least $x + 1$ time-overlapping slots in its local schedule for tuning to the piconet of this master. Therefore, certain slots in a local schedule may not be used for communication but for aligning to different piconet time references. The positions of such slots are the ones where the node must switch to a new piconet and act as a slave[4].

Due to the extra switching slots needed by the slaves, each link endpoint will allocate a different number of slots for this link in its local schedule. The slots where transmissions take place on the link are the ones allocated by the master. More specifically, if $\tau_{i\to j}$ is the number of slots each node i has allocated for link (i, j) in its local schedule $\boldsymbol{S}_i$, the *slot allocation* $\boldsymbol{\tau} = (\tau_{ij})$ realized by the asynchronous network link schedule S is determined by:

$$\tau_{ij} = \begin{cases} \tau_{i\to j} & \text{if } node\ i\ is\ the\ master\ of\ link\ (i,j) \\ \tau_{j\to i} & \text{if } node\ j\ is\ the\ master\ of\ link\ (i,j) \end{cases} \quad (1)$$

Figure 1 illustrates a scatternet where nodes A and C act as

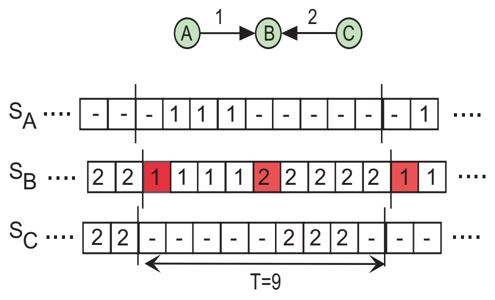


Fig. 1. A simple scatternet topology with two piconets. Nodes A and C act as masters of B (B is an S/S bridge). Each node operates according to a local periodic schedule of $T = 9$ slots. The switching slots in B's local schedule are marked in red.

masters and B acts as a (S/S) bridge. The asynchronous schedule realizes a slot allocation of 3 slots per link over a period of 9 slots. Two slots in S_B are used for switching between the piconets of masters A and C.

IV. The Asynchronicity Overhead

A synchronized system will need a smaller (or at least equal) period than an asynchronous one for realizing the same slot allocation. When global synchronicity is not present, bridge nodes need extra switching slots in their local schedules for supporting the same slot allocation in their adjacent links. This may force an increase in the overall system period for realizing the same allocation.

Figure 2 illustrates a representative example where a slot allocation of 3 slots per link is required for the scatternet topology of Figure 1. If the scatternet were synchronized, this allocation could be realized by a link schedule of minimum period of 6 slots. However, both asynchronous schedules of Figure 2 need a larger period for realizing this allocation. This is due to the extra switching slots needed by bridge node B.

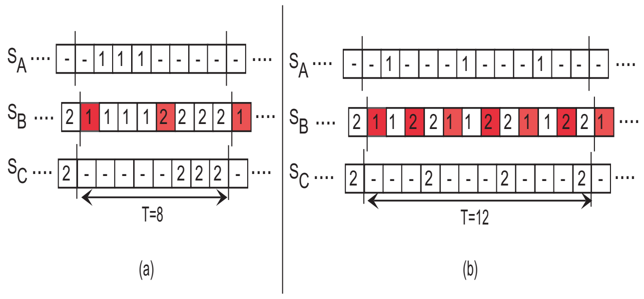


Fig. 2. Two asynchronous schedules realizing slot allocation $(3, 3)$ for the scatternet of Figure 1.

Another observation derived from this example is that for the same demand allocation, the bridge node needs a different

[1]According to the TDD polling protocol, if the master polled more than one slaves during the first half of the slot, these slaves would automatically respond in the second half causing a reception collision at the master

[2]Each half-duplex slot lasts $0.625ms$

[3]The relative phases can be acquired during the Bluetooth link establishment.

[4]If the node operates as master in all of its adjacent links, there are no switching slots in its local schedule

amount of switching slots in the asynchronous schedules (a) and (b), depending on the order links are activated. In (a) the bridge node switches piconets only once during the period (and thus needs only two additional switching slots with respect to the reference synchronized system,) while in (b) it switches piconet every other slot yielding a higher required system period of 12 slots.

According to the above example, the *asynchronicity overhead* for realizing a given slot allocation can be defined as an increase in period with respect to a perfectly synchronized system. Also, the amount of overhead depends on the link activation order. The problem then is to find an asynchronous schedule and link activation order of minimum overhead. The following sections provide with a formulation of this problem and an approach for solving it.

A. Synchronized link schedules and their instances

Consider a scatternet that is synchronized on a slot basis. During a slot, we say that a link is activated for full-duplex communication if both endpoints have assigned it to this slot in their local link schedules. A set of links that can be activated at the same slot without transmission conflicts at the intended receivers is called a *link activation set*. According to the scheduling constraints $S.1-S.3$, along with the fact that Bluetooth is a multi-channel system, a link activation set consists of links that do not have common node endpoints[5].

A synchronized link schedule $\tilde{\boldsymbol{S}}$ of period $\tilde{T}$ is a periodic sequence of link activation set instances $(A_1, ..., A_k, ..., A_{\tilde{T}})$. Let $\mathcal{M}$ be the set of all distinct link activation sets in the network topology and $M(\tilde{\boldsymbol{S}}) \subseteq \mathcal{M}$ be the distinct link activation sets that appear in schedule $\tilde{\boldsymbol{S}}$. Each link activation instance A_k corresponds to an element of $M(\tilde{\boldsymbol{S}})$. Then $\tilde{\boldsymbol{S}}$ can be compactly represented by the distinct sets M^a in $M(\tilde{\boldsymbol{S}})$ and the number of instances λ^α of each set M^a within the schedule period $\tilde{T}$:

$$\tilde{\boldsymbol{S}} = \{(M^\alpha, \lambda^\alpha) : \ M^\alpha \in \mathcal{M}, \, \alpha = 1, 2, ..., |M(\tilde{\boldsymbol{S}})|, \quad \lambda^\alpha \in \{1, ..., \tilde{T}\}\} \tag{2}$$

The number of slots allocated to each link (i,j) during a period $\tilde{T}$ is given by the slot allocation vector $\tilde{\boldsymbol{\tau}}$:

$$\tilde{\tau}_{ij} = \sum_\alpha \lambda^\alpha I((i,j) \in M^\alpha), \ \ \forall \, (i,j) \in E \tag{3}$$

where $I(.)$ is an indicator function that evaluates to one when its argument is true and to zero otherwise.

The synchronized schedule $\tilde{\boldsymbol{S}} = \{(M^\alpha, \lambda^\alpha)\}$ has $\tilde{T}!$ instances that result from all possible permutations of the link activation set instances A_i. The synchronized schedule instance $\tilde{\boldsymbol{S}}^{(\pi)}$ corresponding to permutation π is given by:

$$\tilde{S}^{(\pi)} = (A_{\pi(1)}, ..., A_{\pi(\tilde{T})}). \tag{4}$$

where π is a mapping of the instance indexes $\pi : \{1, ..., \tilde{T}\} \rightarrow \{1, ..., \tilde{T}\}$.

Figure 3 illustrates two instances of a synchronized schedule for the topology of Figure 1.

[5]Since each piconet channel uses a different frequency hopping sequence we assume that there are no conflicts among transmissions that happen between different co-located master-slave pairs.

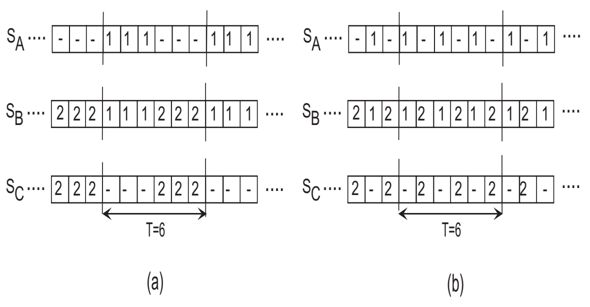


Fig. 3. Two instances of synchronized schedule $S = \{(\{1\}, 3), (\{2\}, 3)\}\}$ for the scatternet of Figure 1. The distinct link activation sets are $M^1 = \{1\}$ and and $M^2 = \{2\}$ and each set has 3 activation instances within the 6 slot period.

B. Equivalent schedules

Let $\tilde{\boldsymbol{\tau}}$ be the allocation realized by a synchronized schedule $\tilde{\boldsymbol{S}}$. An asynchronous schedule $\boldsymbol{S}^{(\pi)}$ realizing slot allocation $\boldsymbol{\tau}$ is called *equivalent* to an instance $\tilde{\boldsymbol{S}}^{(\pi)}$ of $\tilde{\boldsymbol{S}}$ if the following conditions hold:

- **(E.1):** Each node activates its adjacent links in $\boldsymbol{S}^{(\pi)}$ in the same order as in $\tilde{\boldsymbol{S}}^{(\pi)}$.
- **(E.2):**$\boldsymbol{\tau} = \tilde{\boldsymbol{\tau}}$.
- **(E.3):** $\boldsymbol{S}^{(\pi)}$ is conflict-free and satisfies the above conditions using the minimum possible period.

As an example, consider the asynchronous schedule of Figure 1 and the synchronized schedule instance in Figure 3(a). Both schedules realize a slot allocation of 3 slots per link (condition E.2) and all nodes activate their adjacent links in the same order in both schedules (condition E.1). However, the two schedules are not equivalent according to the above definition. This is because the asynchronous schedule of Figure 2(a) also satisfies conditions (1) and (2) using a smaller period of 8 slots. In fact, this is the equivalent asynchronous schedule of the synchronized instance of Figure 3(a) because with this link activation order, bridge node B needs at least 2 switching slots in its local schedule to guarantee the time overlap with the master transmissions.

Thus, an equivalent schedule is the minimum period asynchronous schedule for a given ordering of link activations in a synchronized reference schedule.

C. An algorithm for finding equivalent schedules

In this section, we present an algorithm called EQUIVALENT that takes as input a scatternet configuration[6] and a reference synchronized schedule instance $\tilde{\boldsymbol{S}}^{(\pi)}$. The output is the equivalent asynchronous schedule $\boldsymbol{S}^{(\pi)}$ of $\tilde{\boldsymbol{S}}^{(\pi)}$.

The algorithm constructs $\boldsymbol{S}^{(\pi)}$ by iterating over the link activation instances $A_{\pi(1)}, ..., A_{\pi(\tilde{T})}$ of $\tilde{\boldsymbol{S}}^{(\pi)}$. During iteration k, the link activation set instance $A_{\pi(k)}$ is considered. Let l be a link in $A_{\pi(k)}$ and i and j be its master and slave endpoints respectively. Also for each node n, let $p_n^{(k-1)}$ be the maximum slot position assigned so far in its asynchronous local schedule $\boldsymbol{S}_n^{(\pi)}$.

[6]The scatternet configuration consists of the topology graph, master-slave role assignments on the links and relative phases

First the master i must assign[7] to link l the earliest slot position in its local schedule after $p_i^{(k-1)}$ that does not overlap in time with the last assigned slot $p_j^{(k-1)}$ of slave j. There are three cases to consider when computing slot $p_i^{(k)}$ for the master:

- **Case A: Link (i,j) was activated in iteration $k-1$ as well**: In this case the nodes' local schedules are "in synch" due to the previous iteration and the master simply allocates to link l the next slot:

$$p_i^{(k)} = p_i^{(k-1)} + 1 \tag{5}$$

- **Case B: Link (i,j) was not activated in iteration $k-1$ and $p_i^{(k-1)} > p_j^{(k-1)}$**: In this case the master's local schedule is forward in time with respect to the slave's. The earliest non-overlapping slot is again given by:

$$p_i^{(k)} = p_i^{(k-1)} + 1 \tag{6}$$

- **Case C: Link (i,j) was not activated in iteration $k-1$ and $p_j^{(k-1)} \geq p_i^{(k-1)}$**: In this case the slave's local schedule is considered forward with respect to the master, so the master must find the earliest possible non-overlapping slot. Depending on the nodes' relative phase, the position of slot $p_i^{(k)}$ is given by:

$$p_i^{(k)} = p_j^{(k-1)} + \frac{\phi_{ij}^2 - \phi_{ij} + 2}{2}, \; \phi_{ij} \in \{1, 0, -1\} \tag{7}$$

Then i assigns slot $p_i^{(k)}$ to link l. If there are any intermediate unassigned slots between $p_i^{(k-1)}$ and $p_i^{(k)}$ they are assigned as idle in $\boldsymbol{S}_i^{(\pi)}$.

Once the master i updates its local schedule,the slave j must compute the earliest unassigned slot in $\boldsymbol{S}_j^{(\pi)}$ that will exceed in time slot $p_i^{(k)}$ in $\boldsymbol{S}_i^{(\pi)}$. Depending on the nodes' relative phase the position of this slot is computed as:

$$p_j^{(k)} = p_i^{(k)} + \frac{\phi_{ij}(1 + \phi_{ij})}{2}, \; \phi_{ij} \in \{1, 0, -1\} \tag{8}$$

If there are any unassigned slots between $p_j^{(k-1)}$ and $p_j^{(k)}$, they are assigned to link l in $\boldsymbol{S}_j^{(\pi)}$.

The same assignment steps happen for the node endpoints of every other link l in $M_{\pi(k)}$. For every node n that did not update its local schedule during iteration k, $p_n^{(k)} = p_n^{(k-1)}$.

At the end of the iteration k, the algorithm keeps track of the asynchronous schedule *forward progress* $f(k)$ which is the maximum progress over all local schedules after this iteration:

$$f(k) = \max_{n \in N}\{p_n^{(k)}\} \tag{9}$$

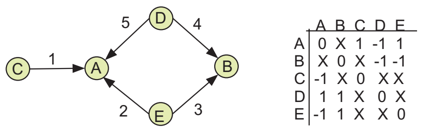
After $\tilde{T}$ iterations, all link activation set instances $M_{\pi(k)}$ have been added to the asynchronous schedule $\boldsymbol{S}^{(\pi)}$ and each node n has assigned $p_n^{(\tilde{T})}$ slots in its local schedule. The asynchronous schedule period $T^{(\pi)}$ must be set to the maximum of these values, which is the forward progress after the last iteration:

$$T^{(\pi)} = f(\tilde{T}) \tag{10}$$

Finally, the algorithm restarts from $M_{\pi(1)}$ and performs one or more extra iterations so that all nodes fill their local schedules up to slot $T^{(\pi)}$. When this happens, all nodes use the first $T^{(\pi)}$ slots in their local schedules to form an asynchronous schedule with this period.

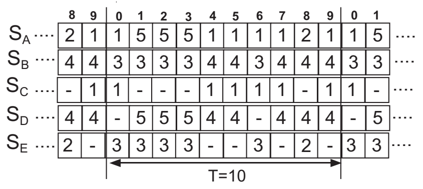
The detailed algorithm pseudocode is provided in the Appendix, while Figure 4 provides an example of the algorithm

[7]In different iterations, the same node may have a different role, depending on the activated link. When the node assigns slots to a link, it also marks the role it will play. If it is a master it will be polling while if it is a slave it will be tuned to the piconet of the link for the duration of these slots.



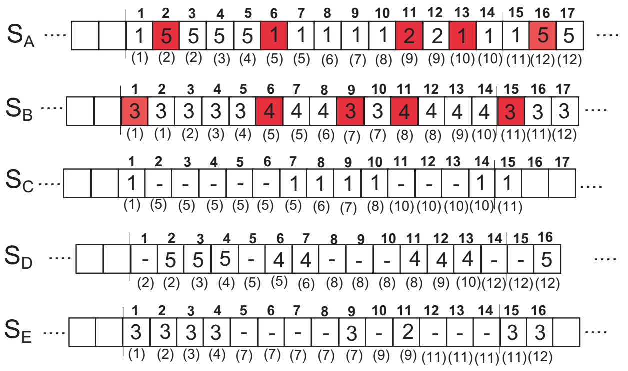
$$\begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & 0 & X & 1 & -1 & 1 \\ B & X & 0 & X & -1 & -1 \\ C & -1 & X & 0 & X & X \\ D & 1 & 1 & X & 0 & X \\ E & -1 & 1 & X & X & 0 \end{array}$$

(a) Scatternet topology: Nodes C,D,E are masters while A,B are S/S bridges. Phase matrix: Entries marked by X are invalid because no link has been formed between the corresponding nodes.



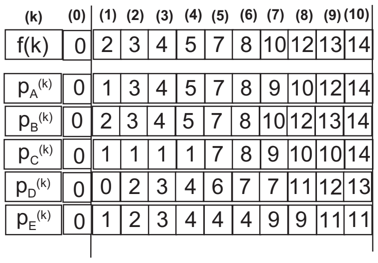
	8	9	0	1	2	3	4	5	6	7	8	9	0	1
S_A	2	1	1	5	5	5	1	1	1	1	2	1	1	5
S_B	4	4	3	3	3	3	4	4	3	4	4	4	3	3
S_C	-	1	1	-	-	-	1	1	1	1	-	1	1	-
S_D	4	4	-	5	5	5	4	4	-	4	4	4	-	5
S_E	2	-	3	3	3	3	-	-	3	-	2	-	3	3

(b) An instance of a synchronized schedule $\tilde{S} = \{(\{1,3\},2), (\{5,3\},3), (\{1,4\},4), (\{2,4\},1)\}$ of (minimum) period $\tilde{T} = 10$.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
S_A	1	5	5	5	5	1	1	1	1	1	2	2	1	1	1	5	5
	(1)	(2)	(2)	(3)	(4)	(5)	(5)	(6)	(7)	(8)	(9)	(9)	(10)	(10)	(11)	(12)	(12)
S_B	3	3	3	3	3	4	4	4	3	3	4	4	4	4	3	3	3
	(1)	(1)	(2)	(3)	(4)	(5)	(5)	(6)	(7)	(7)	(8)	(8)	(9)	(10)	(11)	(11)	(12)
S_C	1	-	-	-	-	-	1	1	1	1	-	-	-	1	1		
	(1)	(5)	(5)	(5)	(5)	(5)	(5)	(6)	(7)	(8)	(10)	(10)	(10)	(10)	(11)		
S_D	-	5	5	5	-	4	4	-	-	-	4	4	4	-	-	5	
	(2)	(2)	(3)	(4)	(5)	(5)	(6)	(8)	(8)	(8)	(8)	(9)	(10)	(12)	(12)	(12)	
S_E	3	3	3	3	-	-	-	-	3	-	2	-	-	-	3	3	
	(1)	(2)	(3)	(4)	(7)	(7)	(7)	(7)	(7)	(9)	(9)	(11)	(11)	(11)	(11)	(12)	

(c) The numbers in parentheses indicate the iteration where the slot was placed by the algorithm on each node's local schedule. Switching slots are indicated by red. The equivalent schedule period is determined at the 10th iteration and is equal to 14. Two additional iterations are performed so that all nodes fill their local schedules up to this period.



(k)	(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
f(k)	0	2	3	4	5	7	8	10	12	13	14
$p_A^{(k)}$	0	1	3	4	5	7	8	9	10	12	14
$p_B^{(k)}$	0	2	3	4	5	7	8	10	12	13	14
$p_C^{(k)}$	0	1	1	1	1	7	8	9	10	10	14
$p_D^{(k)}$	0	0	2	3	4	6	7	7	11	12	13
$p_E^{(k)}$	0	1	2	3	4	4	4	9	9	11	11

(d) Evolution of the $p_n^{(k)}$ and progress $f(k)$.

Fig. 4. An example of the EQUIVALENT algorithm execution

operation.

The algorithm possesses two important properties, summarized by the following theorems:

Theorem 1: The asynchronous schedule $\boldsymbol{S}^{(\pi)}$ derived by algorithm EQUIVALENT is indeed equivalent to the reference synchronized schedule instance $\boldsymbol{S}^{(\pi)}$.

Theorem 2: If $\tilde{T}$ is the period of the reference synchronized schedule instance, the period $T^{(\pi)}$ of any equivalent asynchronous schedule is upper bounded by $2\tilde{T}$.

The proofs can be found in the Appendix. Theorem 2 states that an equivalent asynchronous schedule can have an overhead of at most $\tilde{T}$ slots. This implies the following statement for feasibility of allocations in scatternets:

Corrolary on feasibility: Consider an (asynchronous) scatternet operating with a period T_{system} and a given demand allocation $\boldsymbol{\tau}$. If $\boldsymbol{\tau}$ can be realized by a *synchronized* schedule $\tilde{\boldsymbol{S}}$ of minimum period $\tilde{T}(\boldsymbol{\tau}) \leq \lfloor T_{system}/2 \rfloor$, then $\boldsymbol{\tau}$ is guaranteed to be feasible by the scatternet.

The proof is in the Appendix. The corollary establishes that a scatternet using the EQUIVALENT algorithm can realize at least half of the allocations that can be realized under perfect synchronization. Also for allocations $\boldsymbol{\tau}$ for which the condition $\tilde{T}(\boldsymbol{\tau}) \leq \lfloor T_{system}/2 \rfloor$ holds, any synchronized schedule instance will generate an asynchronous schedule that realizes this allocation. If the condition does not hold for $\boldsymbol{\tau}$, then we must solve the optimization problem addressed in the next section.

V. Optimal equivalent schedules

A. Optimal algorithm

Algorithm EQUIVALENT generates an asynchronous schedule of minimum period $T^{(\pi)}$ for a fixed ordering π of link activation sets in the synchronized schedule. The optimal asynchronous schedule is the one that has the minimum period over all possible permutations π. This schedule can be found if we execute EQUIVALENT for all synchronized schedule instances $\tilde{\boldsymbol{S}}^{(\pi)}$ and selecting the equivalent schedule $\boldsymbol{S}^{(\pi)}$ of minimum period. However an exhaustive search over the $\tilde{T}$! orderings makes this approach prohibitive even for small $\tilde{T}$. [8]

The problem search space can be reduced if we only consider synchronized schedule instances where the λ^{α} link activation instances of each distinct link activation set M^{α} are scheduled in consecutive slots. This is because there are no switching slots generated by EQUIVALENT when $A_{k-1} = A_k$ and the overhead is zero for this transition. Thus, if $M(\tilde{\boldsymbol{S}})$ is the set of distinct link activation sets in the synchronized schedule $\tilde{\boldsymbol{S}}$, we only need to search $|M(\tilde{\boldsymbol{S}})|!$ synchronized schedule instances instead of $\tilde{T}!$. However, for large problem sizes even $|M(\tilde{\boldsymbol{S}})|$ can be prohibitively large for exhaustive search. In this case we use the heuristic approach described next.

[8](According to [8], enumerating 20! possibilities would about 76 years and for 30!, 8×10^{15} years!

B. A heuristic algorithm

MIN_PROGRESS is a heuristic algorithm for overhead minimization that consists of two phases. The first phase determines an ordering π_h of the distinct link activation sets in $M(\tilde{\boldsymbol{S}})$. The second phase first forms a reference synchronized schedule where link activation sets are ordered according to π_h and the λ^{α} instances of M^{α} are activated in consecutive slots. Then EQUIVALENT is used to generate an equivalent asynchronous schedule for this reference synchronized schedule.

We now describe phase I that selects permutation π_h. An asynchronous schedule is constructed using only the distinct link activation sets instead of all their instances. The sets are added to the asynchronous schedule in the same way as instances are added in EQUIVALENT. Upon initialization, an arbitrary set M^{a} is selected from $M(\tilde{\boldsymbol{S}})$ and added to the asynchronous schedule. Let U-set be the set of all unassigned link activation sets and $U^{(k-1)}$ its current contents at the start of iteration k. The addition of each set M^{α} of $U^{(k-1)}$ would generate a forward progress $f(\alpha, k)$ for the asynchronous schedule. The algorithm selects the link activation set of minimum forward progress. If more than one sets generate this progress, one of them is arbitrarily selected and added to the asynchronous schedule. Let M^{α_k} be this set. Then the k-th entry of π_h is set to α_k. At the end of the iteration k, M^{α_k} is removed from the U-set (($U^k = U^{(k-1)} - \{M^{\alpha_k}\}$). The same steps are performed until the U-set becomes empty after $\tilde{T}$ iterations. At that point π_h contains the indices of the activation sets that were selected by the algorithm iterations.

The complexity of MIN_PROGRESS is dominated by phase I whose complexity depends on the number of distinct activation sets $|M(\tilde{\boldsymbol{S}})|$ in the reference synchronized schedule. During iteration k, $|U^{(k-1)}| = |M(\tilde{\boldsymbol{S}})| - k$ sets are considered for addition in the asynchronous schedule. Thus, the total number of link activation sets considered over all the iterations is $(|M(\tilde{\boldsymbol{S}})|-1)+(|M(\tilde{\boldsymbol{S}})|-2)+...+1 = |M(\tilde{\boldsymbol{S}})|(|M(\tilde{\boldsymbol{S}})|-1)/2$, yielding a complexity of $O(|M(\tilde{\boldsymbol{S}})|^2)$.

VI. Performance evaluation

A. Factors affecting the overhead

We are interested in evaluating the performance of the proposed algorithms in view of the factors that affect the asynchronicity overhead. The overhead is first related to the scatternet topology structure. In general, denser topologies are expected to produce more overhead since more links mean more piconet switches. Performance is also affected by the way master-slave roles are assigned in the scatternet topology graph. For example if node B in Figure 1 is assigned as master instead of S/S bridge for nodes A and C, the overhead is always zero since there is only a single time reference in the system.

For a specific scatternet configuration the overhead depends on the demand allocation at hand. A parameter specific to the demand allocation is the ratio $|M(\tilde{S})|/\tilde{T}$ of distinct link activation sets to the period $\tilde{T}$ of the optimal reference schedule. A small ratio is desirable because overhead is generated only during the transitions between distinct activation sets in the synchronized schedule. Another related parameter is the period $\tilde{T}$ of the synchronized schedule. Larger periods may allow for smaller $|M(\tilde{S})|/\tilde{T}$ ratios and therefore less generated overhead.

B. Generating reference synchronized schedules and scatternet topologies

The performance of the algorithms must be evaluated over a variety of scatternet configurations and demand allocations. Demand allocations must be provided by optimal reference synchronized schedules. Determining the minimum period link schedule for a specific allocation is a hard problem even for synchronized systems. Hajek and Sasaki [3] proved that for the class of multi-channel systems where synchronized scatternets belong, the problem can be solved by a polynomial algorithm. However this algorithm is of very high complexity and hard to implement in practice. While [6] provides a practical heuristic with good properties for the solution of this problem, we want to ensure that an optimal synchronized schedule is an input to our algorithms.

The work in [3] also showed that if the network topology graph is bipartite, the minimum synchronized schedule period $\tilde{T}(\boldsymbol{\tau})$ equals the maximum node utilization imposed by $\boldsymbol{\tau}$:

$$\tilde{T}(\boldsymbol{\tau}) = \max_{i \in N} \sum_{j \in N(i)} \tau_{ij}. \tag{11}$$

where $N(i)$ the set of one-hop neighbors of i. The above result provides with a straightforward method for generating minimum period synchronized schedules for bipartite scatternets [9]: If $\tilde{T}$ is the desired period of the reference synchronized schedule, we only need to generate an arbitrary conflict-free schedule of period $\tilde{T}$ and ensure that there is at least one node with utilization equal to $\tilde{T}$.

In the following experiments we consider bipartite topology graphs consisting of $|N|/2$ nodes per bipartite set. This provides a baseline topology of $|N|^2/4$ possible links. A Bluetooth node cannot participate in more than seven active links at a time. As a result the maximum node degree in the network is 7. This restricts the maximum number of links that can be simultaneously established in the baseline topology.

A network designer or scatternet topology construction algorithm may wish keep the maximum number of piconets a node should participate to $B_{max} \leq 7$. The less B_{max}, the less piconets a bridge needs to visit and therefore the less the potential overhead. We also use a parameter f ($0 \leq f \leq 1$) for tuning the density of the scatternet topologies. This parameter is used to generate topologies where f% of the links have been arbitrarily removed from the baseline graph and is used to investigate the effect of topology density when there is no restriction on B_{max}.

Given a topology graph constructed as above, asynchronicity can then be introduced by a link phase matrix and master-slave role assignment. For each link in the scatternet the role assignment determines its master and slave node endpoints. This is the most general role assignment method since after the assignment each node may have a different role on its adjacent links [10].

[9]Bipartite topologies arise very frequently in the Bluetooth setting. For example, a scatternet that uses only S/S bridges (i.e. bridges that act only as slaves in the piconets they participate in) is by definition bipartite.

[10]This method can also capture the more special case where the scatternet consists only of masters and S/S bridges.

C. Performance of MIN_PROGRESS with respect to optimal

Six 20-node bipartite topologies of increasing density are used for this experiment. These topologies consist of 10 masters and 10 S/S bridges. For each topology we randomly generate 100 reference schedules of small period $\tilde{T} = 7$. This period allows the use of exhaustive search on all possible (7!) instances of each schedule and the determination of the optimal solution[11].

Figure 5 compares the average computed optimal period and the one computed by MIN_PROGRESS. For almost all topologies, the period computed by the heuristic exceeds the optimal by less than one slot on the average, while in topology 5 the optimal is exceeded by 1.3 slots.

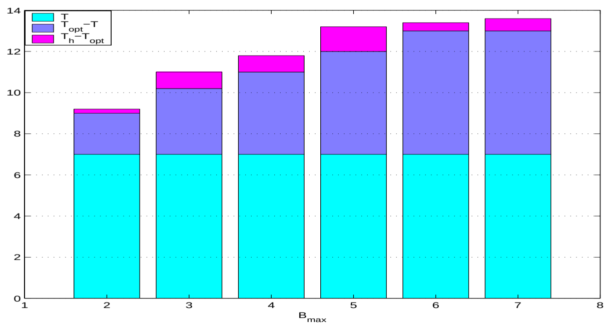


Fig. 5. Each bar graph corresponds to a different topology, where density increases by varying B_{max} from 2 to 7. The reference period is 7 slots. The optimal T_{opt} and the heuristic T_h asynchronous periods of each bar are averages over 100 reference schedules.

Also notice that both the optimal and the heuristic asynchronous periods increase as B_{max} increases. For $B_{max} = 7$ both come very close to the worst case upper bound which is $2\tilde{T} = 14$. This is because the small period $\tilde{T}$ is comparable to B_{max} and bridge nodes with this degree will need to switch piconet at every slot thus causing high overhead.

D. Performance evaluation for large problem sizes

In the following experiments we evaluate the performance of MIN_PROGRESS for larger topologies and reference periodic schedules. We use an $|N| = 100$ *baseline* bipartite topology (50 nodes per bipartite set) where all nodes have the maximum possible degree of seven adjacent links.

For each experiment the overhead is plotted as the %increase in the synchronized system period $\tilde{T}$ due to asynchronicity. If T_h is the period computed by MIN_PROGRESS, this quantity is $\frac{T_h - \tilde{T}}{\tilde{T}}$. A value of 100% denotes that the heuristic reaches an overhead equal to the worst case upper bound.

1) **Effect of topology density***:* Given the baseline $|N| = 100$ bipartite topology, the parameters B_{max} and f are used to derive topologies of variable density. For each set of parameters (B_{max}, f), we generate 10 topologies and for each topology 100 arbitrary reference optimal synchronized schedules of period $\tilde{T}$ ($\tilde{T}$ is also a parameter). Link phases and roles are randomly assigned for each topology.

[11]Since the optimal algorithm considers only the distinct link activation sets $|M(\tilde{\boldsymbol{S}})|$ instead of all the $\tilde{T}$ activation set instances, we ensure that for each generated schedule $\tilde{\boldsymbol{S}}$, $|M(\tilde{\boldsymbol{S}})| = \tilde{T} = 7$.

For each set of parameters (B_{max}, f) and $\tilde{T}$ the figures illustrate the average %overhead over all generated topologies and reference synchronized schedules. Figure 6 investigates the effect of B_{max} ($f = 1.0$). For a fixed $\tilde{T}$, the overhead consistently increases as the maximum number of piconets per bridge increases. For $\tilde{T} = 28$, when B_{max} is restricted to 2 the overhead is 15% but when there is no restriction ($B_{max} = 7$), it reaches 60%. The overhead decreases for larger periods. Thus for $B_{max} = 7$, while the overhead reaches a 60% for $T = 28$ slots, it reduces to 30% for $T = 896$ slots. While this decrease is more drastic for smaller periods (i.e from 28 to 56 slots), it is less in the transition for larger periods (i.e from 448 to 896 slots). This implies that there may still be a non-negligible overhead even if the reference synchronized system operates with a large period.

Similar trends arise in Figure 7 where the topology density is varied without enforcing a particular B_{max}. This shows that the overhead will generally increase with the scatternet density regardless of whether a B_{max} is enforced or not in the system.

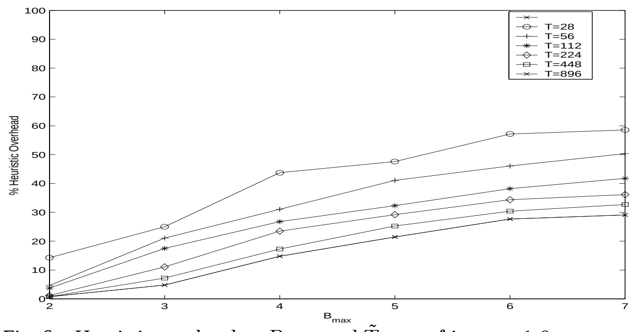


Fig. 6. Heuristic overhead as B_{max} and $\tilde{T}$ vary. f is set to 1.0

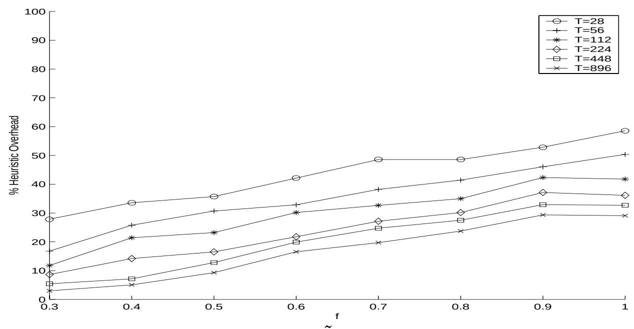


Fig. 7. Heuristic overhead as f and $\tilde{T}$ vary. B_{max} is set to 7

2) **Effect of demand slot allocation***:* The previous experiments investigated the algorithm performance averaged over arbitrary demand allocations and scatternet topologies. A natural question is whether there exists a scatternet role assignment and/or demand allocation for which the generated asynchronicity overhead is maximized. In this section we make a first attempt to informally classify such worst case instances and then test our intuition via simulations.

Let $G(N, E)$ be a bipartite topology graph and $\Psi^{(\tilde{T})}$ the set of all allocations realized by a synchronized schedule of minimum period $\tilde{T}$. For any allocation $\boldsymbol{\tau}$ of $\Psi^{(\tilde{T})}$, let $BN(\boldsymbol{\tau})$ be the set of bottleneck nodes that get the maximum utilization under $\boldsymbol{\tau}$. (Since the graph is bipartite, the maximum utilization equals the minimum period $\tilde{T}$.)

$$BN(\boldsymbol{\tau}) = \{n : arg \max_{i \in N} \sum_{j \in N(i)} \tau_{ij}\}. \tag{12}$$

We conjecture that maximum overhead will be generated if the following conditions hold for a demand allocation $\boldsymbol{\tau}^{max}$ in $\Psi^{(\tilde{T})}$ and at least one of the bottleneck nodes in $BN(\tau^{max})$:

- **P1:** In addition to maximum utilization, the node must have the maximum number of adjacent links in the network.
- **P2:** The node has been assigned as an S/S bridge.
- **P3:** Allocation $\boldsymbol{\tau}^{max}$ is such that the node is requested to allocate an equal number of slots to its adjacent links.

The intuition in the above conditions is that a maximum utilization node will need to be considered at every iteration of an overhead minimization algorithm. Also since this is a node of maximum degree and acts as an S/S bridge it will visit the maximum possible number of piconets in the system (B_{max}). If the requested slots are evenly distributed for this node, then we can show that the overhead will be maximized under the worst permutation if its adjacent link activations. According to [14][15], a maxmin fair allocation in a synchronized multi-channel wireless ad hoc network maximizes the utilization of the maximum degree nodes in the network. If at least one of these nodes is also assigned as an S/S bridge, then the above conditions hold for at least one node in the network.

Figure 8 compares the heuristic overhead resulting from a maxmin fair synchronized schedule and the average heuristic overhead over 100 other schedules realizing arbitrary allocations. (The maxmin fair schedule is computed using the algorithm in [15]). Each point in the bar graphs is the average of the overheads generated by the scatternet topologies of Figures 6 and 7. Each bar graph corresponds to a different synchronized schedule period $\tilde{T}$. As expected, the average heuristic overhead for arbitrary allocations decreases as the system period increases. However, the one due to the maxmin fair allocation does not change significantly and it is in the order of 80% for all cases. This shows that the overhead can be very high for the allocations we identified even if we use an overhead minimization algorithm such as MIN_PROGRESS. A counter-intuitive result is that the overhead remains constant even if the period $\tilde{T}$ increases. Nevertheless, it will always be less than the upper bound $\tilde{T}$ given by Theorem 2.

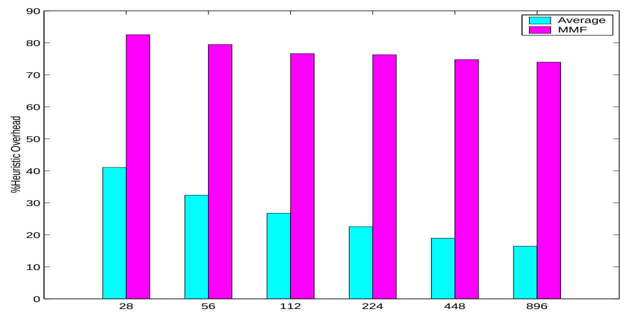


Fig. 8. Comparing the heuristic overhead for maxmin fair allocation and the average overhead generated by arbitrary allocations. Both overheads have been averaged over all topologies considered in Figures 6 and 7

VII. Conclusions

In this paper we addressed for the first time the problem of minimizing the piconet switching overhead in Bluetooth scatternets. This overhead arises due to slots wasted when bridge nodes synchronize to the different piconet time references. While the problem was studied in the Bluetooth context, the results apply to any wireless ad hoc network using slotted TDMA access and multiple local time references instead of a global synchronization mechanism.

It was shown that this overhead can significantly affect the bandwidth allocation ability of a scatternet if no measures are taken to minimize it. To this end we introduced two scheduling algorithms that aim for overhead minimization while ensuring that the generated overhead has an upper bound regardless of the scatternet or demand allocation at hand. The first algorithm reaches the optimal solution but cannot be applied to large problem sizes because it relies on exhaustive search. For large problem sizes a heuristic algorithm was devised and through simulations it was shown to have excellent performance . We also identified certain conditions on demand allocations and scatternet configurations for which the overhead can be high even if an overhead minimization algorithm is run. We outlined the general properties of such allocations and verified our intuition through simulations. A formal study of the exact nature of these allocations is an interesting future work direction.

Both the optimal and heuristic overhead minimization algorithms are centralized and can be used in settings where global information is available. More important though is the fact that that they can provide design insights and be used as a reference performance measure for distributed overhead-aware scatternet scheduling protocols.

Finally, we believe that the derivation of a similar overhead minimization framework for "soft-coordination" scatternet scheduling schemes is another challenging open research issue.

References

[1] Bluetooth Special Interest Group,*Specification of the Bluetooth system, ver 1.0B.* , www.bluetooth.com, October 2000.

[2] E. Arikan, *Some complexity results about packet radio networks*. IEEE Trans. Inform. Theory, Vol. IT-30 pp. 681-685, July 1984.

[3] B. Hajek and G. Sasaki, *Link Scheduling in Polynomial Time*. IEEE Trans. Inform. Theory, No 5, Vol. 34, 1988.

[4] R. Ramaswami and K. Parhi, *Distributed Scheduling of Broadcasts in a Radio Network*. In Proc. IEEE INFOCOM'89, Ottawa, Ont., Canada, Apr. 1989.

[5] A. Ephremides and T. V. Truong, *Scheduling Broadcasts in Multihop Radio Networks*. IEEE Trans. Commun., vol. 38, no. 4, April 1990.

[6] M. Post, P. Sarachik and A Kershenbaum, *A Biased Greedy Algorithm for Scheduling Multihop Radio Networks*. 19th Annu. Conf. on Information Sciences and Systems, Johns Hopkins Univ., March 1985.

[7] J Silvester, *Perfect Scheduling in Multihop Broadcast Networks* In Proc. 6th Int. Conf. on Computer Communications, London, England, Sept. 1982.

[8] B. Korte and J. Vygen, *Combinatorial Optimization*. Springer Verlag, 1991.

[9] G. Miklos et al, *Performance Aspects of Bluetooth Scatternet Formation*. Proceedings of IEEE/ACM MobiHoc, Boston, MA, Aug. 2000.

[10] N.Johansson, F. Alriksson, U. Jonsson, *JUMP mode - a dynamic window-based scheduling framework for Bluetooth scatternets*. Proceedings of IEEE/ACM MobiHoc, Long Beach CA, Oct. 2001.

[11] A. Racz, G. Miklos, F. Kubinszky, A. Valko *A Pseudo Random Coordinated Scheduling algorithm for Bluetooth Scatternets*. Proceedings of IEEE/ACM MobiHoc, Long Beach CA, Oct. 2001.

[12] Simon Baatz, Matthias Frank, Carmen K uhl, Peter Martini, Christoph Scholz, *Bluetooth Scatternets: An Enhanced Adaptive Scheduling Scheme*. Proceedings of Infocom 2002, New York, 2002.

[13] N. Johansson, U. Korner, L. Tassiulas, *A distributed scheduling algorithm for a Bluetooth scatternet*. In Proc. Of the 17th International Teletraffic Congress, ITC '17. Salvador da Bahia, Brazil, Sep. 2001.

[14] L. Tassiulas and S. Sarkar, *Maxmin Fair Scheduling in Wireless Networks*. Proceedings of Infocom 2002, New York, 2002.

[15] ———, *Distributed on-line schedule adaptation for balanced slot allocation in Bluetooth scatternets and other ad hoc network architectures*.

APPENDIX A: Proof of Theorems 1 and 2

Proof of theorem 1: We need to show that the following conditions are satisfied:

1) Nodes must activate the links in the same order in both $\tilde{\boldsymbol{S}}^{(\pi)}$ and $\boldsymbol{S}^{(\pi)}$.
2) The two schedules must realize the same slot allocations.
3) The asynchronous schedule must be conflict-free and of minimum period.

Condition 1 is satisfied because the link activation set instances are added to $\boldsymbol{S}^{(\pi)}$ in a sequential manner. When a link $l = (i, j)$ is considered at iteration k, the master i assigns only one slot to link l. Thus the link masters assign in their local schedules a number of slots equal to the number of slots that are assigned to l in the synchronized schedule. By considering the definition of allocation in an asynchronous schedule (equation (1)), condition 2 holds as well.

Regarding condition 3, when a link l is considered on iteration k, equations (7) and (5) for $p_i^{(k)}$ ensure that the master i assigns the earliest possible slot in its local schedule that does not overlap in time with the last assigned slot $p_j^{(k-1)}$ of slave j. Then, equation (8) for $p_j^{(k)}$ ensures that the slave will assign the smallest possible number of time overlapping slots with respect to $p_i^{(k)}$. Similarly, every other endpoint node for a link of iteration k progresses in its local schedule by the minimum number of slots that guarantee a conflict-free transmission. Thus, at every step k, the forward progress $f(k) = \max_{n\in N}\{p_n^{(k)}\}$ is the minimum possible. Since this property holds for all steps k, it also holds for $f(\tilde{T})$ which is by definition the period of the resulting asynchronous schedule.

Proof of Theorem 2: For this proof we need the following lemmae:

Lemma 1: For every master-slave link (i, j) let $L_{ij}^{(k)} = max\{p_i^{(k)}, p_j^{(k)}\}$. Then the following inequalities hold:

$$L_{ij}^{(k)} - L_{ij}^{(k-1)} \geq 0,\ \forall k = 1, 2, .., \tilde{T}. \tag{13}$$

$$L_{ij}^{(k)} - L_{ij}^{(k-1)} \leq 2,\ \forall k\ where\ link\ (i, j)\ is\ activated \tag{14}$$

Proof: When link (i, j) is activated in iteration k, both nodes i and j assign slots in their local schedule and therefore $L_{ij}^{(k)} > L_{ij}^{(k-1)}$. If nodes i and j are not involved in any link activation during iteration k, then $L_{ij}^{(k)} = L_{ij}^{(k-1)}$ since the p_i and p_j are not updated. Therefore in general $L_{ij}^{(k)} \geq L_{ij}^{(k-1)}$.

We now prove the upper bound. Let link (i, j) where master is i and slave is j be activated in iteration k. If this is the case then due to equation (8), $p_j^{(k)} \geq p_i^{(k)}$ and therefore $L_{ij}^{(k)} = p_j^{(k)}$. We now distinguish 3 different cases that arise when the link (i,j) is activated in iteration k:

- **Link** (i,j) **was activated in iteration** $k-1$: Equation (8) was used in iteration k-1 and therefore $p_j^{(k-1)} = p_i^{(k-1)} + \frac{\phi_{ij}(1+\phi_{ij})}{2} \geq p_i^{(k-1)}$. Therefore $L_{ij}^{(k-1)} = p_j^{(k-1)}$. From equations (7) and (8), $p_j^{(k)} = p_i^{(k-1)} + 1 + \frac{\phi_{ij}(1+\phi_{ij})}{2}$. Since $L_{ij}^{(k)} = p_j^{(k)}$, we finally have that
$$L_{ij}^{(k)} - L_{ij}^{(k-1)} = 1 \leq 2. \tag{15}$$
- **Link** (i,j) **was not activated in iteration** $k-1$ **and** $p_i^{(k-1)} > p_j^{(k-1)}$: In this case $L_{ij}^{(k-1)} = p_i^{(k-1)}$. Also from equations (7) and (8) we have that $L_{ij}^{(k)} = p_j^{(k)} = p_i^{(k-1)} + 1 + \frac{\phi_{ij}(1+\phi_{ij})}{2}$. Therefore,
$$L_{ij}^{(k)} - L_{ij}^{(k-1)} = 1 + \frac{\phi_{ij}(1+\phi_{ij})}{2} \leq 2. \tag{16}$$
- **Link** (i,j) **was not activated in iteration** $k-1$ **and** $p_j^{(k-1)} \geq p_i^{(k-1)}$: In this case $L_{ij}^{(k-1)} = p_j^{(k-1)}$. Application of equations (7) and (8) yields $L_{ij}^{(k)} = p_j^{(k)} = p_j^{(k-1)} + 2$ and then:
$$L_{ij}^{(k)} - L_{ij}^{(k-1)} = 2 \leq 2. \tag{17}$$

For all cases $L_{ij}^{(k)} - L_{ij}^{(k-1)} \leq 2$ and the proof is complete.

Lemma 2: The following property holds for the forward progress $f(k)$ for every iteration k:
$$0 \leq f(k) - f(k-1) \leq 2, \; \forall k = 1, 2.., \tilde{T} \tag{18}$$

Proof We use contradiction. Suppose there is an iteration k for which $f(k) - f(k-1) > 2$. Since $f(k)$ is strictly greater than $f(k-1)$ the increase in the forward progress was contributed by at least one link $l = (i,j)$ in the link set $A_{\pi(k)}$ that was activated during this iteration. This means that $L_{ij}^{(k)} = f(k)$. From Lemma 1 it holds that:
$$\begin{aligned} L_{ij}^{(k-1)} &\geq L_{ij}^{(k)} - 2 \Leftrightarrow \\ L_{ij}^{(k-1)} &\geq f(k) - 2 \end{aligned} \tag{19}$$
and from the hypothesis we have that $f(k-1) < f(k) - 2$. Therefore it must be that $L_{ij}^{(k-1)} > f(k-1)$. We arrive at a contradiction since by the definition of these quantities this implies that $max\{p_i^{(k-1)}, p_j^{(k-1)}\} > \max_{n \in N}\{p_n^{(k-1)}\}$.

Proof of Theorem 2

Starting from Lemma 2 we have that:
$$\begin{aligned} \sum_{k=1}^{\tilde{T}}(f(k) - f(k-1)) &\leq \sum_{k=1}^{\tilde{T}}(2) \overset{f(0)=0}{\Longleftrightarrow} \\ f(\tilde{T}) &\leq 2\tilde{T} \overset{T^{(\pi)} \triangleq f(\tilde{T})}{\Longleftrightarrow} \\ T^{(\pi)} &\leq 2\tilde{T} \end{aligned}$$

The proof is complete.

Proof of Corrolary on scatternet feasibility: From Theorem 2, for any π:
$$\begin{aligned} T^{(\pi)}(\boldsymbol{\tau}) &\leq 2\tilde{T}(\boldsymbol{\tau}) && (20) \\ &\leq 2(\lfloor T_{system}/2 \rfloor) && (21) \\ &\leq T_{system}. && (22) \end{aligned}$$

Theorem 1 states that $T^{(\pi)}(\boldsymbol{\tau})$ is the minimum period that can be generated by link activation ordering π. Since the minimum period is less than or equal to the system period, the allocation $\boldsymbol{\tau}$ is feasible.

APPENDIX B: Algorithm pseudocodes

Procedure EQUIVALENT
```
input          : $G(N,E), \boldsymbol{\Phi} = [\phi_{ij}]$, $\tilde{\boldsymbol{S}}^{(\boldsymbol{\pi})} = [A_{\pi(k)}]$, $\boldsymbol{\pi}$, $\tilde{T}$
output         : $\boldsymbol{S}_{eq} = [\boldsymbol{S}_n]$, $n \in N$: The asynchronous equivalent schedule of $\tilde{\boldsymbol{S}}^{(\boldsymbol{\pi})}$
                 $T_{eq}$: The period of $\boldsymbol{S}_{eq}$.
local          : $\boldsymbol{p} = [p_n^{(k)}]$, $\boldsymbol{f} = [f(k)]$, $k$
Initialization:  $f(0) = 0, p_n^0 = 0, \forall n \in N$;
begin
1   for $k = 1$ to $\tilde{T}$ do
        AddLinkActivationSet($G, \boldsymbol{\Phi}, k, A_{\pi(k)}, \boldsymbol{p}, f(k), \boldsymbol{S}_{eq}$);
    end
    $T_{eq} = \boldsymbol{f}(\tilde{T})$ ;
    AddLinkActivationSet($G$, $\boldsymbol{\Phi}$, $\tilde{T}+1$, $A_{\pi(1)}$, $\boldsymbol{p}$, $f(\tilde{T}+1)$, $\boldsymbol{S}_{eq}$);
end
```

Function `AddLinkActivationSet`

```
Add a set of links to the asynchronous schedule $\boldsymbol{S}$
input  : $G(N,E)$, $\boldsymbol{\Phi}$, $k$, $LINKSET$, $\boldsymbol{p}^{(k-1)}$, $\boldsymbol{S}$
output : $\boldsymbol{p}^{(k)}$, $f(k)$, $\boldsymbol{S}$
local  : $ACTIVATED\_NODES\_SET = \{\ \}$
begin
1   for every link $l$ in $LINKSET$ do
        $i = l.master,\ j = l.slave$ ;
        Add $i$ and $j$ in $ACTIVATED\_NODES\_SET$ ;
        if ($p_i^{(k-1)} == 0$ AND $p_j^{(k-1)} == 0$) then
            /*This is the first activation for both nodes*/;
            $p_i^{(k)} = p_i^{(k-1)} + 1,\ \ \boldsymbol{S}_i(p_i^{(k)}) = l$ ;
            $p_j^{(k)} = p_i^{(k)} + \frac{\phi_{ij}(1+\phi_{ij})}{2},\ \ \boldsymbol{S}_j(p_j^{(k)}) = l$ ;
        end
        if ($\boldsymbol{S}_i(p_i^{(k-1)}) == l$ AND $\boldsymbol{S}_j(p_j^{(k-1)}) == l$) then
            /*Case A*/ ;
            $p_i^{(k)} = p_i^{(k-1)} + 1$;
        end
        else
            if ($p_i^{(k-1)} > p_j^{(k-1)}$) then
                /*Case B*/ ;
                $p_i^{(k)} = p_i^{(k-1)} + 1$;
            end
            else
                /*Case C*/ ;
                $p_i^{(k)} = p_j^{(k-1)} + \frac{\phi_{ij}^2 - \phi_{ij} + 2}{2}$ ;
            end
        end
        $\boldsymbol{S}_i(p_i^{(k)}) = l$ ;
        for any unassigned $t \in p_i^{(k-1)} + 1, ..., p_i^{(k)} - 1$ do
            $\boldsymbol{S}_i(t) = idle$ ;
        end
        $p_j^{(k)} = p_i^{(k)} + \frac{\phi_{ij}(1+\phi_{ij})}{2}$ ;
        $\boldsymbol{S}_j(p_j^{(k)}) = l$ ;
        for any unassigned $t \in p_j^{(k-1)} + 1, ..., p_j^{(k)} - 1$ do
            $\boldsymbol{S}_j(t) = l$ ;
        end
    end
    for every $n$ not in $ACTIVATED\_NODES\_SET$
    do
        $p_n^{(k)} = p_n^{(k-1)}$ ;
    end
    $f(k) = \max_{n \in N}\{p_n^{(k)}\}$;
end
```

Function `GetForwardProgress`

```
Compute forward progress on $\boldsymbol{S}$ due to $LINKSET$
input : $\boldsymbol{\Phi}$, $k$, $\boldsymbol{p}^{(k-1)}$, $LINKSET$, $\boldsymbol{S}$
local : $f$, $\boldsymbol{p} = [p_n],\ n = 1, .., N$ ,
        $ACTIVATED\_NODES\_SET = \{\ \}$
begin
1   for every link $l$ in $LINKSET$ do
        $i = l.master,\ j = l.slave$ ;
        Add $i$ and $j$ in $ACTIVATED\_NODES\_SET$ ;
        if ($p_i^{(k-1)} == 0$ AND $p_j^{(k-1)} == 0$) then
            $p_i = p_i^{(k-1)} + 1,\ \ p_j = p_i + \frac{\phi_{ij}(1+\phi_{ij})}{2}$;
        end
        if ($\boldsymbol{S}_i(p_i^{(k-1)}) == l$ AND $\boldsymbol{S}_j(p_j^{(k-1)}) == l$) then
            $p_i = p_i^{(k-1)} + 1$;
        end
        else
            if ($p_i^{(k-1)} > p_j^{(k-1)}$) then
                $p_i = p_i^{(k-1)} + 1$;
            end
            else
                $p_i = p_j^{(k-1)} + \frac{\phi_{ij}^2 - \phi_{ij} + 2}{2}$ ;
            end
        end
        $p_j^{(k)} = p_i^{(k)} + \frac{\phi_{ij}(1+\phi_{ij})}{2}$ ;
    end
    for every $n$ not in $ACTIVATED\_NODES\_SET$
    do
        $p_n = p_n^{(k-1)}$ ;
    end
    $f = \max_{n \in N}\{p_n\}$;
    return $f$;
end
```

Function `FindHperm`

```
Phase I of MIN_PROGRESS that finds permutation $\boldsymbol{\pi}_h$
input  : $\boldsymbol{\Phi}$, $\tilde{\boldsymbol{S}} = \{M^\alpha, \lambda^\alpha : \alpha = 1, .., |M(\tilde{\boldsymbol{S}})|\}$, $\tilde{T}$
output : $\boldsymbol{\pi}_h$: $\boldsymbol{\pi}_h(k)$ contains the activation set index
         selected at iteration $k$
local  : $\boldsymbol{U}$, $currmin$, $f$, $\boldsymbol{p} = [p_n^{(k)}]$
         $\boldsymbol{S}$ : $dummy\ asynchronous\ schedule$
Initialization: $\boldsymbol{U} = \{M^1, .., M^{|M(\tilde{\boldsymbol{S}})|}\}$ ;
begin
1   for $k = 1$ to $|M(\tilde{\boldsymbol{S}})|$ do
        $currmin = 3|M(\tilde{\boldsymbol{S}})|$ ;
        for every set $M^g \in \boldsymbol{U}$ do
            f=GetForwardProgress($\boldsymbol{\Phi}$, $k$, $\boldsymbol{p}^{(k-1)}$, $M^g$, $\boldsymbol{S}$);
            if ($currmin < f$) then
                $\boldsymbol{\pi}_h(k) = g$;
                currmin = f ;
            end
        end
        AddLinkActivationSet($G$, $\boldsymbol{\Phi}$, $k$, $M^{\boldsymbol{\pi}_h(k)}$, $\boldsymbol{p}$, $f$, $\boldsymbol{S}$);
        $\boldsymbol{U} = \boldsymbol{U} - \{M^{\boldsymbol{\pi}_h(k)}\}$ ;
    end
end
```

Procedure `MIN_PROGRESS`

```
input  : $G(N, E)$, $\boldsymbol{\Phi}$, $\tilde{\boldsymbol{S}} = \{M^\alpha, \lambda^\alpha\}$, $\tilde{T}$
output : $\boldsymbol{S}$: The asynchronous schedule computed by
         the heuristic
local  : $\boldsymbol{\pi}_h$: permutation of the activation sets $M^\alpha$
begin
1   /*Phase I*/ ;
    FindHperm($\boldsymbol{\Phi}$, $\tilde{\boldsymbol{S}}$, $\tilde{T}$, $\boldsymbol{\pi}_h$);
    Form $\tilde{\boldsymbol{S}}^{(\boldsymbol{\pi}_h)}$ from $\tilde{\boldsymbol{S}}$ using $\boldsymbol{\pi}_h$ for the ordering
    of sets $M^\alpha$ and activating the $\lambda^\alpha$ instances of
    each set $M^\alpha$ in consecutive slots.;
    /*Phase II*/ ;
    EQUIVALENT($G$, $\boldsymbol{\Phi}$, $\tilde{\boldsymbol{S}}^{(\boldsymbol{\pi}_h)}$, $\boldsymbol{\pi}_h$, $\boldsymbol{S}$);
end
```