Decentralized Optimization of Dynamic Bluetooth Scatternets

Sewook Jung
Mario Gerla
Department of Computer Science
University of California, Los Angeles
{sewookj,gerla}@cs.ucla.edu

Csaba Kiss Kalló
Mauro Brunato
Dipartimento di Informatica e Telecomunicazioni
Università di Trento, Trento, Italy
{kkcsaba,brunato}@dit.unitn.it

Abstract

Previous work analytically showed that communication path length reduction is an efficient way for improving the performance of Bluetooth scatternets. Maintaining short communication paths is mainly important in dynamic scatternets with changing traffic flows, mobile nodes and in the presence of interference, when the network topology changes continuously. In this work we aim at demonstrating through simulations that in such dynamic scatternets by periodically reducing the path length (i.e. hop count) between the communicating nodes, the overall throughput supported by the network can be significantly increased and the available energy of nodes can be consumed more efficiently. For this purpose, we present a distributed technique for repeatedly re-configuring the scatternet topology such that to support the current traffic flows between all of the communicating peers with a small number of hops.

1 Introduction

Bluetooth is a short-range wireless network technology that supports ad hoc networking. In Bluetooth, a maximum of 8 active nodes form a star-shaped cluster, called *piconet*. The cluster head is called *master* while the other nodes are its *slaves*. Piconets interconnected through so-called *bridge* nodes form a *scatternet*. Bridges are nodes participating in more than one piconet on a time sharing basis. We call *slave&bridges* those nodes that have slave role in all of the piconets they participate in, while nodes having both, slave and master roles in different piconets are *master&bridges*.

Bluetooth data communication happens through Asynchronous Connectionless Links (ACL) using time slots of $625\mu s$. Data packets may use 1, 3 or 5 slots and they may be Forward Error Coded (FEC). FEC packets are DM1, DM3 and DM5, enabling the correction of single-bit errors in each codeword of 15 bits, while the non-error coded ones are DH1, DH3 and DH5 (with the digits indicating the number of slots used). The maximum useful payload of these packets is 136, 968 and 1816 bits for DM and 216, 1464 and 2712 bits for DH packets, respectively. Packets with bigger payloads can achieve higher throughput in error-free environments (i.e. with high link quality). However, if a single bit gets corrupted, the whole packet will have to be retransmitted. Therefore, when retransmissions occur often, smaller packets are more efficient. DM packets have smaller payloads than their DH counterparts, but their content is error checked, in contrast with DH packets.

The latest Bluetooth Specification 2.0 [4] introduces the concept of scatternet formation, but it does not define it in detail. In consequence, numerous scatternet formation algorithms were proposed in the literature. Some earlier protocols [8, 7] require the nodes to be all in range, which simplifies node discovery and piconet formation. Other approaches [11] form tree-shaped scatternets that simplify routing, but also transform the root node into a bottleneck and are not robust in the presence of mobility. Several later protocols [3, 10] form mesh-shaped scatternets without the above shortcomings and operate well in general scenarios. However, even if we had an optimal scatternet formation algorithm that produces optimal topologies, in a mobile scenario, some time after the network formation the scatternet topology would become suboptimal. Mobility, the dynamic traffic flows and similar factors transform the topology such that it can not support the current traffic flows with optimal paths.

In [6] we analytically show that the number of hops along all of the traffic flows in the scatternet is in close relation with the overall performance in terms of throughput and power consumption. As the path lengths become longer the scatternet throughput decreases and a higher amount of power is consumed. To counterweight this tendency, in [5] we developed a suite of heuristic algorithms based on local search techniques that is capable of dynamically adapting the scatternet topology to the current traffic flows. In that optimization work we aim at correcting the suboptimal traffic paths that are formed when nodes change their communication peers or migrate across the scatternet. Our algorithms update the topology of the scatternet making it possible for the routing algorithms to identify shorter paths between the communicating peers. This, in turn, results in higher aggregate throughput and reduced power consumption. In that work we devised an algorithm suite for reducing the hop count between all communication peers in the scatternet. In this paper we demonstrate through simulations based on those algorithms that in such dynamic scatternets by periodically reducing the path length (i.e. hop count) between the communicating nodes, the overall throughput supported by the network can be significantly increased and the available energy of nodes can be consumed more efficiently. On this purpose, we present a distributed technique for repeatedly re-configuring the scatternet topology such that to support with a low number of hops the current traffic flows between all of the communicating peers.

Bluetooth scatternets with dynamic traffic connections can be found in several application scenarios. Beside the well-known conference-room scenario, we can foresee the use of scatternets in *interfering industrial environments* with machinery that autonomously or semi-autonomously accomplishes its tasks. Components of such an automated environment are static and *mobile* robots, sensors of various type and human supervisors. All these components need to be networked for exchanging the data necessary for accomplishing their tasks. Raw data used for the tasks, progress reports and control data are all examples of information that need to be exchanged among the components. Also, each node may have multiple communication peers sustaining *random data traffic sessions* with them, sequentially and/or in parallel.

A data network supporting such a scenario needs to be adaptive for achieving high performance in terms of throughput, power consumption and packet delivery delay. Factors that influence networking predictably in such a scenario, and that in principle can reduce the aggregate system performance, are mobility, interference and random communication sessions. Bluetooth scatternets are a good candidate for supporting such an ad hoc networking scenario since the technology is robust to interference, given its communication mode based on frequency hopping.

The remaining part of this paper is structured as follows. In Section 2 we present the way we build the initial scatternet for our optimization experiments and provide a scatternet model useful for implementing our technique and formalize our optimization problem. In Section 3 some background information are presented, necessary for understanding our approach to scatternet optimization. Section 4 presents our approach to dynamic scatternets and describes our optimization technique while Section 5 presents our experiments.

2 Scatternet Formation and Modeling

In this section, we present how initial scatternets are formed for our optimizations and provide a scatternet model that is used by the optimization algorithms. By means of this model, we also formalize our optimization objectives.

2.1 Scatternet Formation

In order to perform the scatternet optimizations we need an initial functional scatternet. However, for our work it is not significant how the initial scatternet is formed, since the optimizations are based on the idea that the initial network topology will change in time due to mobility and dynamic traffic flows. For our work we implemented a scatternet formation algorithm based on [7, 3], given the good performance in reducing the interference of these algorithms.

We form scatternets in two phases. In the first phase, nodes execute inquiry and inquiry scan with a probability of $1/4$ and $3/4$, respectively. As soon as an inquiring node discovers another node that performs inquiry scan, it pages it. This way the inquiring node becomes a master of the other node in the newly formed piconet. A node may reply to more than one master, thus becoming a slave&bridge between its masters. However, multiple one-hop connections between the same two masters are removed later. By continuing this process, after a while all nodes will be grouped into piconets and one-hop bridges will provide a first-stage connectivity among the nodes. At this point the first phase of the scatternet formation terminates.

The aim of the second phase is to identify master&bridge nodes between those piconets that could not be connected with slave&bridge nodes. In this phase slave nodes alternate between inquiry and inquiry scan states while masters do nothing. If a slave hears the inquiry of another slave it checks with its master(s) the hop count to that slave. If the two slaves are not yet connected or they are at least 6 hops away, the slave answers to the inquiry and becomes a slave&bridge between its old master(s) and the inquiring slave that now becomes a master&bridge. (Notice that the minimum distance between the slaves of two two-hop pure masters is 5, hence the value of 6 above.) In this manner, if physically possible, also those masters that could not be connected with a one-hop slave&bridge will be able to communicate.

At the end of the above two phases of the scatternet formation algorithm we obtain an initial connected scatternet on which we can perform our optimization experiments.

2.2 Scatternet Model

For modeling a scatternet we use the following notations. Let $\mathcal{N}$ be the *set of nodes* in the scatternet, $\mathcal{M}$ the *set of mas-*

ters, and $\mathcal{S}$ the *set of all slaves*. Notice that among masters, pure masters are not elements of $\mathcal{S}$ but master&bridge are elements of $\mathcal{S}$. So, $\mathcal{S} \bigcap \mathcal{M} \neq \emptyset$ if there are master&bridge nodes in the scatternet. We denote by $\mathcal{C}$ the *set of traffic connections* in the scatternet. (Note that in this work the term *connection* refers to one- or multi-hop traffic flows and it should not be confused with the term *link* that we use for a physical Bluetooth radio channel between two nodes.)

The *link matrix* $\mathcal{L} = \{l_{ij}\}$ is defined as

$$l_{ij} = \begin{cases} 1 & \text{if } i \text{ is master of } j, \forall i, j \in \mathcal{N}; i \neq j \\ 0 & \text{otherwise.} \end{cases}$$

The link matrix indicates the master-slave connections in the scatternet. Link matrix properties are explained below.

1) A *master k* has on its row one entry equal to 1 for each of its slaves.

$$\sum_{j=1}^{N} l_{kj} \geq 1$$

2) A *pure slave k* has one entry equal to 1 on its column corresponding to its master.

$$\sum_{i=1}^{N} l_{ik} = 1$$

3) A *slave&bridge k* has on its column exactly one entry equal to 1 for each of its masters.

$$\sum_{i=1}^{N} l_{ik} \geq 2$$

4) A *master&bridge k* node has one entry equal to 1 for each of its slaves on its row and for each of its masters on its column.

$$\sum_{j=1}^{N} l_{kj} \geq 1 \text{ and } \sum_{i=1}^{N} l_{ik} \geq 1$$

5) A *free node k* – a node not belonging to any piconet – has all 0s on both its row and column.

$$\sum_{j=1}^{N} l_{kj} = 0 \text{ and } \sum_{i=1}^{N} l_{ik} = 0$$

The link matrix is subject to the following constraints.

- A piconet must contain one master and up to 7 slaves.

$$\sum_{j=1}^{N} l_{kj} \leq 7, \forall k \in \mathcal{M} \tag{1}$$

- If i is master of j, then j cannot be master of i.

$$l_{ij} + l_{ji} \leq 1, \forall i, j \in \mathcal{N}; i \neq j \tag{2}$$

We also define the *hop matrix* $\mathcal{H} = \{h_{sd}\}$, which contains the minimum number of hops on any connection $(s, d) \in \mathcal{C}$. If we denote by $\mathcal{R}_{sd}$ the set of all possible paths between source s and destination d

$$\begin{aligned} \mathcal{R}_{sd} = & \{\{k_1, \ldots, k_2\} | k_1 = s, k_n = d, \\ & l_{k_1 k_{i+1}} + l_{k_{i+1} k_1} = 1, \forall i = 1, \ldots, n-1\}, \end{aligned}$$

then the minimum number of hops between $s = k_1$ and $d = k_n$ can be obtained as

$$h_{sd} = \min_{K \in \mathcal{R}_{sd}} |K|,$$

where $|K|$ represents the number of nodes in the sequence $K \in \mathcal{R}_{sd}$.

For any pair of nodes (s, d) that do not communicate (i.e., $(s, d) \notin \mathcal{C}$) we have $h_{sd} = 0$.

We define function F as the total number of hops on all traffic connections in the scatternet:

$$F = \sum_{(s,d) \in \mathcal{C}} h_{sd}. \tag{3}$$

Thus, by solving the following minimization problem repeatedly, for the different configurations of the scatternet topology as it changes in time, we manage to keep the length of communication paths shorter, which leads to the objective of our optimizations: higher aggregate throughput and reduced overall power consumption in dynamic scatternets.

$$\mathcal{P}: \qquad \min_{H} F$$

The above problem is solved by our optimization algorithms that in our approach are executed by each node in a distributed fashion, as explained in Section 4.

3 Background

In this section we briefly present the basic ingredients of our optimizations. For details the reader is referred to [5].

After generating a connected and totally functional scatternet and setting up an initial set of traffic connections, network nodes repeatedly reconfigure the scatternet topology in their neighborhood to achieve higher performance for the communication on their connections. In our approach, the neighborhood reconfiguration is achieved through so-called *moves*. A *move* is a set of modifications on the links and master-slave relationship of scatternet nodes. Such modifications are made by link creation, deletion and/or role exchange. Due to these modifications, some nodes may become disconnected, then the operations necessary to reconnect them to the scatternet are considered as a part of the same move.

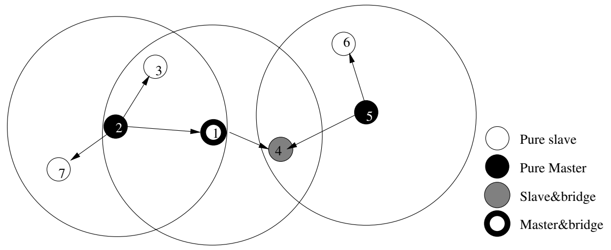

Figure 1. Scatternet snapshot

We identify four kinds of possible moves.

Slave to Slave (SS) – a slave removes its link to its current master and connects to a different master.

Slave to Master (SM) – a slave removes its link to its current master and creates a new piconet by paging another node.

Master to Slave (MS) – a master becomes a pure slave by assigning all of its slaves to other masters.

Master to Master (MM) – merging two piconets: a master overtakes all of the slaves of another master as well as the old master itself.

As an example, consider the scatternet shown in Figure 1. If there is a high traffic flow between slaves 3 and 6 then the scatternet can be optimized by removing node 3 from master 2 and assigning it to master 1. These modifications represent an SS move. It is easy to see that such moves are simple to perform on the above link matrix based scatternet model, since they only imply the modification of several link matrix entries. Also notice that, in order to avoid the interruption of traffic flows through a link that it is to be replaced with another, it is possible to set up the new link first and remove the old link only when its traffic was rerouted on the new link.

The optimizations are executed periodically. We call the time between two executions *optimization period*. Each node may use a different optimization period. Implicit feedback from the scatternet, like the number of recently arrived/left nodes and the gain of previous executions, may be used for dynamically determining the optimization period.

4 Dynamic Scatternets

Users participating in a Bluetooth scatternet may want to communicate with multiple peers sequentially. Therefore, the traffic pattern in a real scatternet in most od the cases is not static, but is changing dynamically. Further, node mobility also introduces a high degree of dynamics in such networks. In this section we present how we address dynamic traffic flows and mobility in our optimizations and we describe the used optimization process.

4.1 Dynamic Connections

In our model, connections are assigned a life time during which we assume that they communicate on the full bandwidth that has been allocated to each of them. A connection is removed as soon as its associated life time expires. For generating new connections, two methods were considered. Originally connections were generated according to the Poissonian distribution. Later, we switched to a simpler connection generation method which simply replaces the expired connections with new ones. Although the Poissonian method is more realistic, it does not guarantee a constant number of connections during the entire simulation. Since the number of traffic connections in the scatternet has a major impact on the aggregate throughput and power consumption, for obtaining a clear view on the performance improvement caused by our optimization algorithms, we need to keep this number constant. Therefore, for the simulations we used the latter approach without loss of generality. By repeatedly replacing expired traffic flows with new ones (i.e. with communication sessions between different end-nodes) we obtain a permanently changing traffic pattern in the scatternet, similar to the traffic flow dynamics in real ad hoc networks. This motivates the periodic execution of the optimization procedure, which reconfigures the scatternet to support the newly evolved traffic pattern more efficiently.

The details of how dynamic traffic connections are embedded in the optimization process can be found in Section 4.3.

4.2 Mobility

For simulating mobility, we use the random waypoint model. Initially, we set a random walking speed in a predefined range and a random moving direction for each node. Then, the direction is periodically changed with an offset in the range [-10,10] degrees with respect to the original direction. A node reaching the boundary of the simulation area is “mirrored” back into the area, that is, the smaller angles between the moving direction and area boundary before and after reaching the boundary are equal.

4.3 Optimization Process

For performing the scatternet optimizations we define the optimization processes presented in Figure 2 and 3. The entire optimization is controlled from the main optimization process (Figure 2) while individual optimizations are performed by the node process (Figure 3) executed by each node. In our approach, these two processes control the dynamics of the network and execute the scatternet optimizations in a decentralized manner.

```
1.  OPTIMIZATION PROCESS
2.  set $unit\_t$, $sim\_t$
3.  set $N$, $nr\_conns$, $conn\_length$
4.  set node positions
5.  set optimization type
6.  build scatternet topology
7.  repeat $nr\_conns$ times
8.      generate new connection $c$
9.          $c.src \leftarrow$ rand($N$)
10.         $c.dst \leftarrow$ rand($N$) $\neq c.src$
11.         $c.expiration\_t \leftarrow$ rand($conn\_length$)
12.     send connection data to $c.src$ NODE PROCESS
13. while $t < sim\_t$ do
14.     if $crt\_nr\_conns < nr\_conns$ then
15.         generate new connection $c$
16.             $c.src \leftarrow$ rand($N$)
17.             $c.dst \leftarrow$ rand($N$) $\neq c.src$
18.             $c.expiration\_t \leftarrow t +$ rand($conn\_length$)
19.         send connect. data to $c.src$ NODE PROCESS
20.     $t \leftarrow t + unit\_t$
21. end OPTIMIZATION PROCESS
```

Figure 2. Pseudo code of the main process

```
1.  NODE PROCESS
2.  get initial data from OPTIMIZATION PROCESS
3.  set $optimiz\_t$
4.  while $t < sim\_t$ do
5.      if $nr\_my\_conns > 0$ and\\
6.      ($t$ - $init\_t$) % $optimiz\_t = 0$ then
7.          $hop\_count \leftarrow$ total hop count on my connects.
8.          execute appropriate optimization module
9.          if $new\_hop\_count < hop\_count$
10.             execute move
11.     forall my connections $c$ do
12.         if $t \geq c.expiration\_t$ then
13.             remove $c$
14.     if new connection initiated by \\
15.     OPTIMIZATION PROCESS or other node then
16.         establish connection
17.         $nr\_my\_conns \leftarrow nr\_my\_conns + 1$
18.         if $nr\_my\_conns = 1$ then
19.             $init\_t \leftarrow t$
20.     calculate new direction
21.     move to new position
22.     $t \leftarrow t + unit\_t$
23. end NODE PROCESS
```

Figure 3. Pseudo code of the node optimization process

The purpose of the main optimization process is to manage the scatternet data necessary for performing simulations and to generate traffic connections in a controlled manner. The main optimization process starts out by initializing the minimum time period, *unit_t*, in which we evaluate the performance of the scatternet and the simulation time, *sim_t*. Further, the number of network nodes *N*, the total number of traffic connections *nr_conns* and the time range *conn_length* that limits the length of connections, are also initialized. The initial positions of the nodes are set in line 4 while the used optimization module is specified in line 5. In line 6 an algorithm is used to form the initial scatternet topology. An initial set of traffic connections is generated in lines $7 - 12$. The source *c.scr* and destination *c.dst* of each connection is randomly selected from the *N* nodes of the network. Each connection is assigned an expiration time *c.expiration_t*, which represents the time instant when the connection will be removed from the network. The newly generated connection data are transmitted to the source node *c.scr* which then will set up the connection in its node process.

In real scenarios, the applicantions running at each device initiate a connection setup. In contrast to that, in our approach we generate the connections in a centralized manner to ensure a constant number of connections through the simulation. This is necessary for the performance evaluation considerations presented earlier in this section, and it does not jeopardize the decentralized nature of our approach.

The main loop of the optimization process starts in line 13. The only objective of this loop is to generate new connections when the current number of connections, *crt_nr_conns*, decreases. The newly generated connection data is transmitted to the appropriate source node, which then sets up the connection. The current simulation time is maintained in the variable *t*, which is synchronized at all of the nodes.

The node process (Figure 3) is run individually by each node. Each node process receives from the main optimization process a set of initialization data that enables the node to participate in the scatternet. For the nodes that start out with one or more connections the relevant connection data is also communicated. These data include also the value of the *init_t* variable, which represents the time instant when the number of connections of this node was changed from 0 to 1. After this centralized initialization each node process operates autonomously. Thus, the optimization period *optimiz_t*, representing the time between two consecutive optimizations, can be initialized with different values for each node.

The main loop of the node process starts in line 4. If the node has at least one connection, it will immediately perform an optimization since we have $t = init\_t$ in the beginning. To decide which optimization to perform, the node will match its role against the optimization type communicated by the main optimization process. The node will execute all of the specified optimizations corresponding to its role. For example, if the SS_MS_MM optimization type was specified, a master would execute the MS and MM optimizations only, while a slave would perform only the SS one.

After the optimization, the hop count on all of the node's connections is counted (*new_hop_count*) and is compared to the *hop_count* before the optimization (measured in line 7). If the hop count has been reduced by the optimization the move is made permanent (line 10).

After finishing the optimization, the node removes its expired links, if any (lines $11-13$), and then checks for newly created connections. New connection notifications may arrive directly from the main optimization process if the node has been selected to be the source. Otherwise, the node will be a destination in a connection initiated by another node. In any case, the node will cooperate in setting up a new connection and increase its connection counter, *nr_my_conns*, by 1. If this is the only connection of the node then it sets the initial optimization time, *init_t*, to the current time instant.

Before moving to the next cycle of the main loop, the node process updates the position, including the moving direction, of the node.

5 Simulation

In this section we present the simulation environment that we used for evaluating our approach. Further, we describe and discuss the experiments that we performed with this simulator.

5.1 UCBT Simulator

For evaluation purposes, we implemented our optimization algorithms in the *UCBT* ns-2-based Bluetooth simulator [2], which is the only publicly available open source Bluetooth simulator that supports mesh-shaped scatternets. We also tried to use the *Blueware* simulator [9], which adds scatternet support to the ns-2 based *Bluehoc* simulator [1] of IBM. However, Blueware supports tree-shaped scatternets only and it does not support slave&bridge type of nodes, thus requiring voluminous modifications for supporting our algorithms that were designed for mesh-shaped scatternets.

UCBT implements the majority of the protocols in the Bluetooth protocol stack. The simulator has recently added support for mesh-shaped scatternets, although only manual scatternet topology formation is possible at the moment. Therefore, in order to test our optimization technique on many scatternets, we also added to UCBT a simple scatternet formation protocol (described in Section 2.1), beside our optimization algorithms and the support for dynamic connections. In the following section we present the experiments that we performed with this simulator in order to evaluate our optimization technique.

5.2 Results

For evaluating our approach we performed experiments of 300s with scatternets made of 50 nodes scattered on an area of $22 \times 22\text{m}^2$. For the experiments we considered both static and dynamic traffic connections, on which data was transmitted in DH5 packets. In both cases we kept the total number of connections constant at 25. This number of connections implies that, on average, each node is involved in one traffic connection either as source or destination. However, there may be nodes participating in multiple connections, while others are not involved in any of them.

In the case of dynamic connections the connection lifetime was randomly distributed in the range of 10 to 30 seconds. After the expiration of a connection, a new one was generated on its place to keep the number of connections constant for the entire simulation, as explained in Section 4.1. Considering an average connection lifetime of 20s, on 25 connections we obtain 375 connection replacements during a 300s simulation. These settings enable the observation of the scatternet performance in its steady state of operation when nodes continuously change their communication peers.

We evaluated the scatternet performance with mobile nodes moving with different walking speeds from 0 to 1.2m/s. With higher speeds the topology changed too fast, so the optimizations could not have long-lasting effect.

The main simulation results are shown in Figure 4. In the left and right side of the figure we show the scatternet performance with static and dynamic connections, respectively. The main metrics that we are interested in are the overall throughput and power consumption in the scatternet. Further, since an increased throughput implies higher power consumption because of the higher number of transmitted bits, we also use the energy efficiency metric to express the number of bits transmitted with the unit of energy.

In Figure 4.a, we show the evolution of the aggregate throughput in the scatternet with different optimization types and compare the results to the non-optimized scatternet throughput. In the throughput calculations we considered those packets only that reached the destination. Corrupted and lost packets were not considered. As shown in the figure, our optimizations in both static and dynamic connections improved the overall throughput significantly with respect to the non-optimized case. Also, in both cases the negative impact of mobility on the throughput shows up im-

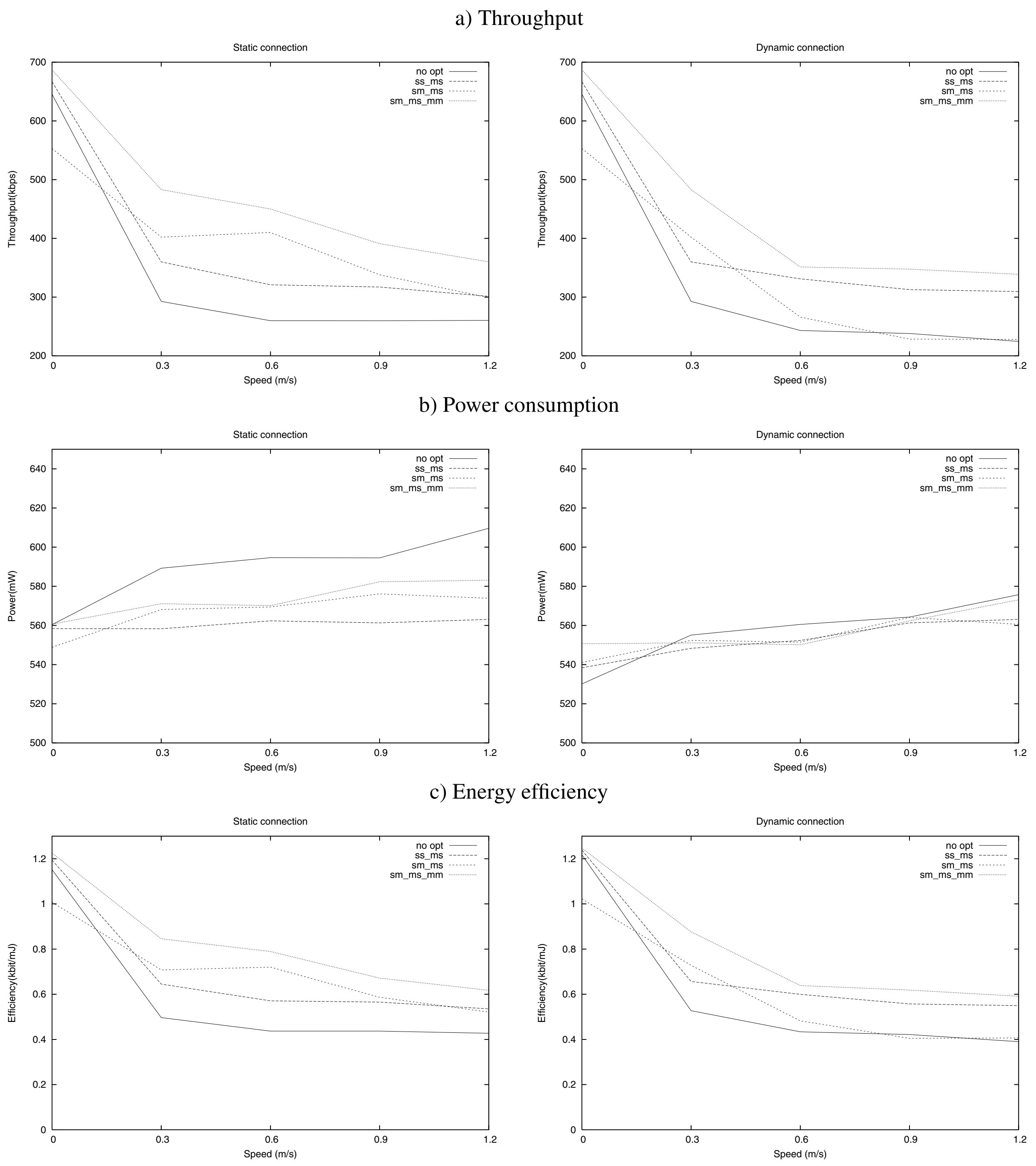

Figure 4. Disseminated simulation results with static (left) and dynamic (right) traffic flows: a) Throughput; b) Power consumption; c) Energy efficiency

mediately as soon as we set a very low moving speed of 0.3m/s for the nodes. When the nodes do not move (i.e. at 0m/s) the optimizations can not provide such a big throughput improvement like later on. This shows that our optimizations make sense in dynamic scenarios. However, if the moving speed increases, even at running speeds the im-

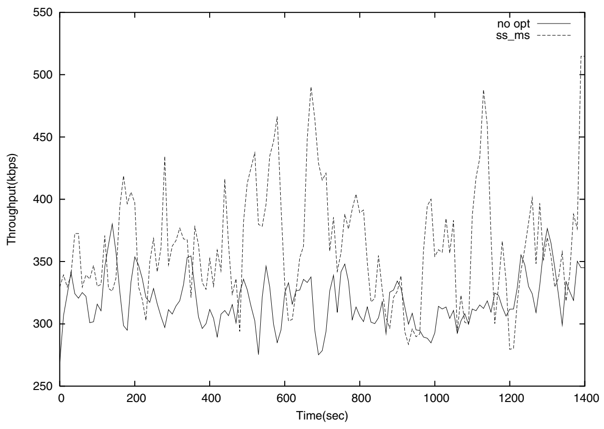

Figure 5. Throughput

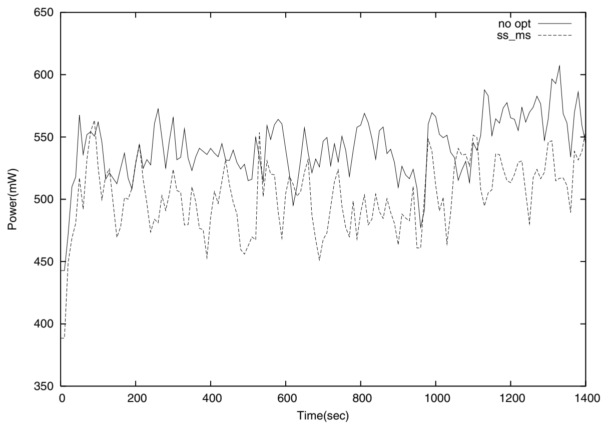

Figure 6. Power consumption

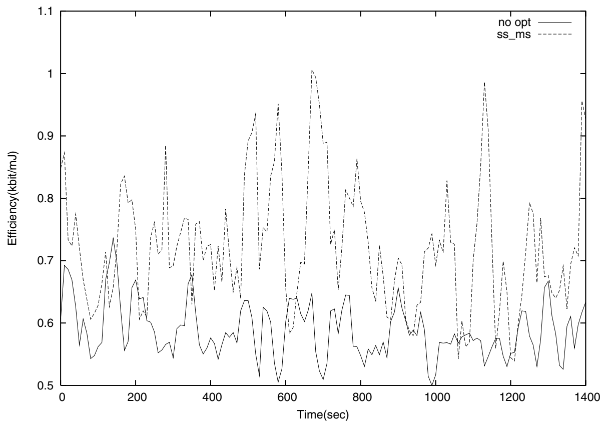

Figure 7. Energy efficiency

pact of the optimizations becomes insignificant, since the optimized topology changes before the communication can take advantage of it.

Mobility has its individual impact on the scatternet topology, therefore the network is not exclusively shaped by our optimization algorithms. Occasionally this can lead to situations where the optimized case produces somewhat worse results than the non-optimized one. This may happen when the optimization algorithms can not provide a significant hop count reduction and mobility reconfigures the network topology very unfavorably with respect to the current traffic connections.

The dynamic traffic flows have not as high impact on the throughput as mobility. However, it can be seen in the figure that the overall throughput in the static case is generally higher and it decreases more slowly. The best optimization was produced by the most complex optimization type used, SM_MS_MM, because this optimization takes advantage of three-move types to find a better topology instead of the two-move types used by the other two optimizations.

The evolution of the power consumption is presented in Figure 4.b. The power consumption is calculated at the lower layers (i.e. Baseband), therefore it includes also the power consumed for transmitting lost packets. Therefore, packet retransmissions have a significant impact on the power consumption. Since static connections last for the entire duration of the simulations (dynamic connections last $10-30$s only) there will be more retransmissions in the static case because after the termination of a dynamic connection no retransmissions are done for the lost packets of that connection. This is one of the reasons why the power consumption in the case of dynamic connections is significantly lower than with static connections. Another reason is that the higher throughput obtained with the static connections consumes more power as well. This is confirmed also by the different optimizations. For instance, while the SM_MS_MM optimization produced the highest throughput improvement, it also implied the highest consumption of energy.

The impact of mobility can be seen also on the power consumption. In both cases the power consumption increases with the moving speed of the nodes.

As mentioned above, the optimization types that produce bigger throughput improvements imply higher energy consumption as well. To demonstrate that it is still worth performing these optimizations, we define a metric that we call *energy efficiency* as the ratio of the throughput to the power consumption. This metric expresses the amount of bits transmitted with the unit of energy. We show the energy efficiency of the simulated scatternets in Figure 4.c. It can be seen in the figure that the energy efficiency decreases with the the moving speed and that scatternets with dynamic connections use the energy less efficiently than those with static connections. However, using our optimization algorithms, we can still obtain an improvement of $30 - 40\%$ on this metric.

In Figures 5–7 we present a sample simulation run with the SS_MS optimization in case of dynamic connections with the node moving speed of 0.3m/s. The graphs show the evolution of the throughput, power consumption and energy efficiency with and without optimization. It can be seen that the optimizations provide significant improvements on all these metrics.

6 Conclusion

In this paper we presented a decentralized optimization technique for dynamic Bluetooth scatternets. Our optimizations are based on an algorithm suite capable of reconfiguring the scatternet topology in order to support the current traffic flows between the nodes with shorter communication paths. Shorter paths imply that packets occupy less bandwidth and consume less energy, because they are retransmitted by a lower number of nodes. Thus, a significant improvement on the overall scatternet throughput and power consumption can be achieved through hop count reduction. Our simulations show that using such optimizations the available energy of the nodes can be consumed about 40% more efficiently than without the optimizations.

Although our results are promising, we still need to perform more extensive simulations on bigger scatternets and with higher moving speeds. Testbed experiments should also be performed in order to verify our approach in real environments as well. This would help us to obtain a more accurate understanding also on the connection setup delays and power consumption at higher layers. Finally, in the future we also propose to improve our optimization algorithms to support high mobility.

Acknowledgement

This work was partially funded by the MICRO/STMicroelectronics award number 03-032a and by the Autonomous Province of Trento through the WILMA project. We also thank Qihe Wang for his help with using the UCBT simulator.

References

[1] Bluehoc simulator. http://oss.software.ibm.com/bluehoc/, Last accessed: December 2004.

[2] UCBT simulator. https://www.ececs.uc.edu/ cdmc/ucbt/, Last accessed: December 2004.

[3] S. Basagni and C. Petrioli. A scatternet formation protocol for ad hoc networks of Bluetooth devices. In *IEEE Vehicular Technology Conference (VTC)*, pages 424–428, 2002.

[4] Bluetooth SIG. *Bluetooth Specification v2.0.* 2004.

[5] C. Kiss Kalló, C.-F. Chiasserini, R. Battiti, and M. Ajmone Marsan. Reducing the number of hops between communication peers in a Bluetooth scatternet. In *IEEE Wireless Communications and Networking Conference (WCNC'04)*, Atlanta, GA, USA, March 2004.

[6] C. Kiss Kalló, S. Jung, L.-J. Chen, M. Brunato, and M. Gerla. Throughput, energy and path length tradeoffs in Bluetooth scatternets. In *IEEE International Conference on Communications (ICC'05)*, Seoul, Korea, May 2005.

[7] C. Law, A. K. Mehta, and K.-Y. Siu. A new Bluetooth scatternet formation protocol. *Mobile Networks and Applications*, 8:485–498, October 2003.

[8] T. Salonidis, P. Bhagwat, L. Tassiulas, and R. LaMaire. Distributed topology construction of Bluetooth personal area networks. In *IEEE INFOCOM*, Anchorage, April 2001.

[9] G. Tan. Blueware: Bluetooth simulator for ns. Technical Report MIT-LCS-TR-866, MIT, Cambridge, MA, USA, October 2002.

[10] Z. Wang, R. J. Thomas, and Z. Haas. Bluenet – a new scatternet formation scheme. In *35th Annual Hawaii International Conference on System Sciences*, Big Island, Hawaii, 2002.

[11] G. V. Záruba, S. Basagni, and I. Chlamtac. Bluetrees - scatternet formation to enable Bluetooth-based ad hoc networks. In *ICC 2001*, volume 99, pages 273–7, 2001.