An Efficient Scatternet Formation Algorithm for Dynamic Environments

Godfrey Tan, Allen Miu, John Guttag, and Hari Balakrishnan
MIT Laboratory for Computer Science
Cambridge, MA 02139
{godfreyt, aklmiu, guttag, hari}@lcs.mit.edu

ABSTRACT
There is increasing interest in wireless *ad hoc* networks built from portable devices equipped with short-range wireless network interfaces such as Bluetooth. This paper addresses issues related to internetworking such networks to form larger "scatternets." Within the constraints imposed by the emerging standard Bluetooth link layer and MAC protocol, we describe an efficient online scatternet topology formation algorithm, called TSF (Tree Scatternet Formation). TSF connects nodes in a tree structure that simplifies packet routing and scheduling. Unlike earlier work, our protocol is designed to work well with dynamic environments where nodes arrive and leave arbitrarily. TSF incrementally builds the topology and healing partitions when they occur. We have developed a Bluetooth simulator in *ns* that includes most aspects of the entire Bluetooth protocol stack. Using this, we derive simulation results that show that TSF has low latencies in link establishment, tree formation and partition healing, all of which grow logarithmically with the number of nodes in the scatternet. Furthermore, TSF generates tree topologies where the average path length between any node pair grows logarithmically with the size of the scatternet.

KEY WORDS
Bluetooth Scatternet, Dynamic Topology Contruction

1 Introduction

Bluetooth [1] is emerging as an important standard for short range, low-power wireless communication. It's link-layer medium access (MAC) protocol is designed to facilitate the construction of *ad hoc* networks without manual configuration or wired infrastructure.

Bluetooth communication is based not on distributed contention resolution, as in traditional wireless LANs, but on a *time-division duplex (TDD)* master-slave mechanism. A Bluetooth *piconet* consists of one master and up to seven slaves. The master allocates transmission slots[1](and therefore, channel bandwidth) to the slaves in the piconet. The basic idea is for the master and slaves to use alternate transmission slots, with each odd slot being used only by the slave to which the master sent a frame in the previous even transmission slot.

[1]A Bluetooth link has a maximum capacity of 1Mbps and each time slot takes 625 microseconds.

Frequency hopping is used to permit multiple concurrent Bluetooth communications within radio range of each other, without adverse effects due to interference. This facilitates high densities of communicating devices, making it possible for dozens of piconets to co-exist and independently communicate in close proximity without significant performance degradation. In principle, this raises the possibility of internetworking multiple piconets. The Bluetooth specification alludes to this possibility, calling it a *scatternet,* but does not specify how it is to be done.

In this paper, we present an efficient scatternet formation algorithm, called TSF (for Tree Scatternet Formation). The algorithm, combined with a distributed scheduling algorithm described in [2], provides a complete approach to building connected scatternets for applications that require fast network formation and connectivity preservation in a dynamic environment. TSF

1. Decides dynamically and in a distributed fashion which nodes act as masters and which as slaves, thus avoiding manual configuration of roles to nodes or centralized decision making,
2. Exploits the asymmetry in Bluetooth's link discovery protocol to make efficient use of resources, e.g., energy,
3. Is completely decentralized in that nodes maintain information only about adjacent nodes,
4. Enables nodes to begin communication while the scatternet is being constructed,
5. Is self healing in that nodes can join and leave at any time without causing long disruptions in connectivity, and
6. Creates a topology that simplifies routing and scheduling without sacrificing communication latency.

In Section 2, we explain the Bluetooth link formation process, prior work on scatternets and the design rational behind TSF. Section 3 describes in detail our algorithm, TSF. Section 4 evaluates the efficiency of TSF using a detailed Bluetooth simulator and analyzes the properties of resulting topologies.

2 Background

In this section, we provide background information about the relevant aspects of Bluetooth. We start by describing how two nodes establish a bi-directional communications link. An understanding of this link formation process is necessary to understand our topology formation algorithm.

2.1 Bluetooth Link Formation

The link formation process specified in the Bluetooth Baseband specification consists of two processes: $Inquiry$ and $Page$ [1]. The goal of the Inquiry process is for a master node to discover the existence of neighboring devices and to collect enough information about the low-level state of those neighbors (primarily related to their native clocks). The goal of the Page process is to use the information gathered during the Inquiry process to establish a bi-directional frequency hopping communication channel.

During the Inquiry process, a device enters either the Inquiry or the Inquiry Scan state (mode). A device in the Inquiry state repeatedly alternates between transmitting short ID packets containing an Inquiry Access Code (IAC) and listening for responses. A device in the Inquiry Scan state constantly listens for packets from devices in the Inquiry state and responds when appropriate. It is important to note that once a node is in the Inquiry state, it remains in that state for several seconds. A node periodically (every $1.28s$ or so) enters the Inquiry Scan state to scan continuously over a short window of $11.25ms$, and thus, can communicate with other nodes or sleep in between consecutive scans. Thus, the duration that a node stays in Inquiry or Inquiry Scan mode is asymmetric.

A node remains in the Inquiry state until a timeout period elapses, keeping track of which nodes respond during this time. After this time, if the number of responses is greater than zero, it enters the Page state. Analogously, a node in the Inquiry Scan state also periodically enters the Page Scan state. A device in the Page state uses the signaling information obtained during the Inquiry state and sends out trains of ID packets based on the discovered device's address, BD_ADDR[2]. When the device in the Page Scan state responds back, both devices proceed to exchange necessary information to establish the master-slave connection and eventually enter the Connection state. The device in the Page state becomes the master and the device in the Page Scan state the slave. It is important to note that the time taken to complete the Page process (in the order of milliseconds) is typically much smaller than that of the Inquiry process (in the order of seconds).

To avoid manual configuration of nodes to carry out Inquiry or Inquiry Scan operations, earlier work in [3, 4] suggests that nodes alternate between the Inquiry and Inquiry Scan operations continuously while randomizing the interval to carry out each operation. We introduce a new mechanism to discover neighboring nodes. A node that is not connected to any other nodes alternates between the two Inquiry modes until it is connected to another node. However, unlike earlier schemes, an existing node in the scatternet only conducts periodic Inquiry Scan. Thus, as more and more nodes are connected, it gets faster for a new unconnected node to attach to the existing network since multiple nodes in the network are listening to the Inquiry messages sent out by the unconnected node conducting Inquiry. The asymmetric nature of the protocol not only speeds up the connection setup delay and scatternet formation delay but also improves the power consumption significantly.

2.2 Related Work

The research area of developing topology construction protocols necessary to form piconets and interconnect them via bridges is still active. Salonidis *et al.* present a scatternet formation scheme that uses an election process to elect a leader to configure a particular scatternet topology [3]. The scheme described limits the maximum number of nodes involved in the scatternet formation to be 36. Law *et al.* have developed a randomized distributed scatternet formation protocol and evaluated the performance [4]. Their scheme attempts to form scatternets with the number of piconets close to the minimum while limiting the piconet membership of each device to at most two. Their protocol relies on synchronized rounds where devices discover their neighbors simultaneously. Both protocols are only meant to work in environments where every node can hear other.

Zaruba *et al.* have presented a high level solution to build tree scatternets [5]. They do not, however, describe in detail how nodes discover each other and establish links using Bluetooth primitives. Schemes presented in [6, 7] build looped topologies and do not have the requirement that nodes are within the transmission range of each other.

However, to our knowledge, none of the previous schemes deals with dynamic environments where nodes may arbitrarily join or leave the network. In contrast, our approach is intended for dynamic environments with the following characteristics:

1. Nodes arrive and depart at arbitrary times,
2. The node density is high,
3. Nodes have stringent energy requirements,
4. Most applications run in the network are low-bandwidth[3]but latency sensitive such as mouse controller applications and instant messaging, and
5. The speed with which nodes are connected and topologies are formed and healed is a dominant concern.

[2]BD_ADDR is the globally unique 48-bit address of the Bluetooth device.

[3]We argue that for high-bandwidth applications, 802.11b technology should be used instead of Bluetooth.

Example usage scenarios include interactive conference scenarios and *ad hoc* information exchanges in shopping malls or open fields. We explain the design rationale behind TSF in the next section.

2.3 Design Rationale

An important goal is to simplify link scheduling. As authors in [8] argue, scatternet-wide coordination of link schedule is difficult for looped networks. This led them to invent a randomized scheduling scheme. Authors in [9] prove that constructing an optimal link schedule that maximizes total throughput in a Bluetooth scatternet of arbitrary topology is an NP-hard problem, even if a central entity coordinates the schedule.

These kinds of concerns led us to design a topology formation algorithm that is guaranteed to yield a loop-free topology. Additionally, the topology formation process is designed to interact well with a companion link scheduling scheme for loop-free scatternets described in [2]. The tree hierarchy allows a scheduling algorithm to resolve scheduling conflicts and adapt to changing traffic loads in a top-down fashion without incurring significant overheads. This maximizes the number of links that can communicate simultaneously, and thus, increases the scatternet's realized capacity and effectively reduce communication latency.

Compared to many previous approaches [3, 6, 7], TSF simplifies routing because there is no need to worry about routing loops and there exists a unique path between any two nodes. Nodes can be assigned unique addresses based upon their position in the tree. Higher-layer destination identifiers (*e.g.*, IP addresses) can be mapped to these addresses using a mechanism like the address resolution protocol (ARP) that returns a node's scatternet address in response to an ARP query. Armed with this scatternet identifier, the packet forwarding protocol works by simply having each node look at the destination and forward it along one of its links. Thus, a tree topology facilitates *stateless ad-hoc routing*. In many cases, this approach is more efficient than traditional *ad hoc* routing protocols , which either incur per-packet overhead as in Dynamic Source Routing (DSR) [10] or increase memory requirements as in Ad-hoc On-Demand Distance Vector (AODV) [11].

A major problem that needs to be solved in forming scatternets is deciding which nodes should act as relays that interconnect piconets. In particular, one must decide whether to use master relay or slave relay nodes. Our algorithm does not rely on which relay nodes are used. However, for simplicity, we currently use master relay nodes.

3 TSF: Tree Scatternet Formation

This section presents *TSF*, a tree scatternet formation algorithm that has the following desirable properties.

1. Connectivity: TSF rapidly converges toward a steady-state in which all nodes can reach each other. At any time, the topology produced by TSF is a collection of one or more rooted spanning trees, which are each autonomously attempting to merge and converge to a topology with a smaller number of trees.

2. Healing: TSF handles nodes arriving incrementally or *en masse*, and nodes departing incrementally or *en masse*, avoiding loops and healing network partitions.

3. Communication and topology formation efficiency: TSF produces topologies where the average node-node path length is small (logarithmic in the number of nodes, avoiding long chains). TSF uses a randomized protocol to balance the time spent by nodes already in the scatternet between communicating data and performing the social task of forming a more connected scatternet.

3.1 State Machines

At any point in time, the TSF-generated scatternet is a forest consisting of connected tree components. Some of these trees are single nodes, called *free nodes*, that are seeking to join another tree to form a larger component and reduce the number of components. We refer all other nodes in a component other than the *root node* as *tree nodes*. Each node in each tree component spends a small amount of time to attempt to rendezvous with another node belonging to a different tree to eventually form a single connected tree scatternet. To preserve loop-freeness, TSF distinguishes between two kinds of component merges: i) merges of trees each having more than one node, and ii) merges of trees, one of which is a free node. The former can cause loops whereas the latter cannot. For trees with at least two nodes, TSF designates a subset of nodes from each tree to be the *coordinators* which are responsible for merging with neighboring trees. All other tree nodes that are not coordinators spend a small of time passively listening to Inquiry messages sent out by free nodes and coordinators. Free nodes constantly search for other tree or free nodes to establish communication links.

TSF is distributed with each node operating autonomously with only local communication. There are three states: Inquire, Scan, and Comm. Each node in the network runs a state-machine algorithm, alternating between two of the three possible combination of states: Inquire, Comm and Comm/Scan. A node in the Inquire state performs the Bluetooth Inquiry operation. In the Comm state, a node is either idle or involved in data communication with other nodes in its connected component. In particular, free nodes in the Comm state remain idle to save power. Similarly, a node in the Comm/Scan state begins in the Comm state while entering the Scan state periodically to perform the Bluetooth Inquiry Scan operation. Thus, in the Inquire and Scan states, a node attempts to rendezvous with another node belonging to a different tree, to form a Bluetooth communication link and thereby improve the connectedness

```
PROCEDURE TSF-FREE() {
    state_pair ← (Inquire, Comm/Scan)
    RUNFOREVER(state_pair, GIAC, 1)
}
PROCEDURE TSF-ROOT() {
    if(designated as coordinator)
        TSF-COORDINATOR()
    else
        Run(Comm, $\infty$, null)
}
PROCEDURE TSF-TREE() {
    if(designated as coordinator)
        TSF-COORDINATOR()
    else
        state_pair ← (Comm, Comm/Scan)
        RUNFOREVER(state_pair, GIAC, n_links)
}
PROCEDURE TSF-COORDINATOR() {
    state_pair ← (Inquire, Comm/Scan)
    RUNFOREVER(state_pair, LIAC, 1)
}
PROCEDURE RUNFOREVER(state_pair, iac, $f_{comm}$) {
    state ← random state from state_pair with 0.5 prob
    do forever
        if(state = Comm)
            t_state ← $f_{comm} \times$ random($E[T_{inq}], D$)
        else
            t_state ← random($E[T_{inq}], D$)
        Run(state, t_state, iac)
        state ← opposite state from state_pair
}
```

Figure 1. Pseudo-code of various TSF state-machines.

of the scatternet. While conducting Inquiry or Inquiry Scan operations, a node use one of two different Inquiry Access Codes to limit merges to suitable kinds of nodes. In particular, *coordinators* use the Limited Inquiry Access Code (LIAC) and all other nodes use the Generic Inquiry Access Code (GIAC). Thus, communication between *coordinators* is isolated from the rest of the nodes as *coordinators* only transmit and listen to LIAC.

The pseudo-code for different state-machines running at various types of nodes is shown in Figure 1. RUNFOREVER initializes $state$ randomly and alternates between $state\_pair$ while randomizing the state residence time, t_state. f_{comm} only applies to tree nodes as explained shortly. Run procedure asks the Bluetooth lower layers to carry out the corresponding operation based on $state$ for t_state amount of time. The specified iac is used as the Inquiry Access Code when conducing Inquiry or Inquiry Scan operations. TSF's state residence time is randomized to avoid periodic synchronization effects. The randomization depends on two parameters: $E[T_{inq}]$ and D. $E[T_{inq}]$ is the expected time taken to complete the Inquiry process. D is a parameter deciding the size of the random interval, which governs how long the node is resident in a given state.

As shown in the pseudo-code, free nodes and coordinator nodes spend roughly equal amount of time in each of the two alternating states to probe and scan for possible connections. Root nodes always remain in the Comm state. For tree nodes, f_{comm} specifies the amount of time spent in the Comm state, which is a function of how busy a node is likely to be in performing its communication tasks. In the interest of simplicity, we approximate the ideal f_{comm} value as a function of how many links a node has, n_links.

The final piece of the TSF algorithm concerns loop-avoidance, which helps preserve the invariant that as nodes join and leave, the scatternet remains a forest. To achieve this, TSF only allows root nodes to heal partitions and join another tree as a child by conducting Page and Page Scan operations. Roots, however, do not spend any time in either Inquire or Scan states. Instead, each root designates a node in its component tree as the coordinator which is responsible for discovering neighboring coordinators. TSF's separation of the component merges from discovery ensures that root nodes are not overburdened with the energy-intensive task of performing Inquiry. In the next section, we explain in detail how coordinators are elected and how component merges take place.

3.2 Forming Communication Links

In the Inquire and Scan states, nodes attempt to establish connections with other nodes. As soon as a node successfully receives an inquiry response from another node, the two nodes immediately enter the Page and Page Scan modes, and attempt to establish a connection. When two free nodes connect, the master node becomes the root and the slave becomes a leaf node.

Every root node elects a single coordinator responsible for discovering other tree scatternets. If the root has only one child, it elects itself as the coordinator. Otherwise, it picks one of its children randomly and asks it to elect the coordinator by sending a request packet. If the chosen child node is not willing to become the coordinator, the child node again elects one of its children randomly. This process continues recursively until a coordinator is selected or a leaf node is reached. A leaf node must become a coordinator once elected. Clearly, leaf nodes are not communication bottlenecks and therefore, have more spare capacity for discovering neighboring devices. Once a node becomes the coordinator, it sends an acknowledgment packet to its root.

As explained previously, coordinators in the Inquire or Scan states, search for other coordinators. When two coordinators establish a communication link, they each inform the corresponding root nodes with necessary signaling information to enter the Page and Page Scan modes. Coordinators then break the link between them and resume their previous roles. Meanwhile, the root nodes establish the connection quickly and the master node becomes the new root node and the slave becomes its child node forming a larger tree and reducing the number of component trees in the forest. The root then selects another coordinator randomly. An important detail here is that the coordinator node may disappear abruptly (e.g., by crashing) without informing the root. We solve this by limiting the life time of the coordinator role to TO_{coord} slots and having the root elect a new coordinator periodically. In addition to making the protocol more robust, this distributes the energy-intensive task of discovering other coordinators uniformly over a large number of nodes.

When two roots form a link, one of them assumes the root role of the combined scatternet, and the other becomes a tree node. Tree nodes alternate between the Comm and Comm/Scan states. They may connect to free nodes as slaves. It is clear that TSF produces loop-free topologies since since component merges are only carried out by the root nodes. We omit a simple proof due to space constraints.

To avoid periodic synchronization effects, TSF randomizes the state residence time from an interval $[E[T_{inq}], D]$. D is based on the expected time for two Bluetooth nodes to discover each other and successfully establish a communication link. If D is too short, the chances of establishing a connection during a slot in which the opportunity for a establishing a connections exists will be too low. If D is too long, a great deal of time (and power) will be wasted during slots in which there is no opportunity to establish a connection. We ran simulations to calculate a good value for D. This is presented in Section 4.

3.3 Healing Partitions

Self-healing is an important requirement for a topology formation scheme, especially in networks in which some nodes are energy-constrained (and thus, may run out of batteries) and many are mobile. We assume that nodes in the network may arbitrarily leave resulting in network partitions. TSF ensures that network partitions heal properly within a reasonable amount of time.

We distinguish two ways in which connectivity can be lost: when a node loses the connection to its child node, and when a child node loses the connection to its parent. When a parent detects the loss of a child, it does not need to do anything except decide if it has become a free node and update its node type accordingly. When a child loses the connectivity to its parent, it updates its node type as follows. A leaf node becomes a free node and an internal node becomes a root node. Each node continues to execute an appropriate state machine as explained earlier.

4 Performance Evaluation

To evaluate the effectiveness of our algorithms, we have developed a Bluetooth simulator as an extension to the Network Simulator (ns) [12]. Our development efforts were eased by the *bluehoc* simulator developed by IBM [13]. Our simulator implements all important aspects of the Bluetooth protocol stack according to the Bluetooth Specification Version 1.1. and hence gives us insights in understanding many of the engineering difficulties as well as performance aspects. We plan to release the simulator in the public domain in the coming weeks.

We conducted several simulations to evaluate the performance of TSF on two different environments: *en masse* arrivals, and *incremental arrivals and departures*. In this section, we present empirical results on link establishment, scatternet formation, and healing latencies, and discuss salient properties of the resulting topologies.

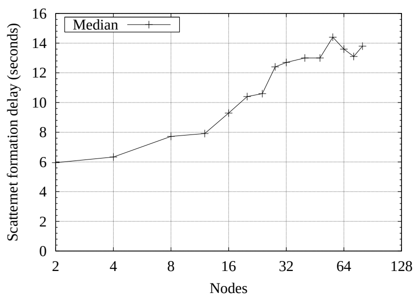
In all the experiments, nodes are assigned to a random clock value between 0 and $2^{27} - 1$. Every data point shown in all figures is the average of 100 independent trials. Through simulation, we determined that the expected time to complete the inquiry process, $E[T_{inq}] = 1.8s$, when two nodes are performing opposite discovery operations namely Inquiry and Inquiry Scan. We also ran numerous simulations with two nodes running TSF to determine a good value for D. We found that setting $2.2 * E[T_{inq}] \leq D \leq 4.4 * E[T_{inq}]$ would give an average connection delay of $6 - 8$s and chose $D = 3.8 * E[T_{inq}]$ for all the simulation runs.

4.1 *En masse* Arrivals

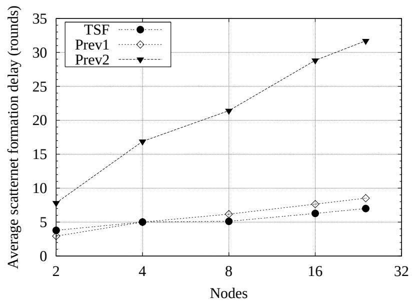
We start by evaluating the performance of TSF when nodes are arriving *en masse* and no nodes are leaving. This scenario models an *ad hoc* interactive meeting where participants carrying Bluetooth devices arrive simultaneously.

Figure 2(a) plots the median delays taken by TSF to build a scatternet for n nodes arriving en masse. It shows that the delay grows logarithmically with the size of the scatternet. The median delays required for 2 and 64 nodes to form a scatternet are $6s$ and $14s$ respectively. We give an intuition why TSF achieves a logarithmic average scatternet formation delay. Whenever a valid communication link is established, the number of components is reduced by 1. The number of parallel links being formed increases linearly with the number of discovering nodes. Since there is at least one node in each component looking for other components, a fraction of the components merge together every time unit.

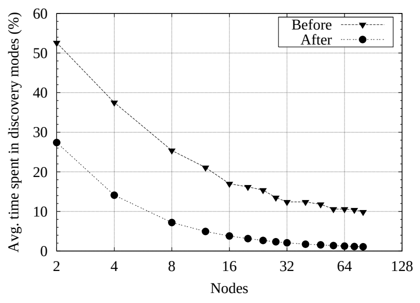
It is difficult to quantitatively compare TSF's performance with the two previous schemes [3, 4] which have been developed under different simulation environments. In particular, their schemes have different assumptions on the efficiency of the Bluetooth link formation process carried out by two nodes in the Inquiry and Inquiry Scan modes respectively. Schemes in [3, 4] assume that nodes have synchronized clocks and therefore, $E[T_{inq}] = 0.34s$ is dominated by the random backoff time between 0 and $0.64s$. We decide not to use this assumption for practical reasons and as described earlier, we found that $E[T_{inq}] = 1.8s$. We conclude that a sensible comparison between the three schemes will be to compare the scatternet for-



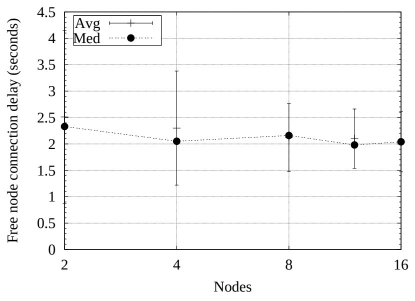
(a) Scatternet formation delay



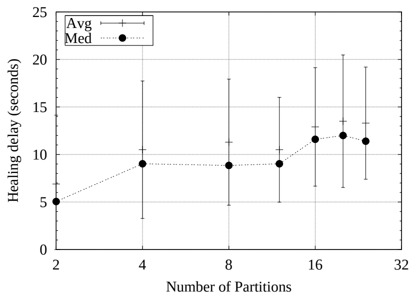
(b) Delay Comparison



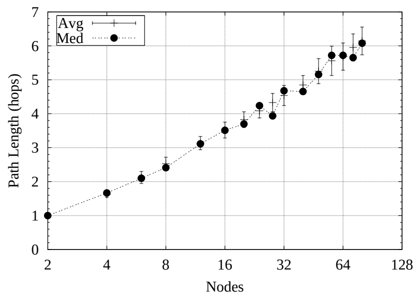
(c) Avg. time spent in Inq/Page modes



(d) Avg. free node connection delay



(e) Delay for healing network partitions



(f) Avg. path length

Figure 2. Performance of TSF in dynamic environments.

mation delay in terms of round which is simply $E[T_{inq}]$. Figure 2(b) plots the average scatternet formation delay in rounds achieved by each of the three schemes. The data points for the two previous schemes are obtained and normalized from [3]($Prev1$) and [4]($Prev2$) respectively. TSF outperforms both schemes in forming scatternets with the exception of $n = 2$. Every data point we use for $Prev1$ represents the ideal time taken to elect the leader from n nodes during Phase I as described in [3]. The actual scatternet formation delay will be a little bit longer since the leader needs to connect to other nodes waiting in the Page Scan mode and so on. As explained in [4], the scatternet formation delay achieved by $Prev2$ is longer than the other two schemes due to the synchronized nature of the algorithm. We also note that the comparison will be much more meaningful if all three schemes are developed under the same environment.

We also measured the amount of time that a node spent in discovering neighbors (Inquiry process) and establishing connections with them (Page process) for two phases: before and after the connected scatternet was formed. Figure 2(c) plots the percentage of time that a node spends in discovery modes before and after a connected scatternet is formed. Not surprisingly, the percentage decreases with the increase in scatternet-size. As explained in Section 2, the time spent in the Inquiry mode dominates the total time required to establish a connection. This is apparent in the 2-node case where each free node spends an equal amount of its time alternating between the Inquire and Comm/Scan states. However, the average time spent by each node in discovery modes is only 52% of the total time. Since only a single coordinator from each component performs Inquiry, the average time a node conducts Inquiry decreases as the scatternet-size increases. The *Before* curve clearly demonstrates that TSF allows a newly connected node to begin communicating with other nodes in its connected component while building up a single connected scatternet. Furthermore, TSF only requires each node to spend a small amount of time in discovery modes (less than 4% for scatternet-size greater than 16) to connect to nodes arbitrarily arriving after a connected scatternet is formed.

4.2 Incremental Arrivals and Departures

We now analyze the performance of TSF in highly dynamic environments such as stores and coffee shops in malls and airports. In these environments, there will usually exist a connected scatternet; thus, we are interested in how fast a newly arrived node can connect to an existing scatternet and how fast the network can heal when nodes leave arbitrarily. We note that the earlier approaches only work when nodes arrive *en-masse* and no nodes leave the network.

We setup a 32-node scatternet and have n nodes arrive randomly over a period of 30 seconds. We then measure the average link establishment delay for various number of arriving nodes. As the arriving nodes are spread out over the $30s$ period, every arriving free node in all the trials connects to an existing non-root node as opposed to connecting to another free node. As shown in Figure 2(a), the average link establishment delay is always less than $2.5s$. The delay goes down slightly as the number of nodes increase. This is because as the tree gets larger and larger, it gets a bit faster for a free node to get attached to a non-root node.

When nodes arbitrarily leave, the scatternet will be partitioned into several smaller networks. We measure how quickly TSF heals network partitions. As explained in Section 3.3, coordinator nodes, one from each network partition, attempt to connect to each other during the healing process. Intuitively, the time taken to heal the network partitions increases as the number of partitions increases. Figure 2(e) shows that TSF achieves the average healing delay that grows logarithmically with the number of network partitions.

The topology of a Bluetooth scatternet affects the overall network capacity and average latency between any two nodes. In multi-hop networks, the path length or hop count between communicating nodes greatly influences the end-to-end latency. Figure 2(f) shows that the average path length grows logarithmically with the number of nodes contained in the scatternet.

5 Discussion

TSF is optimized for those situations in which rapid node connectivity and topology formation is more important than maximizing throughput. It allows nodes to arrive and leave at arbitrary times, incrementally building a tree topology and healing partitions when they occur. Furthermore, it allows an arriving node to begin communication with nodes in its connected component as soon as it is connected. TSF when integrated with the scheduling scheme in [2] provides a complete solution to realize self-organizing scatternets for dynamic environments.

Our simulation results show that the tree scatternet formation latency is logarithmic in the number of nodes. In addition, TSF achieves low average link establishment delay while requiring each node to spend just a small amount of time (5%) discovering neighbors. The average path length between nodes is also logarithmic in the size of the network.

Although TSF is guaranteed to produce a connected scatternet when nodes are within radio proximity, it may not be able to heal all partitions when they are not. This is because TSF limits the tasks of discovering and merging partitions to coordinators and roots respectively. Thus, the algorithm as currently implemented fails to create a single connected scatternet when either coordinators or roots cannot hear each other. TSF can be extended to work for networks with diameter larger than one. A companion loop-detection protocol can be developed to guarantee the formation of a tree scatternet whenever there exists a connected physical topology.

For high bandwidth applications, the topologies produced by TSF will create bottlenecks at the root. However, once a tree topology is constructed quickly and communication is enabled, the topology can be adjusted to suit the traffic patterns. For stable networks with several bandwidth-intensive applications, this should be done.

As explained in Sections 1 and 2, our protocol is targeted for those environments where nodes arrive and depart arbitrarily, node density is high, nodes are energy constrained and the speed with which nodes are connected and topologies are formed and healed is a dominant concern. TSF is, as far as we know, the best algorithm with respect to these criteria.

References

[1] Specification of the Bluetooth System. `http://www.bluetooth.com/`, December 1999. Bluetooth Special Interest Group document.

[2] G. Tan and J. Guttag. A Locally Coordinated Scatternet Scheduling Algorithm. In *The 27th Annual IEEE Conference on Local Computer Networks (LCN)*, Tampa, FL, Nov. 2002.

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

[4] C. Law, A. K. Mehta, and K.-Y. Siu. Performance of a New Bluetooth Scatternet Formation Protocol. In *ACM Symposium on Mobile Ad Hoc Networking and Computing*, Long Beach, CA, October 2001.

[5] G. Zaruba, S. Basagni, and I. Chlamtac. Bluetrees-Scatternet Formation to Enable Bluetooth-Based Ad Hoc Networks. In *IEEE International Conference on Communications*, pages 273–277, 2001.

[6] S. Basagni and C. Petrioli. A Scatternet Formation Protocol for Ad hoc Networks of Bluetooth Devices. In *IEEE Vehicular Technology Conference*, pages 424–428, 2002.

[7] Z. Wang, R. Thomas, and Z. Haas. Bluenet - a New Scatternet Formation Scheme. In *Hawaii International Conference on System Science (HICSS-35)*, 2002.

[8] A. Racz, G. Milklos, F. Kubinszky, and A. Valko. A Pseudo Random Coordinated Scheduling Algorithm for Bluetooth Scatternets. In *ACM Symposium on Mobile Ad Hoc Networking and Computing*, Long Beach, CA, October 2001.

[9] N. Johansson, U. Korner, and L. Tassiulas. A Distributed Scheduling Algorithm for a Bluetooth Scatternet. In *Seventh International Teletraffic Conference*, Salvador da Bahia, Brazil, September 2001.

[10] D. B. Johnson and D. A. Maltz. *Mobile Computing*, chapter 5, pages 153–181. Kluwer Academic Publishers, 1996. Dynamic Source Routing in Ad Hoc Wireless Network.

[11] C. E. Perkins and E. M. Royer. Ad hoc On-Demand Distance Vector Routing. In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, New Orleans, LA, February 1999.

[12] ns-2 Network Simulator. `http://www.isi.edu/vint/nsnam/`.

[13] Bluetooth Extension for ns. `http://oss.software.ibm.com/developerworks/opensource/bluehoc/`.