Analysis of Bandwidth Allocation Algorithms for Bluetooth Wireless Personal Area Networks

Randeep Bhatia[1], Adrian Segall[2] and Gil Zussman[2] ⋆

[1] Bell Labs, Lucent Technologies, Murray Hill, NJ 07974, USA
randeep@research.bell-labs.com

[2] Department of Electrical Engineering, Technion, Haifa 32000, Israel
{segall@ee, gilz@tx}.technion.ac.il

Abstract. A major issue in the design and operation of ad hoc networks is sharing the common spectrum among links in the same geographic area. Bandwidth allocation, to optimize the performance of networks in which each station can converse with at most a single neighbor at a time, has been recently studied in the context of Bluetooth networks. There, centralized and distributed, capacity assignment heuristics were developed, with applicability to a variety of ad hoc networks. Yet, no guarantees on the performance of these heuristics have been provided. In this paper we present our analytic results regarding these heuristics. Specifically, we show that they are β-approximation ($\beta < 2$) algorithms. Moreover, we show that even though the distributed and centralized algorithms allocate capacity in a different manner, both algorithms converge to the same results. Finally, we present numerical results that demonstrate the performance of the algorithms.

1 Introduction

In the last four decades, much attention has been given to the research and development of bandwidth allocation and scheduling schemes for wired and wireless networks [3],[9]. The bandwidth allocation problem in wireless ad hoc networks significantly differs from the problem in static communication networks. For instance, one of the major problems in the design and operation of ad hoc networks is sharing the common spectrum among links in the same geographic area [9]. In this paper, we focus on bandwidth allocation in networks in which *each station can converse with at most a single neighbor at a time*. This problem has been recently studied mainly in the context of Bluetooth Personal Area Networks [2],[10],[11],[12].

Bluetooth enables portable mobile devices to connect and communicate wirelessly via short-range ad-hoc networks [6]. Since the radio link is based on frequency-hop spread spectrum, multiple channels (frequency hopping sequences) can co-exist in the same wide band without interfering with each other. Two or more units sharing the same channel form a *piconet*, where one unit acts as a *master* controlling the communication in the piconet and the others act as *slaves*. Bluetooth uses a slotted scheme where the only allowed communication is between a master and a slave and the master-to-slave and slave-to-master transmissions happen in alternate slots. Connected piconets in the same geographic area form a *scatternet* (see for example Fig. 1). In a scatternet, a unit can participate in two or more piconets, on a time-sharing basis, and even change its role when moving from one piconet to another (we refer to such a unit as a *bridge*).

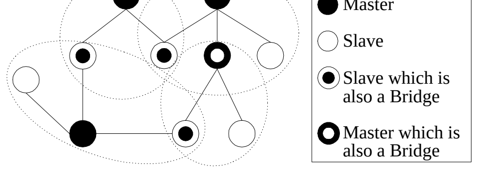

Figure 1. An example of a Bluetooth scatternet composed of 4 piconets.

⋆ This research (No. 148/03) was supported by THE ISRAEL SCIENCE FOUNDATION, by a grant from the Ministry of Science, Israel, and by Intel.

Efficient scatternet operation requires determining the link capacities that should be allocated in each piconet, such that the network performance is optimized. The need to find a feasible capacity allocation has been identified by Baatz et al. [2]. In [12] the capacity assignment problem has been formulated as a problem of minimizing a convex function over a convex set contained in the matching polytope (similar formulations appear in [1] and [11]) and optimal as well as heuristic algorithms for its solution have been proposed. Finally, [10] and [11] have studied the problem of maxmin fair allocation of bandwidth in networks which have similar characteristics to Bluetooth scatternets.

The work in [12] presents distributed and centralized heuristics for the solution of the capacity assignment problem, but does not include a detailed analysis of the performance of the algorithms. In this paper, we analyze the performance of these heuristics and show that although they allocate capacity in a different order, the distributed algorithm converges to the same results as the centralized one. Moreover, we show that the heuristic algorithms are actually β-approximation ($\beta < 2$) algorithms for the solution of the capacity assignment problem. We also present an important property regarding the tightness of the upper bound on their performance. Finally, numerical results are presented.

Since analysis of algorithms tailored for Bluetooth scatternets has been done mostly via simulation, an analytical approach that provides rigorous bounds on the performance is of great importance. As argued by Sarkar and Tassiulas [10],[11], although the algorithms have been developed in the context of Bluetooth scatternets, they can be applied to any ad hoc network in which a node transmits to a single neighbor at a time, and in which multiple transmissions can take place as long as they do not share a common node.

This paper is organized as follows. Section 2 presents the model and the formulation of the problem. The heuristic algorithms presented in [12] are briefly reviewed in Section 3. In Section 4, we show that the distributed and centralized algorithms converge to the same results. In Section 5, we show that the heuristic algorithms are β-approximation ($\beta < 2$) algorithms. Section 6 presents numerical results and Section 7 summarizes the results. Due to space constraints, some of the proofs and numerical results are omitted (they can be found in [5]).

2 Model and Problem Formulation

We model the scatternet by an undirected graph $G = (N, L)$, where N denotes the collection of *nodes* $\{1, 2, \ldots, n\}$. Each of the nodes could be a master, a slave, or a bridge. L denotes the set of *bi-directional links* where the link connecting nodes i and j is denoted by (i, j). We denote by $Z(i)$ the neighbors of node i. Scatternet graphs can be bipartite graphs or nonbipartite graphs [4]. In [4] and [12] it is shown that scatternet topologies which are nonbipartite may result in poor bandwidth utilization. Therefore, in this paper we focus only on bipartite scatternet graphs.

Due to the tight coupling of the uplink and downlink in Bluetooth piconets[1], we concentrate on the total bi-directional link capacity. Hence, we assume that the average packet delay on a link is a function of the total link flow and of the total link capacity. Let F_{ij} be the average bi-directional flow on link (i, j) and let C_{ij} be the capacity of link (i, j) (the units of F and C are bits/second). We assume that the average bi-directional flow is positive on every link ($F_{ij} > 0 \ \forall (i, j) \in L$). We define f_{ij} as the ratio between F_{ij} and the maximal possible flow on a Bluetooth link when using a given type of packets[2]. We also define c_{ij} as the ratio between C_{ij} and the maximal possible capacity of a link. It is obvious that $0 < f_{ij} \leq 1$ and that $0 < c_{ij} \leq 1$. We shall refer to f_{ij} as the *flow on link* (i, j) and to c_{ij} as the *capacity of link* (i, j).

The objective of the capacity assignment algorithms, analyzed in this paper, is to minimize the average delay in the scatternet. Since the total traffic in the network is independent of the capacity assignment procedure, we can minimize the average delay by minimizing the total delay. We use a delay function based on *Kleinrock's independence approximation*[3] [7]. Accordingly, the problem of *scatternet capacity assignment in bipartite graphs* (SCAB) is formulated as follows [12].

[1] A slave is allowed to start transmission, only after a master had addressed it in the preceding slot.

[2] For example, currently the maximal flow on a symmetrical link is 867.8 Kbits/second.

[3] Analytic results regarding the delay function are available only for simple scheduling regimes and simple topologies (e.g. [8],[13]). However, Kleinrock's independence approximation has been shown to provide a relatively good estimation for the delay in networks involving Poisson stream arrivals. Therefore, it is used for the evaluation of *approximation* capacity assignment algorithms.

Problem SCAB

Given: Topology of a bipartite graph and flows (f_{ij}).
Objective: Find capacities (c_{ij}) such that the average delay is minimized:

$$\text{minimize} \sum_{(i,j)\in L} \frac{f_{ij}}{c_{ij} - f_{ij}} \tag{1}$$

$$\text{subject to: } c_{ij} > f_{ij} \;\; \forall\,(i,j) \in L \tag{2}$$

$$\sum_{j\in Z(i)} c_{ij} \leq 1 \;\; \forall\, i \in N\ . \tag{3}$$

The second set of constraints (3) reflects the fact that the total capacity of the links connected to a node cannot exceed the maximal link capacity. Constraints similar to (3) appear in problems formulated in [1],[10], and [11]. We note that henceforth we will assume that there is a feasible solution to Problem SCAB.

The formulation of the problem is based on the assumption that the flow rates are given by higher layer protocols based on the traffic statistics. We believe that algorithms such as the ones described in [12] are expected to provide insight into the development of good capacity allocation schemes.

3 Approximation Algorithms

A *capacity assignment algorithm* has to obtain a solution to Problem SCAB (i.e. to determine what portion of the slots should be allocated to each master-slave link). In this section we briefly review the *centralized* and *distributed* approximation algorithms for bipartite scatternets, presented in [12]. We shall refer to these algorithms as the *heuristic centralized/distributed scatternet capacity assignment* algorithms (Algorithm HCSCA/HDSCA respectively). First, we define the *slack capacity* of a node as follows:

Definition 1. *The* slack capacity of node i *is the maximal capacity which can be added to links connected to the node. It is denoted by s_i and is given by $s_i = 1 - \sum_{j\in Z(i)} c_{ij}$.*

In both algorithms, all link capacities are initially equal to the flows on the links ($c_{ij} = f_{ij},\ \forall\,(i,j) \in L$). The algorithms select a node and allocate the slack capacity to some of the links connected to it. Then, another node is selected, capacity is allocated and so on. In both algorithms the nodes are selected according to their delay derivative, which is defined below.

Definition 2. *The* delay derivative of node i *is denoted by d_i and is given by:*

$$d_i = \frac{\sum\limits_{m:\, m\in Z(i),\, c_{im}=f_{im}} \sqrt{f_{im}}}{s_i}\ . \tag{4}$$

Once a node k is selected, *the slack capacity of this node is allocated to those adjacent links, whose capacities have not yet been assigned.* The slack capacity is assigned according to the square root assignment:

$$c_{kj} = f_{kj} + \frac{s_k\sqrt{f_{kj}}}{\sum\limits_{m:\, m\in Z(k),\, c_{km}=f_{km}} \sqrt{f_{km}}} \;\; \forall\, j \in Z(k),\ c_{kj} = f_{kj}\ . \tag{5}$$

Capacity is allocated to a link only once. Hence, we define a *fully allocated node* as a node whose all adjacent link capacities have been assigned[4]. Accordingly, a *non-fully allocated node* is a node which has at least one adjacent link whose capacity has not been assigned.

3.1 Centralized Algorithm (Algorithm HCSCA)

Node k, whose link capacities are next to be assigned, is selected from the non-fully allocated nodes. The delay derivatives d_i of these nodes are computed and the node with the largest derivative is selected. Algorithm HCSCA, which is based on this methodology, is described in Fig. 2. The input is the topology and the flows (f_{ij}), and the output is the link capacities: c_{ij}. It can be seen that the complexity of the algorithm is $O(n^2)$, which is about the complexity of a single iteration in the optimal algorithm, presented in [12]. Moreover, in [12] it has been shown that the capacity values obtained by the algorithms are always feasible.

[4] A fully allocated node does not necessarily utilize its full capacity.

```
1 set $c_{ij} = f_{ij} \ \forall (i,j) \in L$
2 set $k = \underset{i \in N \bigcap i \text{ non-fully allocated}}{\arg\max} d_i$
3 set $c_{kj} = f_{kj} + \sqrt{f_{kj}}/d_k \,\forall j : j \in Z(k),\ c_{kj} = f_{kj}$
4 if there exists $(i,j) \in L$ such that $c_{ij} = f_{ij}$
5     then go to 2
6     else stop
```

Figure 2. Algorithm HCSCA for obtaining an approximate solution to Problem SCAB.

3.2 Distributed Algorithm (Algorithm HDSCA)

In the distributed algorithm, a token is passed by the nodes and only the node that holds the token is allowed to allocate capacity. The algorithm is initiated by an arbitrary node that creates the token. Once a node receives the token, it can either allocate its slack capacity or decide to send the token to a neighbor. The assignment of slack capacity is the same as in the centralized algorithm. However, the selection of the node which holds the token and the decision whether it should allocate capacity or transfer the token to a neighbor is different.

Each node keeps a stack, referred to as the *parents stack*, that contains the identities of neighbors from which it had previously received the token. Each node also maintains a list of non-fully allocated neighbors. We define two possible states for the node holding the token:

– *Allocation State* – A non-fully allocated node enters this state when it receives the token. At this time the node pushes the identity of the neighbor, that sent it the token, to the parents stack. The neighbor is referred to as one of the node's parents. The node decides to either transfer the token to a neighbor or allocate capacity and then move to the token transfer state.
– *Token Transfer State* – A node enters this state after it allocates capacity or when it receives the token from a neighbor that popped its details from the stack. In this state, one of the non-fully allocated neighbors will receive the token. If all the neighbors are fully allocated, the token will be returned to the first neighbor in the stack and this neighbor will be popped from the stack. The algorithm halts when all the neighbors are fully allocated and the stack is empty (it always terminates at the initiating node).

Fig. 3 presents the pseudocode of the procedure executed by a node in the allocation state. Unlike the centralized algorithm in which a node allocates capacity, if its d_k is the largest in the network, in the distributed algorithm a node allocates capacity, if it holds the token and its d_k is larger than the d_ks of its neighbors.

```
1 push into the parents stack the details of the node which sent the token
2 find the node with the largest $d_k$ among the non-fully allocated neighbors and yourself
3 if it is a neighbor
4   then send the token to this neighbor
5   else
6     allocate capacity according to (5)
7     update the neighbors
8     change the state to token transfer state
```

Figure 3. Algorithm HDSCA – the procedure executed by a node in the allocation state.

Fig. 4 describes the pseudocode of the procedure executed by a node in the token transfer state. A node enters this state due to two possible events: capacity allocation by the node or receipt of the token from a neighbor that popped its details from the stack. In this state, it can either send the token to its "best" neighbor or return it to one of its parents.

```
1 **find** the node with the largest $d_k$ among the non-fully allocated neighbors
2 **if** such a node exists
3   **then** send the token to this neighbor
4   **else**
5     **if** the stack is empty
6       **then** halt
7       **else**
8         **pop** the first node from the parents stack
9         **send** the token to that parent
```

Figure 4. Algorithm HDSCA – the procedure executed by a node in the token transfer state.

4 Convergence of the Distributed Algorithm

Due to the differences between the algorithms, the centralized algorithm (Algorithm HCSCA) and the distributed algorithm (Algorithm HDSCA) normally allocate capacity in different order. However, in this section we will show that *the two algorithms always converge to the same results.* First, we show that the distributed algorithm halts after all the link capacities have been allocated. Then, we show that these link capacities are the same as the link capacities allocated by the centralized algorithm.

The proof that the distributed algorithm halts after all the link capacities have been allocated is based on the fact that the token either does not traverse a link or traverses it in both directions. Accordingly, since the token cannot be returned to a parent before all the neighbors are fully allocated, the algorithm cannot halt before all the link capacities have been allocated. The formal proof is based on the following lemmas, whose proofs are omitted for lack of space (they can be found in [5]).

Lemma 1. *In Algorithm HDSCA, when a node i enters the token transfer state, it is fully allocated ($c_{ij} \neq f_{ij}$ $\forall j \in Z_i$).*

Lemma 2. *When Algorithm HDSCA halts, every node that has been in the allocation state has also been in the token transfer state.*

The proof of Lemma 2 implies that every node that has been in the allocation state has also executed Step 4 described in Fig. 4 (i.e. checked the status of the stack and then either halted or returned the token to a parent). Using the above lemmas we shall now prove the following proposition.

Proposition 1. *When Algorithm HDSCA halts: $c_{ij} \neq f_{ij}$ $\forall (i,j) \in L$.*

Proof. Assume that when the algorithm halts, there is a link (i,j) for which $c_{ij} \neq f_{ij}$ does not hold (either c_{ij} is not defined or $c_{ij} = f_{ij}$). According to Lemma 1, nodes i and j do not enter the token transfer state during the execution of the algorithm. Consequently, according to Lemma 2, node i and j do not enter the allocation state during the execution. Since node j does not enter the allocation state, none of its neighbors ever executes Step 4 described in Fig. 4 (since before its execution they would send the token to node j which would enter the allocation state). Following the argument used in the proof of Lemma 2, due to the fact that the protocol halts, every node that has been in the allocation state has also executed Step 4 described in Fig. 4. Thus, none of the neighbors of j enters the allocation state and the capacities of the links connecting them to j are not assigned.

Using a similar argument, it can be shown that none of the link capacities of the neighbors of j are assigned. Consequently, no node enters the allocation state and no link capacity is assigned. This is a contradiction to the fact that the algorithm halts. □

We now need to show that the capacity allocated by Algorithm HDSCA is equal to the capacity allocated by Algorithm HCSCA. Thus, we first derive a property of the delay derivative of a node in algorithms HDSCA and HCSCA (this property will also be used in Section 5). Then, we prove by induction that the capacities allocated by the two algorithms are identical.

Lemma 3. *When a node i allocates capacity the delay derivatives ($d_j s$) of its non-fully allocated neighbors $j \in Z(i)$ do not increase.*

Proof. Let k be a non-fully allocated neighbor of i. Let d_k^- and d_k^+ be its delay derivatives just before and just after (respectively) i allocates capacity. Denote by $M(k)$ the set of k's neighbors ($m \in Z(k)$) such that $c_{km} = f_{km}$ just before i allocates capacity. According to (4) and (5), following the allocation:

$$d_k^+ = \frac{\sum\limits_{m \in M(k)} \sqrt{f_{km}} - \sqrt{f_{ik}}}{s_k - (c_{ik} - f_{ik})} = \frac{\sum\limits_{m \in M(k)} \sqrt{f_{km}} - \sqrt{f_{ik}}}{(\sum\limits_{m \in M(k)} \sqrt{f_{km}})/d_k^- - \sqrt{f_{ik}}/d_i^-} . \qquad (6)$$

Since i allocates the capacity, just before the allocation $d_i^- \geq d_k^-$. Accordingly,

$$d_k^+ \leq \frac{\sum\limits_{m \in M(k)} \sqrt{f_{km}} - \sqrt{f_{ik}}}{(\sum\limits_{m \in M(k)} \sqrt{f_{km}})/d_k^- - \sqrt{f_{ik}}/d_k^-} = d_k^- . \qquad (7)$$

□

Theorem 1. *The capacities (c_{ij}) obtained by Algorithm HDSCA are identical to the capacities obtained by Algorithm HCSCA.*

Proof. According to Proposition 1, Algorithm HDSCA halts only after all the link capacities have been allocated. Thus, we have to show that these link capacities are the same as the link capacities assigned by Algorithm HCSCA.

We need to show that when Algorithm HDSCA is executed, at any given time, the following properties hold:

1. For every non-fully allocated node j, the neighbors that allocate capacity after j in Algorithm HCSCA are non-fully allocated (i.e. some or all neighbors which allocate capacity before it in Algorithm HCSCA are fully allocated).
2. The values of the capacities that have already been allocated are the same as in Algorithm HCSCA.

In order to prove it, we assume that the above properties hold at time t^- and we show that if a node i allocates capacity at time t (immediately after time t^-), these properties continue to hold.

At time t, Algorithm HDSCA selects a non-fully allocated node i with a delay derivative (d_i) higher than the delay derivatives of its non-fully allocated neighbors and allocates capacity (Step 6 in Fig. 3). We wish to show that when i is selected, *all* (and not *some*) the neighbors which allocate capacity before it in Algorithm HCSCA are fully allocated.

Denote by $d_i(t)$ the delay derivative of node i at time t. Denote the set of the non-fully allocated neighbors of i at time t^- by $M(i)$ ($m \in M(i)$ if $m \in Z(i)$ and $c_{im} = f_{im}$ at time t^-). let t_m be the time in which a node m allocates capacity in Algorithm HCSCA. Due to the first property and due to Lemma 3, $d_i(t^-) \geq d_m(t^-) \geq d_m(t_m)$, $\forall m \in M(i)$. In order for $m \in M(i)$ to allocate capacity before i in Algorithm HCSCA, $d_m(t_m) \geq d_i(t_m)$ has to hold. However, in order for d_i to become smaller than d_m, one of the other non-fully allocated neighbors of i has to allocate capacity in Algorithm HCSCA before time t_m. This cannot happen, since their delay derivatives are lower than the delay derivative of i. Thus, the nodes in $M(i)$ allocate capacity after i.

Since at time t, *all* the neighbors of i which allocate capacity before it in Algorithm HCSCA are fully allocated, i allocates the same capacities as in Algorithm HCSCA. Thus, at time t^+ (immediately after time t) the second property holds. Moreover, since we have shown that in Algorithm HCSCA, i allocates capacity before the nodes in $M(i)$, at time t^+, the first property holds.

Finally, Since the properties 1 and 2 hold before the first node allocates capacity, they also hold after the last node allocates capacity, and therefore, the capacity values allocated by Algorithm HDSCA are identical to the values allocated by Algorithm HCSCA. □

5 Approximation Ratios

In this section we show that algorithms HCSCA and HDSCA are β-approximation ($\beta < 2$) algorithms for the solution of Problem SCAB. First, we present a new algorithm for the solution of Problem SCAB. Then, we prove that the new algorithm outperforms any 2-approximation algorithm. Finally, we prove that Algorithm HCSCA obtains results which are equal or better than the results obtained by the new algorithm. Since in Section 4 we have shown that algorithms HCSCA and HDSCA converge to the same solution, this implies that Algorithm HDSCA has the same property. We note that in this section we also present an interesting property, of the upper bound, on the performance of the algorithms.

Let $I(i)$ denote the set of links $e \in L$ that are incident on node i. By setting $\tau_e = c_e - f_e, e \in L$ in the non-linear program for the Problem SCAB we obtain the following equivalent non-linear program with variables $\tau_e, \forall e \in L$.

$$\begin{aligned} \text{minimize} \ & \sum_{e \in L} \frac{f_e}{\tau_e} \\ \text{subject to:} \ & \sum_{e \in I(i)} \tau_e \leq s_i \ \forall i \in N \\ & \tau_e > 0 \ \forall e \in L \ . \end{aligned} \tag{8}$$

We denote by τ_e^* the optimal solution to (8). Recall that according to Definition 1, initially (before the first phase of any algorithm) $s_i = 1 - \sum_{e \in I(i)} f_e$. As mentioned before, we assume that the problem has a feasible solution and therefore, s_i must be greater than zero for all $i \in N$.

Let us consider a set of $|N|$ non-linear programs, one for each $i \in N$:

$$\begin{aligned} \text{minimize} \ & \sum_{e \in I(i)} \frac{f_e}{\tau_e} \\ \text{subject to:} \ & \sum_{e \in I(i)} \tau_e \leq s_i \\ & \tau_e > 0 \ \forall e \in L \ . \end{aligned} \tag{9}$$

Note that (9) can be optimally solved in polynomial time for each i and has a unique optimal solution (i.e. the square root assignment [7, p. 20], see also (5)). We denote the optimal solution of (9) for node i by τ_e^i and define the corresponding optimal value OPT_i of the objective function as

$$OPT_i = \sum_{e \in I(i)} \frac{f_e}{\tau_e^i} = \frac{\left(\sum\limits_{e \in I(i)} \sqrt{f_e}\right)^2}{1 - \sum\limits_{e \in I(i)} f_e}. \tag{10}$$

We now present a simple algorithm, referred to as *Algorithm ASCA* (Approximate Scatternet Capacity Assignment), for obtaining an approximate solution to the Problem SCAB. We denote by $\hat{\tau_{ij}}$ the solution obtained by Algorithm ASCA. For every node i, the algorithm computes the optimal solution to (9) (i.e. $\tau_e^i, \forall e \in L$). Then, it sets for all $e = (i, j) \in L$:

$$\hat{\tau_e} = \min{(\tau_e^i, \tau_e^j)}. \tag{11}$$

We first show that $\hat{\tau_e}$ is a feasible solution to Problem SCAB (i.e. to (8)). It is easy to see that $\hat{\tau_e} > 0, \forall e \in L$ and by construction $\hat{\tau_{ij}} \leq \tau_{ij}^i$. Since τ_e^i is an optimal solution to (9), for node i we have $\sum_{e \in I(i)} \tau_e^i \leq s_i$ and hence $\sum_{e \in I(i)} \hat{\tau_e} \leq s_i$.

We now show that Algorithm ASCA is better than a 2-approximation algorithm for the problem SCAB. In order to prove it, we first present the following lemma.

Lemma 4 (Zussman and Segall, 2004 [12]). *If $d_j > d_i$, then $\tau_{ij}^j < \tau_{ij}^i$. If $d_j = d_i$, then $\tau_{ij}^j = \tau_{ij}^i$.*

Theorem 2. *Algorithm ASCA is a β-approximation algorithm ($\beta < 2$) for Problem SCAB.*

Proof. For any node $i \in N$, τ_e^* (the optimal solution to the non-linear program (8)) is a feasible solution to the non-linear program (9). Hence, since OPT_i is the optimal value of the objective function for (9):

$$\sum_{e \in I(i)} \frac{f_e}{\tau_e^*} \geq OPT_i \ \forall i \in N. \tag{12}$$

Adding up for all $i \in N$ and noting that the term for each edge appears exactly twice in the sum we have

$$2 \sum_{e \in L} \frac{f_e}{\tau_e^*} \geq \sum_{i \in N} OPT_i. \tag{13}$$

For each node i we define a set of incident edges $J(i) \subseteq I(i)$ as those edges $e = (i,j) \in L$ for which $\tau_e^i < \tau_e^j$, or in the case of a tie ($\tau_e^i = \tau_e^j$), the node index $i < j$. It is easy to see that $J(i)$ forms a partition of the set of edges L. Thus:

$$\sum_{e \in L} \frac{f_e}{\hat{\tau}_e} = \sum_{i \in N} \sum_{e \in J(i)} \frac{f_e}{\hat{\tau}_e} = \sum_{i \in N} \sum_{e \in J(i)} \frac{f_e}{\tau_e^i} \leq \sum_{i \in N} OPT_i,$$

where the last inequality follows from the definition of OPT_i in (10) and the fact that $J(i) \subseteq I(i)$. However, for a more tighter analysis we make the following observation. From Lemma 4 it follows that the node j with the smallest delay derivative d_j (in case of a tie, j is selected to be the node with the largest index) must have that $J(j) = \emptyset$. Thus, there is at least one node j with $J(j) = \emptyset$, and therefore since by definition $OPT_j > 0$ we have

$$\sum_{e \in L} \frac{f_e}{\hat{\tau}_e} \leq \sum_{i \in N, i \neq j} OPT_i < \sum_{i \in N} OPT_i$$

Combining this with (13)

$$\sum_{e \in L} \frac{f_e}{\hat{\tau}_e} < 2 \sum_{e \in L} \frac{f_e}{\tau_e^*},$$

thus showing that Algorithm ASCA is a β-approximation ($\beta < 2$) algorithm for the Problem SCAB. □

Algorithm HCSCA (described in Section 3.1) can be described as follows. Let ${\tau'}_e$ be the solution obtained by Algorithm HCSCA. At any phase, the algorithm starts out with a graph (initially set to the original graph G) with slack capacities s_i (initially set to the original graphs slack capacities). In this graph, it finds the node i with the maximal value of d_i, solves the non-linear program (9) for that node i optimally, and sets

$${\tau'}_e = \tau_e^i \ \ \forall e \in I(i).$$

It then decreases the slack capacities s_j for every node j, which is a neighbor of i, by the sum of ${\tau'}_e, e \in I(j)$ for all ${\tau'}_e$ that get set in this phase. This is done to reflect the capacity that has been already allocated. Any node i with $s_i = 0$ is removed from the graph. Also all the edges e that are assigned a value τ_e in this phase are removed from the graph. The new graph and the new slack capacities become input for the next phase. The algorithm terminates when no more edges are left in the graph.

Let L^p be the set of edges such that ${\tau'}_e,\ e \in L^p$ is set in phase p. Let ${\tau_e^i}^p$ be the optimal solution obtained for the non-linear program (9) for node i for the graph G^p used by Algorithm HCSCA in phase p along with the slack capacities s_i^p. Let the node delay derivatives in phase p be d_i^p. Note that $G^1 = G$, $s_i^1 = s_i, \forall i \in N$, $d_i^1 = d_i, \forall i \in N$, and ${\tau_e^i}^1 = \tau_e^i, \forall e \in I(i)$. Let $I^p(i)$ be the set of edges incident on node i in G^p. Due to Lemma 4 and the fact that at each phase p, the node with the largest d_i^p is selected, at the end of phase p we have

$${\tau'}_e = \min\left({\tau_e^i}^p, {\tau_e^j}^p\right) \ \ \forall e \in I^p(i). \tag{14}$$

We now show that for each edge the delay obtained by the algorithm HCSCA is at most the delay obtained by the Algorithm ASCA. In order to prove it, we first prove the following lemma.

Lemma 5. *${\tau_e^i}^{p+1} \geq {\tau_e^i}^p$ for all nodes i and edges $e \in I^{p+1}(i)$.*

Proof. According to (4) and (5), and the definition of ${\tau_e^i}^p$, for all edges $e \in I^p(i)$ we have ${\tau_e^i}^p = \sqrt{f_e}/d_i^p$. According to Lemma 3, for all nodes i in G^{p+1} we have $d_i^{p+1} \leq d_i^p$. Combining the above observations completes the proof. □

Proposition 2. *Algorithm HCSCA is a β-approximation algorithm ($\beta < 2$) for the Problem SCAB.*

Proof. We show that ${\tau'}_e \geq \hat{\tau}_e, \forall e \in L$ [5], thus showing that the value of the objective function of the non-linear program (8) for the solution ${\tau'}_e$ is at most the value of the objective function of (8) for the solution obtained by a β-approximation algorithm ($\beta < 2$). It follows from Lemma 5 that for any node i and link $e \in I^p(i)$:

$${\tau_e^i}^p \geq {\tau_e^i}^{p-1} \ldots \geq {\tau_e^i}^1 = \tau_e^i.$$

Thus, ${\tau_e^i}^p \geq \tau_e^i$ and ${\tau_e^j}^p \geq \tau_e^j$ for a link $e = (i,j) \in L^p$. Hence, since for such a link e, we have ${\tau'}_e = \min({\tau_e^i}^p, {\tau_e^j}^p)$ (see (14)) and $\hat{\tau}_e = \min(\tau_e^i, \tau_e^j)$ (see (11)), we have $\forall e = (i,j) \in L^p$:

$${\tau'}_e = \min({\tau_e^i}^p, {\tau_e^j}^p) \geq \min(\tau_e^i, \tau_e^j) = \hat{\tau}_e.$$ □

[5] Recall that ${\tau'}_e$ and $\hat{\tau}_e$ are the solutions obtained by algorithms HCSCA and ASCA, respectively.

Finally, we show that any upper bound on the performance of Algorithm HCSCA which is based on the relationships between OPT_i and the optimal solution (i.e. based on (13)) *is not tight.*

Proposition 3. *Assume that there exists $\beta\,(1 < \beta < 2)$ such that*

$$\sum_{e\in L}\frac{f_e}{\tau'_e} \leq \frac{\beta}{2}\sum_{i\in N} OPT_i \leq \beta\sum_{e\in L}\frac{f_e}{\tau^*_e}, \tag{15}$$

thus implying that HCSCA is a β-approximation algorithm, then there is no tight example in which the heuristic solution (τ'_e), obtained by HCSCA is exactly β times more than the optimal solution. In other words there is no example for which

$$\sum_{e\in L}\frac{f_e}{\tau'_e} = \beta\sum_{e\in L}\frac{f_e}{\tau^*_e}. \tag{16}$$

Proof. Assume that an example in which (16) holds exists. This implies that all inequalities in (15) must hold with equalities. Specifically

$$\sum_{i\in N} OPT_i = 2\sum_{e\in L}\frac{f_e}{\tau^*_e}.$$

Therefore, for any given node i, (12) holds with equality (i.e. $OPT_i = \sum_{e\in I(i)} f_e/\tau^*_e$). Consequently, since the optimal solution for (9) is unique, we have for every link $e = (i,j)$: $\tau^i_e = \tau^*_e = \tau^j_e$. Thus, the solution obtained by Algorithm ASCA ($\hat{\tau}_e$) is equal to the optimal solution (since $\hat{\tau}_e = \min\{\tau^i_e, \tau^j_e\}$). Finally, in the proof of Proposition 2, we have shown that $\tau'_e \geq \hat{\tau}_e, \forall e \in L$. Thus

$$\sum_{e\in L}\frac{f_e}{\tau'_e} \leq \sum_{e\in L}\frac{f_e}{\tau^*_e},$$

which contradicts (16). □

6 Numerical Results

From the observations made in Section 5 it follows that:

$$\sum_{e\in L}\frac{f_e}{\tau^*_e} \leq \sum_{e\in L}\frac{f_e}{\tau'_e} \leq \sum_{e\in L}\frac{f_e}{\hat{\tau}_e} < \sum_{i\in N} OPT_i \leq 2\sum_{e\in L}\frac{f_e}{\tau^*_e}\,.$$

Namely: Optimal Solution $\leq$ Solution by HCSCA $\leq$ Solution by ASCA $<$ Upper Bound ($\sum_{i\in N} OPT_i$) $\leq$ 2$*$(Optimal Solution). In this section we present a numerical example that demonstrates the difference between the optimal results, the results obtained by the Algorithms HCSCA and HDSCA, and the results obtained by Algorithm ASCA. Additional examples can be found in [5].

Fig. 5(a) illustrates a scatternet with different values of flow represented in terms of the variable x. Fig. 5(b) presents the values of the optimal and approximate solutions as well as the upper bound for this scatternet for different values of x. It can be seen that for all flow values, the approximate solutions are very close to the optimal solution and that there is a relatively large difference between the upper bound ($\sum_{i\in N} OPT_i$) and the approximate solutions. We note that in all the cases we have checked, the ratio of the solution obtained by Algorithm HCSCA to the optimal solution was much lower than 2.

7 Conclusions and Future Study

This paper analyzes heuristic centralized and distributed capacity assignment algorithms and provides an upper bound on their performance. Those algorithms have been designed for Bluetooth scatternets but can be applied to any ad hoc network in which a node transmits to a single neighbor at a time, and in which multiple transmissions can take place as long as they do not share a common node.

We have defined a simple approximation algorithm and shown that the ratio between the results obtained by this algorithm and the optimal results is less than two. Then, we have shown that the heuristic algorithms presented in [12] obtain results which at the worst case are the same as the results obtained by the new algorithm, thus establishing that these algorithms are β-approximation ($\beta < 2$)

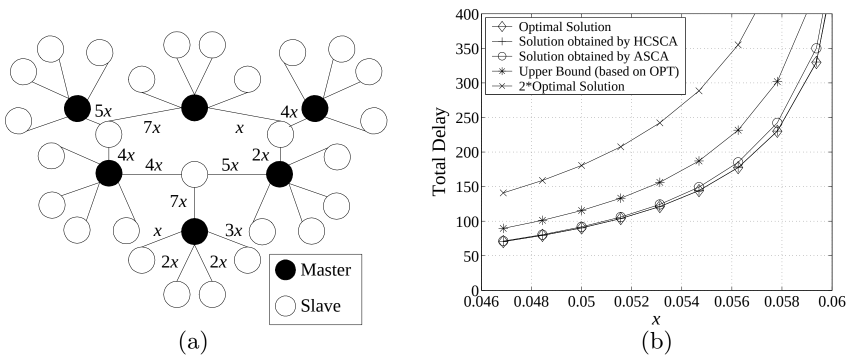

Figure 5. A scatternet: (a) flows expressed in terms of x (the flow values to the non-bridge slaves in every piconet are identical to the values in the lowest piconet), (b) upper bound, optimal solution, and approximate solutions.

algorithms for the capacity assignment problem. Moreover, we have shown that although the distributed and centralized algorithms allocate capacity in a different manner, both algorithms converge to the same results. Finally, we have presented numerical results and have compared the approximate solutions to the optimal solution and the upper bound.

There are still many open problems to deal with. For example, it seems that the ratio between the approximate and the optimal solutions is much lower than 2. However, proving this property requires further research. Moreover, future study will focus on improving the distributed algorithm and on investigating its performance in a dynamic topology. Finally, we note that a major future research direction is the development of bandwidth allocation methods that will be able to deal with various quality-of-service requirements and to interact with topology construction, scheduling, and routing protocols.

References

1. S. Baatz, C. Bieschke, M. Frank, C. Kühl, P. Martini, and C. Scholz, *"Building Efficient Bluetooth Scatternet Topologies from 1-Factors"*, Proc. IASTED WOC 2002, July 2002.
2. S. Baatz, M. Frank, C. Kühl, P. Martini and C. Scholz, *"Adaptive Scatternet Support for Bluetooth using Sniff Mode"*, Proc. IEEE LCN'01, Nov. 2001.
3. D. P. Bertsekas and R. Gallager, *Data Networks*, Prentice-Hall Inc., New Jersey, 1992.
4. P. Bhagwat and S. P. Rao, *"On the Characterization of Bluetooth Scatternet Topologies"*, Submitted for Publication, Available at http://www.winlab.rutgers.edu/~pravin/publications/papers/bt-top.ps, Feb. 2004.
5. R. Bhatia, A. Segall, and G. Zussman, *"Analysis of Bandwidth Allocation Algorithms for Bluetooth Wireless Personal Area Networks"*, CCIT Report No. 452, Dept. of Electrical Engineering, Technion, Available at http://www.comnet.technion.ac.il/~gilz/pub_files/ccit_452.pdf, Nov. 2003.
6. Bluetooth Special Interest Group, *Specification of the Bluetooth System – Version 1.2*, Nov. 2003.
7. L. Kleinrock, *Communication Nets: Stochastic Message Flow and Delay*, McGraw-Hill, New York, 1964.
8. D. Miorandi, C. Caimi, and A. Zanella, *"Performance Characterization of a Bluetooth Piconet with Multi-Slot Packets"*, Proc. WiOpt'03, Mar. 2003.
9. S. Ramanathan, *"A Unified Framework and Algorithm for Channel Assignment in Wireless Networks"*, ACM/Kluwer Wireless Networks, Vol. 5, No. 2, pp. 81-94 Mar. 1999.
10. S. Sarkar and L. Tassiulas, *"End-to-end bandwidth guarantees through fair local spectrum share in wireless ad-hoc networks"*, Proc. IEEE CDC'03, Dec. 2003.
11. L. Tassiulas and S. Sarkar, *"Maxmin Fair Scheduling in Wireless Networks"*, Proc. IEEE INFOCOM'02, June 2002.
12. G. Zussman and A. Segall, *"Capacity Assignment in Bluetooth Scatternets - Optimal and Heuristic Algorithms"*, ACM/Kluwer Mobile Networks and Applications (MONET), Vol. 9, No. 1, pp. 49-61, Feb. 2004.
13. G. Zussman, U. Yechiali, and A. Segall, *"Exact Probablistic Analysis of the Limited Scheduling Algorithm for Symetrical Bluetooth Piconets"*, Proc. IFIP-TC6 PWC'03, LNCS Vol. 2775 (eds: M. Conti et al.), Springer, Sep. 2003.