Forming Scatternets from Bluetooth Personal Area Networks

Godfrey Tan, Allen Miu, John Guttag and Hari Balakrishnan
MIT Laboratory for Computer Science

Abstract—
There is increasing interest in wireless *ad hoc* networks built from portable devices equipped with short-range wireless network interfaces. 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 topology formation algorithm, called TSF (Tree Scatternet Formation) to build scatternets. TSF connects nodes in a tree structure that simplifies packet routing and scheduling. The design allows nodes to arrive and leave arbitrarily, incrementally building the topology and healing partitions when they occur. We present simulation results that show that TSF has low tree formation latency and also generates an efficient topology for forwarding packets.

I. INTRODUCTION

Bluetooth [1] is emerging as an important standard for short range, low-power wireless communication. It provides a decentralized communication substrate that standardizes the link-layer medium access (MAC) and physical layer functionalities of the traditional networking protocol stack [1], [2], [3]. It operates in the 2.4 GHz frequency band employing a pseudo-random frequency-hopping scheme.

The Bluetooth MAC protocol is designed to facilitate the construction of *ad hoc* networks without the need for manual configuration, cables, or wired infrastructure. It is based not on distributed contention resolution, as in traditional wireless LANs, but on a 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 slave slot (an odd-numbered slot, by convention) being used only by the slave to which the master sent a frame in the previous (even-numbered) transmission slot. This MAC protocol is an example of a *time-division duplex (TDD)* scheme.

Frequency hopping allows multiple concurrent Bluetooth communications within radio range of each other,

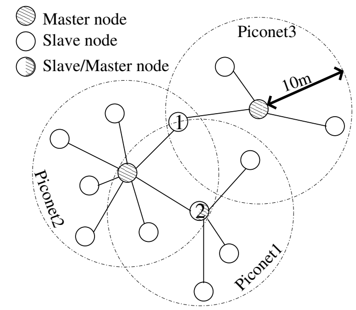

Figure 1. A Bluetooth scatternet with two types of relay nodes: node 1 is a "slave relay", while node 2 is a "master relay".

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. 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.

An obvious starting point is to judiciously choose nodes, such as nodes 1 and 2 in Figure 1, to participate as *relays* in multiple piconets, forwarding data between piconets. Since two slave nodes cannot be linked together directly, the path of a packet must alternate between master and slave nodes, until it reaches its final destination. While the basic idea is simple enough, a number of challenging problems need to be solved before this can become a reality.

We present an efficient topology formation algorithm, called TSF (for Tree Scatternet Formation), which assigns master/slave roles to nodes while connecting them in a tree structure. Our algorithm is both decentralized and self-healing, in that nodes can join and leave at any time without causing long disruptions in connectivity. It also 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. Furthermore, our scheme does not require any communication between nodes already in the scatternet, using only Bluetooth's lower-layer primitives for detecting potential nodes to form links with and establish communication links.

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

We chose a tree topology, in contrast to the approach proposed in [4], because it simplifies both the routing of messages and the scheduling of communication events. Routing is simplified 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 identifier (*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. This kind of approach could be more efficient than many traditional ad-hoc routing protocols [5], [6], [7], which either incur per-packet overhead as in Dynamic Source Routing (DSR) [8] or Routing Vector Method (RVM) [9], or increase memory requirements as in Ad-hoc On-Demand Distance Vector (AODV) [5].

A tree topology is effective in reducing the average communication latency between all node pairs for Bluetooth-like TDM networks. We show this in Section IV-D by defining the topology efficiency metric and evaluate the tree topology against various topologies. The intuition for why a tree topology is a reasonably efficient one is that it minimizes the total number of links and the number of average piconets per bridge node. Minimizing the total number of links in a topology reduces the potential for contention for transmission slots in the Bluetooth TDD scheme. Reducing the average piconets per bridge node avoids bridges becoming communication bottlenecks as they participate in multiple piconets on a time division basis. Our algorithm achieves the minimum number of average piconets per bridge node by ensuring that every bridge node participates in exactly two piconets.

In Section II, we explain the Bluetooth link formation process and prior work on scatternets. Section III describes the details of the TSF algorithm. We evaluate the performance TSF and compare it to another scheme in Section IV, and offer our conclusions in Section V.

II. Background

In this section, we provide background information about some aspects of Bluetooth. We start by describing how two nodes establish a bi-directional communications link. An understanding of this link formation process, which is part of the Bluetooth specification, is necessary to understand topology formation algorithm. We then discuss a probabilistic topology formation scheme, which we used as a benchmark for evaluating our scheme.

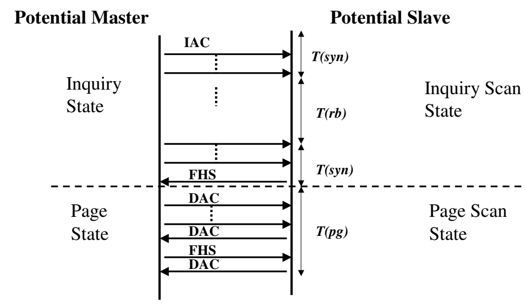

Figure 2. Bluetooth link formation process.

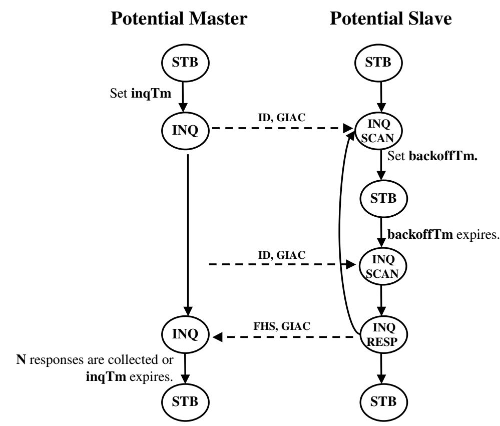

Figure 3. State transitions during the Inquiry process.

A. 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) to allow it to establish a frequency hopping connection with a subset of those neighbors. The goal of the Page process is to use the information gathered in 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. 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. The Bluetooth specification states that a node in the INQUIRY state devotes sufficient amount of time transmitting and listening whereas a node periodically enters the INQUIRY SCAN state to scan continuously over a short window.

During the Inquiry process, all nodes hop over 32 dedicated frequencies.[2] Of course, the inquiring node and the scanning node could be out of phase since the phase of each is determined by its local clock. To facilitate proper frequency synchronization within a reasonable amount of time, the Bluetooth Baseband specification requires that the INQUIRY node hops at a much faster rate than the INQUIRY SCAN node.

Multiple INQUIRY SCAN nodes can simultaneously receive messages from the same INQUIRY node. To avoid contention, each scanning node chooses a random backoff interval, T_{bo}, between 0 and 1023 time slots before responding with the signaling information. If T_{sync} is the delay before two nodes can synchronize their frequencies during the Inquiry process, the time taken to complete the Inquiry process is given by:

$$T_{inq} = 2T_{sync} + T_{bo} \tag{1}$$

A node remains in 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.[3] 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. Figures 3 and 4 illustrate the state transitions during the Inquiry and Page processes respectively.

The Page process is similar to the Inquiry process except that the paging device already knows the estimated clock value and BD_ADDR of the paged device. However, there will still be some synchronization delay before the pager and the paged devices can communicate. We define T_{pg} as the time taken to complete the Page process. It is worth while to note that it will be most efficient for the two nodes in the Inquiry process to enter the Page process as soon as the inquiring node has received the inquiry response. Thus, the total time taken to establish a link between two nodes is:

$$T_{conn} = T_{inq} + T_{pg} \tag{2}$$

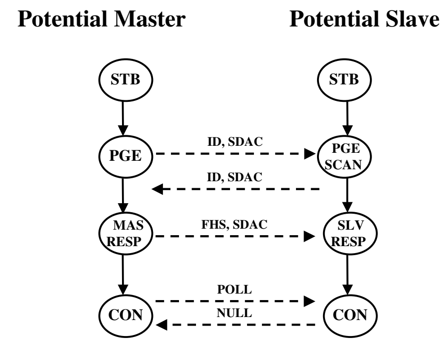

Figure 4. State transitions during the Page process.

T_{inq} is typically much larger than T_{pg} and dominates the delay to enter the CONNECTION state.[4]

B. Scatternet Formation

A topology construction protocol is needed to form piconets and interconnect them via bridges. There exists an extensive literature on distributed protocols for self-configuring networks [10], [11], [12]. Little of it, however, deals with the complications introduced by the master-slave frequency hopping TDD MAC layer used in Bluetooth.

The Bluetooth specification assumes that each node knows whether it is to be a master or a slave. The need for manual configuration of master or slave roles is unattractive when more than a few nodes are attempting to form a connected scatternet in an *ad hoc* fashion. To deal with this problem, the Bluetooth specification provides a Host Controller Interface (HCI) specification that provides a standardized method of accessing the Bluetooth baseband capabilities. This interface can be used to implement various topology formation schemes.

Salonidis *et al.* present a symmetric link formation scheme where no configuration of potential master or slave roles is necessary [4]. In their scheme, every node wishing to establish links with other nodes alternates between the INQUIRY and INQUIRY SCAN states continuously and attempts to connect with another node which is in a different state. The state residence time is randomized. The scheme uses an election process to elect a leader to configure a particular scatternet topology. The scheme is limited to scenarios where all nodes arrive over a small window and are within radio proximity of each other. It does not take into account for scenarios where nodes in the scat-

[2] The number of frequencies used during the inquiry or page process is 32 in Europe and US and 16 in other countries such as Japan.

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

[4] T_{inq} is in the order of seconds whereas T_{pg} is in the order of milliseconds if both nodes in the Inquiry process enter the Page process immediately after the inquiry response is received.

ternet may arbitrarily disappear due to mobility or other constraints such as drained batteries. This scheme also currently limits the maximum number of nodes involved in the scatternet formation to be 36. The authors show that the performance of their scheme under such constraints is reasonably good.

In later sections, we compare our topology formation scheme to a probabilistic scheme. The probabilistic scheme follows straightforwardly from the Bluetooth baseband specification [1], which specifies recommended timer values for potential masters and slaves. When a node comes online, it configures itself as a potential master with a probability of P_m. A potential master node stays in the INQUIRY state constantly sending out inquires for the neighboring nodes and attempts to establish links with a maximum of N_s potential slaves. A potential slave periodically enters the INQUIRY SCAN and the PAGE SCAN states and establish links with any master node. Since master nodes always stay in the INQUIRY state, it generally follows that slave nodes become bridges between multiple piconets [5]. As time goes on, new links continue to form. P_m and N_s govern the connectivity of the topology and its efficiency. In Section IV, we discuss the performance of this probabilistic scheme as a function of the parameters P_m and N_s.

III. TSF: Tree Scatternet Formation

Bluetooth-like link technologies are a recent development, and one can only speculate on how they might be networked together and used. Broadly speaking, there are two distinct environments in which Bluetooth-based scatternets will be used. In some environments, it will be reasonable to statically configure scatternets in the way many wired (and wireless) networks are configured today. In many other environments, the relatively frequent arrival and departure of nodes and node mobility will make manual configuration problematic. These are the environments of interest to us.

Within these environments, one can envision two usage modes. In the first mode, most (or all) nodes arrive *en masse*, such as in a scheduled meeting with several participants equipped with Bluetooth devices. In the second mode, nodes arrive and leave in incremental fashion, such that at any time there is a "core" operating network that a new node should join. This situation would arise in a deployment with several access points and a combination of static and mobile (or battery-operated) devices. Our goal is to efficiently construct topologies for both these modes of operation.

This section presents and proves the correctness of *TSF*, a tree scatternet formation algorithm that has the following properties that meet the requirements of our operating environment.

1. Connectivity: TSF constantly attempts to converge to 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 (a *forest*), 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 on *en masse*, and nodes departing incrementally or *en masse*, avoiding loops and healing network partitions.
3. Communication efficiency. TSF produces topologies where the average node-node latency 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.

A. Protocol

At any point in time, the TSF-generated scatternet is a forest consisting of c connected tree components $\{T_1, T_2, \ldots, T_c\}$. Some of these trees are single nodes (also called *free nodes*) that are seeking to join another tree to form a larger component and reduce the number of components. Each tree is rooted; we denote the root of tree T_k by r_k.

TSF is distributed with each node operating autonomously with only local communication. Each node in the network runs the same state-machine algorithm, transitioning between two states: FORM, which consists of two sub-states FORM: INQUIRY and FORM:INQUIRY-SCAN, and COMM. In the FORM state, the node attempts to rendezvous with another node belonging to a different tree, to form a Bluetooth communication link and thereby improve the connectedness of the scatternet. In the COMM state, the node is involved in data communication with other nodes in its connected component and not in scatternet formation. This division of states is necessary because Bluetooth is a frequency-hopped and time-slotted system.

The pseudo-code for the state-machine running at each node is shown below.

[5] For simplicity, we limit the maximum number of piconets a slave participates in to 8.

```
PROCEDURE TSF() {
    do forever {
        state ← OPPOSITE(state)
        t_form ← random(E[T_inq], D)
        Remain in "state" for time t_form
        if (root) {
            state ← OPPOSITE(state)
            t_form ← random(E[T_inq], D)
            Remain in "state" for time t_form
        }
        t_comm ← f_comm × random(E[T_inq], D)
        if(t_comm) state ← COMM
        Remain in "state" for time t_comm
    }
}

PROCEDURE OPPOSITE(STATE) {
    if (state == FORM:INQUIRY)
        state ← FORM:INQUIRY-SCAN
    else
        state ← FORM:INQUIRY
    return state
}
```

The FORM state is used by nodes to reduce the number of partitioned scatternet components. It consists of two sub-states, FORM:INQUIRY and FORM:INQUIRY-SCAN, which correspond to the Bluetooth-specified states that allow two nodes to rendezvous and then establish a communication link. While all nodes spend time in this state, the roots of each tree in the forest play a special role and spend more time in this state than the other nodes in the tree.

TSF has two parameters in the FORM state, $E[T_{inq}]$ and D. $E[T_{inq}]$ is the time taken to complete the Inquiry process, given by Equation 1. D is a parameter deciding the size of the random interval, which governs how long the node is resident in a given state. We analytically derived optimal value for D and also ran experiments to verify that value.

The time spent in the COMM state is a function of f_{comm}, which in turn is a function of how busy a node is likely to be in performing its communication tasks. Clearly, if the node is a free node, f_{comm} must be 0, since it cannot be involved in any communication. In this case, the node spends all of its time in FORM, attempting to join a scatternet. In contrast, the bigger the tree, it is important for a node to spend more of its time involved in communication. However, it is also important for each node in the tree to play a part in forming bigger trees and improve the overall connectivity of the scatternet.

We find that a choice of f_{comm} as a function of the age of the node (in terms of how long ago it entered the scatternet), and in proportion to the node's number of children in the current tree, can produce efficient communication topologies where the average path length is short. The intuition behind the age term is that if a node has only recently joined, it is worthwhile making it spend more of its time trying to form a bigger scatternet, relative to an older node that may be involved in, and essential for, efficient data communication. The intuition behind using the number of children is that the larger this number, the more likely it is to be involved in communication.

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 associates a special role for the root of each component tree: Only root nodes can attempt to heal partitions and join another tree as a slave. As shown in the pseudocode above, root nodes spend roughly double the amount of time as non-root nodes in the FORM state, to account for their performing the task of healing their current tree with another to form a bigger tree. In contrast, non-root nodes play a role in helping free nodes join the scatternet (and so need to spend some time in the FORM state), but do not need to spend as much time as a root node because they are not involved in healing operations to form bigger connected trees (and also becaue there are multiple non-root nodes in any tree of more than two nodes).

The rest of this section describes the FORM and COMM states in more detail and proves some properties of TSF. The next sub-section shows how to implement TSF using Bluetooth primitives.

A.1 FORM state

In the FORM state, a root node transitions to the "opposite" of its current state and spends a random period of time there performing the task corresponding to either the Bluetooth INQUIRY or Bluetooth INQUIRY SCAN mode. It then transitions to the other FORM sub-state and spends a random interval of time there. Notice that a free node that has no other children in its tree has no COMM state, and therefore simply alternates between the two FORM sub-states. This idea is motivated by a suggestion in [4].

When a root node successfully receives an inquiry response from another (root) node, the two nodes immediately enter the PAGE and PAGE SCAN states, and attempt to establish a connection. After a link connection is established the master node becomes the root node and the slave becomes a leaf node forming a larger tree and reducing the number of component trees in the forest.

When a root node joins another node as a child, the child

Mas/Slave	Root	Non-root	Free
Root	1	0	0
Non-root	0	0	1
Free	0	1	1

TABLE I
LINK FORMATION COMBINATION: ENTRIES WITH 0 ARE INVALID.

is made the slave and the parent node the master of the Bluetooth piconet. The parent then serves as a relay and forwards packets to the subtree rooted at the erstwhile root. We use this master-relay strategy because it is simple and easy to reason about, and because it minimizes the number of piconets in which a relay node participates (at most two, the minimum possible) and therefore minimizes the scheduling and piconet-switching overhead, both of which are significant in Bluetooth.

TSF uses three rules to form bigger trees while avoiding loops:

1. Free nodes may only connect to other free nodes, or to non-root nodes. In the first case, one of the nodes becomes master and the other the slave of the newly formed Bluetooth piconet; in the second case, the erstwhile free node becomes the slave.
2. Root nodes of trees with more than one node may only connect to other root nodes. One of the erstwhile root nodes becomes the master and the other the slave for the newly formed Bluetooth piconet.
3. Non-root nodes do not attempt to form larger trees with nodes that are not free nodes.

Theorem 1: TSF produces loop-free topologies.

Proof: By induction on the number of nodes n in the scatternet. For $n \leq 2$, this is clearly true (Rule 1). Suppose it is true for all trees of size $< n_0$; consider two trees T_1 and T_2, of sizes n_1 and n_2, both smaller than n_0. The number of links in tree T_i is $n_i - 1$, by definition.

Without loss of generality, suppose T_1's root r_1 attempts to join T_2 as a slave. If T_1 is a free node, then it links with a non-root node in T_2 and forms a tree of size $n_2 + 1$, without loops (Rule 1). If T_1 has more than one node in it, then r_1 links with r_2 and produces a new connected graph with $n_1 + n_2$ nodes with $(n_1 - 1) + (n_2 - 1) + 1 = n_1 + n_2 - 1$ links, which must be a loop-free tree (Rule 2). Rule 3 ensures that loops are avoided since only r_1 in T_1 can merge with another non-trivial tree. ■

TSF can be visualized as various free nodes joining existing trees (or other free nodes) in the scatternet, while root nodes attempt to merge together to eventually form a single connected scatternet. Table I shows the valid combination of master-slave connection establishment between different types of nodes.

We do not allow the connection between non-root nodes and root nodes since this has the potential to create self-loops or multi-hop loops. Of course, it would be possible to allow the connection and check for loops, but doing so would involved a significant amount of communication within the scatternet, which has high overhead. In fact, TSF produces trees without any communication between nodes already in the scatternet, and is well-suited to a Bluetooth implementation as explained in Section III-B.

We also note that making free nodes children of root nodes of trees that are not themselves free nodes cannot create loops. However, TSF precludes this possibility, to save links of root nodes for merging with other trees. We find that this partitioning of functionality, where the root node is involved with merging with other non-trivial trees, and the non-root nodes help free nodes join the scatternet, works well.

The FORM state is characterized by the amount of time spent alternating between the FORM:INQUIRY and FORM:INQUIRY-SCAN sub-states. To avoid periodic synchronization effects, TSF picks a time from a random interval for this, given by:

$$t_{form} = random(E[T_{inq}], D) \tag{3}$$

It is clear that this time must at least be as long as $E[T_{inq}]$ to ensure enough time for a successful handshake. D should be 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.

A.2 COMM State

In the COMM state, a node spends a period of time given by:

$$t_{comm} = f_{comm} \times random(R, D) \tag{4}$$

The value of f_{comm} depends on whether the node is a free node, on the age of the node, and on the number of adjacent links, d.

$$f_{comm} = \begin{cases} 0 & \text{if free node} \\ d & \text{if not free \& age} < \text{threshold} \\ Ad & \text{if not free \& age} > \text{threshold; A>1} \end{cases} \tag{5}$$

A.3 Per-node state

Clearly, TSF needs very little per-node state information. In fact, only two bits of information is necessary so that a node knows which type of node it is. Figure 5 shows the transitions between different node types based on a new link creation. When links are torn down, each node updates the information in a similar fashion.

A.4 Healing

Self-healing is an important requirement for a topology formation scheme, especially in networks in which many nodes are energy-constrained 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 master node loses the connection to a slave node, and when a slave loses the connection to its master. When a master detects the loss of a child, it does not need to do anything except decide if it has become a free node. When a slave loses the connectivity to its parent, it updates its node type and sets age (see Equation 5) to zero. A leaf node in this situation becomes a free node and an internal node becomes a root node.

An important detail concerns the Bluetooth limitations on the maximum number of links. In a situation where multiple nodes arrive at roughly the same time, several communication links will be established simultaneously resulting in many network components. Currently, our scheme only allows root nodes to merge together to produce a single connected scatternet tree. This simplifies the protocol for avoiding loops. However, a master node in Bluetooth piconet can only have a maximum of 7 slaves. Thus, there could be situations where all the root nodes may not be able to merge together as all of them have already had the maximum number of children.

To avoid this case, when a root node is about to reach a maximum number of children, it designates a child to become the root and the two nodes switch roles as master and slave. We have not experienced this particular situation in any simulation simulations involving 100 or fewer nodes. There are three reasons for this. First, as the size of the scatternet increases, newly arrived free nodes will be most likely to attach to an existing tree immediately instead of forming a separate sub-tree with other free nodes. Second, by putting the number of adjacent links into consideration in Equation 5, TSF preferentially induces mutliple smaller (in terms of degree) sub-trees to merge together before eventually merging with the largest tree. Finally, when the two root nodes merge, the root node assuming

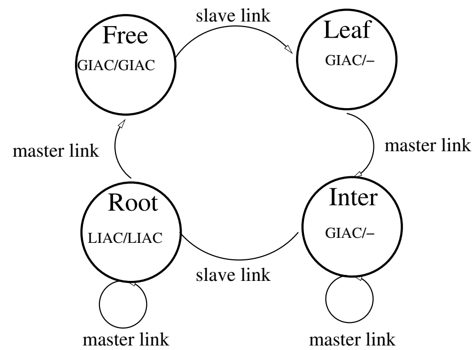

Figure 5. Node state transitions during topology construction. IACs used to transmit and listen during the Inquiry process are separated by /.

the master role becomes the parent, and thus, it is unlikely that a particular root node will exhaust its links since this will require that root node to always assume the role of a master.

B. Bluetooth Implementation

To implement TSF in Bluetooth, nodes need to know the kind of node with which they are about to establish a link. This information can be exchanged once two nodes have already established a link, and based on that they can decide to either break the link or continue. Obviously, this is inefficient. Fortunately, the Bluetooth specification allocates 64 Dedicated Inquiry Access Codes (IAC) to be used during the Inquiry process. Currently only the Generic Inquiry Access Code (GIAC) and the Limited Inquiry Access Code (LIAC) are defined. The Bluetooth HCI specification allows nodes in the INQUIRY SCAN state to filter certain types of IAC or listen to a particular list of IAC. In our scheme, we use both GIAC and LIAC. To isolate the communication between root nodes, roots only transmit and listen to ID packets containing LIAC. All other nodes transmit ID packets with GIAC and never listen to ID packets with LIAC. This prevents nodes from attempting to establish unwarranted connections and significantly improves the efficiency of the protocol. Figure 5 shows the IAC transmitted and listened to by each node type.

There is one circumstance under which two nodes might attempt to form a connection that would lead to a loop. This happens because a node in the INQUIRY state does not know whether an inquiry response (FHS packet) is in response to that node's inquiry. Consider two root nodes A and B which are in the INQUIRY and INQUIRY SCAN states respectively. After receiving ID packets from A, B responds to A with an FHS packet. However, suppose a non-root node, C, from the tree rooted at B is also in the INQUIRY state and accidentally receives the FHS packet from B. Since C has no way knowing that B is a root rather than a free node, both A and C will attempt to page B which has entered the PAGE SCAN state as described in Section II-A. If C is successful before A in establishing a link with B, it will produce a cycle. This problem can be easily avoided by including one extra bit of information, stating whether the node sending the response is a root node or not. The FHS packet does have two reserved bits, but these are not accessible through HCI commands. Because we want our scheme to work with the current HCI specification, we have decided not to use this approach. Instead, our scheme requires the parent node to send a single slot packet to a new child node including information about the type of the parent node after a connection is established. If the child node is not a free node and both nodes are not root nodes, the child will tear down the link by sending appropriate HCI commands to the Baseband module.

IV. Performance evaluation

We implemented our algorithms in the ns-2 [13] network simulator using a Bluetooth extension module for ns developed at IBM [14]. We conducted several simulations to evaluate the performance of our algorithms. This section presents our results on link establishment latency, scatternet formation latency, and communication efficiency in terms of latency of the scatternet topologies under different conditions.

A. Configurations

In all the experiments, nodes arrive uniformly over a 15 seconds window. The simulation is run until a steady state is reached. Every data point shown in the figures is the average of 10 runs. We compare TSF to several configurations of the probabilistic scheme described in II with various values of P_m and N_s. Recall that P_m is the probability with which a node configures itself as a master and N_s is the maximum number of slaves with which a master node attempts to establish communication links. For clarity, we choose to show a particular configuration of P_m and N_s where $P_m = 0.5$ and $N_s = 5$. We will refer to this scheme as PROB. TSF is configured so that $threshold$ specified in Equation 5 is larger than the simulation run.

B. Link Establishment

In this section, we analyze the performance of the connection setup delay which is defined as the time taken before a free node can establish its first communication link. This is an important metric because it gives a sense of how fast a node can, on average, talk to its first neighbor. Figure 6 shows the average connection setup delay of the PROB and the TSF schemes as a function of the number of nodes arrived. TSF achieves achieves an average connection setup delay about 3 seconds regardless of the number of nodes, and clearly outperforms PROB.

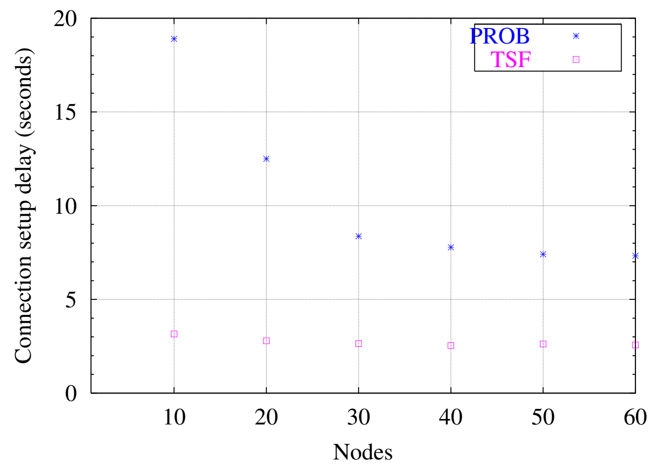

Figure 6. Average connection setup delay for the TSF and PROB schemes.

C. Scatternet Formation

As proven in Section III, TSF attempts to monotonically reduce the number of trees and to converge to a topology with a single connected scatternet when nodes are in radio range. In contrast, PROB may not converge to a single scatternet at all since master nodes may run out of available links. Figures 7 illustrates how long it takes to form a connected scatternet for both PROB and TSF. We eliminate many trials where PROB cannot produce a connected scatternet.

The performance is comparable for scenarios involving less than 40 nodes. However, the delay for TSF significantly increases when the number of nodes is 50 or larger. The reason for that is as the number of nodes increases, the average degree of root nodes increases and thus, it takes longer for roots to merge together (Recall that the time a node spends staying in the FORM state depends on the degree.)

In return for the longer setup time, TSF yields a far simpler topology. Figure 8 and 9 show the scatternet topologies produces by PROB and TSF for a 50-nodes scenario respectively.

D. Topology Efficiency

The topology of a Bluetooth scatternet affects the overall network capacity and average latency between any two nodes. The efficiency of a topology can be defined using a variety of metrics, e.g., througput, goodput and la-

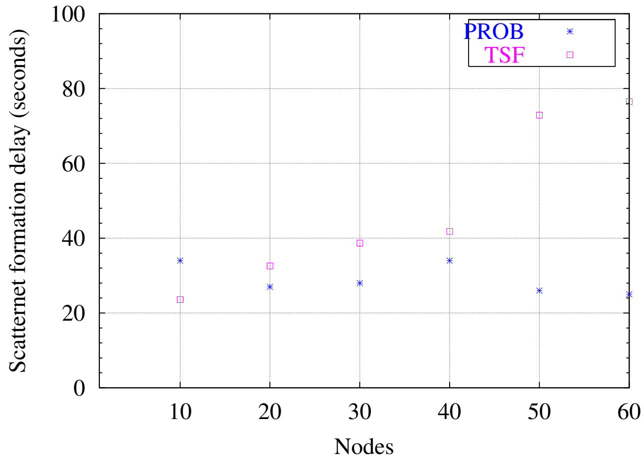

Figure 7. Scatternet formation delay as a function of nodes.

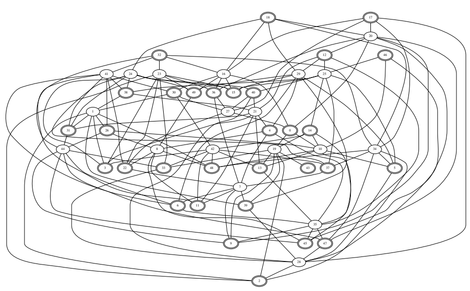

Figure 8. A 50-node scatternet created with PROB.

tency. We choose communication latency as an important metric to determine the efficeincy of Bluetooth scatternets made up of low-bandwidth links. In the following subsections, we define a metric to measure the average path latency between node pairs and evaluate the performance of the topologies generated by PROB and TSF schemes using that metric.

D.1 Efficiency Metric

The communication latency between two nodes in the scatternet is governed largely by three factors: i) hop count, ii) intra-piconet scheduling delay and ii) inter-piconet bridging delay. Clearly, the values of each component vary based on the scheduling and routing policies.

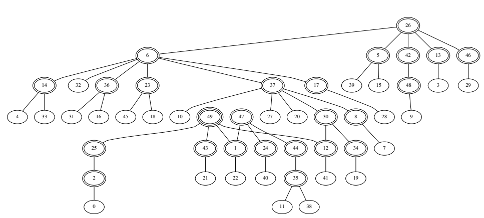

Figure 9. A 50-node scatternet created with TSF.

There is no generally accepted scheduling scheme for scatternets. Moreover, since there are relatively few deployed Bluetooth networks, finding representative and realistic traffic patterns for performance evaluation is difficult, if not impossible.

In light of this, we evaluate communication latency using a new model that approximates the efficiency of a scatternet topology in a way that is independent of scheduling algorithms and traffic patterns. In particular, we present a way to approximate the average path latency L between all pairs of source and destinations on the given scatternet, T_s. Let V and E be the set of nodes and edges in the topology T_s. The average latency between nodes is:

$$L = \frac{1}{|V| \cdot (|V| - 1)} \sum_{s,d \in V} l(s,d) \tag{6}$$

where $l(s,d)$ is the average path latency between s and d. Let $E_{(s,d)} \subseteq E$ be the set of edges in the path between (s,d) defined by the *routing* topology, T_r. Then $l(s,d)$ is the sum of the link latencies in the path $P(s,d)$:

$$l(s,d) = \sum_{(u,v) \in E_{(s,d)}} t(u,v) \tag{7}$$

Because the link latency between any two neighboring nodes depends on intra-piconet and inter-piconet scheduling, we use the expected link latency $t(u,v)$. We define the expected link latency to be the sum of two components l_{intra} and l_{inter}, which are the expected latencies contributed by intra-piconet and inter-piconet scheduling respectively. To find l_{intra}, observe that Bluetooth transmissions always take place between a master and a slave. Thus, one of the u or v nodes must be a master and the other must be the slave, and the intra-piconet latency is governed by the master's schedule, which depends on its number of slaves. Let $m(u,v)$ denote the master node of the link (u,v), and $N_s(m(u,v))$ be the number of slaves in the piconet of which $m(u,v)$ is the master. Then we make l_{intra} independent of the master's schedule by assuming that $m(u,v)$ will schedule every link with an average period of $\alpha N_s(m(u,v))$, where α is the average transmission time alotted to a single link. Assuming that a packet arrives during this period with uniform probability, the *average* intra-piconet latency, l_{intra}, is:

$$l_{intra} = \frac{1}{2} \cdot \alpha N_s(m(u,v)) \tag{8}$$

Next, we find l_{inter} by observing that a relay node (regardless of whether it is a master or a slave) spends some amount of time in each piconet for which it acts as a relay.

For a given link (u, v), either u or v or both can be relaying nodes[6], but the transmission on this link can take place only when both nodes are switched to the same frequency hopping sequence. Let β be the average time spent by a relaying node in one piconet and $N_m(a)$ be the number of piconets of which a relay node a is a member. Then we define the inter-piconet delay by assuming that the relay node a schedules a given hopping sequence with an average period of $\beta N_m(a)$. In the case when both u and v are relay nodes, the piconet switching periodicity may be different. For simplicity, we make the conservative assumption that the two relay nodes will always meet in the same piconet on the larger of the two inter-piconet scheduling periods. Again, let us assume that a packet arrives anytime during this period with uniform probability. The *average* inter-piconet latency, l_{inter} becomes:

$$l_{inter} = \frac{1}{2} \cdot \beta \max(N_m(u), N_m(v)) \tag{9}$$

For simplicity, let β_α be the normalized value of β in terms of α. Then we combine the two latency components to obtain an expression for the expected link latency between node u, v:

$$l(u, v) = \frac{1}{2} \left[N_s(m(u, v)) + \beta_\alpha \max(N_m(u), N_m(v)) \right] \tag{10}$$

From this expression, observe that the weight of the edges of a given scatternet topology T_s is a function of the degree of the endpoints at each edge. *This presents an interesting tradeoff between increasing connectivity of a topology, which reduces the average number of hop counts between any two pairs of nodes, and reducing connectivity which reduces the expected latency at each hop.*

D.2 Results

In this section, we compare the topology efficiency between the topologies generated by TSF and PROB. The PROB algorithm generates a graph topology with many more links than a TSF tree topology so the average hop count between any two nodes on the PROB topology will be lower. Thus, by comparing the efficiency of TSF and PROB topologies, we show the latency tradeoff between reducing the average hop count and increasing the expected link latencies. The results in the next section show that despite the smaller number of links the average path latency of the tree topologies generated by TSF is comparable to the average path latency of the graph topologies generated by PROB.

We use the definition of average path latency, L, from the previous section to evaluate and compare the topology efficiency of the scatternets generated by TSF and PROB using the schemes mentioned in Section IV-A. We observe that L depends on the routing alogorithm used to carry traffic between any two nodes on the graph. For tree topologies generated by TSF, there is only one route whereas for PROB topologies generated by PROB, there are many. To find the best efficiency measurement for PROB topologies, we use the all pairs shortest-path routing topology, which uses path latency as the distance metric.

Figure 10 and 11 shows the average path latency of scatternets formed by the TSF and PROB as a function of network size. The average path latency is normalized to α, the average transmission time alotted to a link. Each point on the graph represents a value averaged over 10 different topologies of the same size generated by each of the algorithms. The different curves represents the average path latencies calculated by setting β_α to 1, 3.5, and 7.

Surprisingly, the TSF tree topologies have lower average path latency than the PROB graph topologies for all network sizes and all values of β_α. Furthermore, as β_α increases, the average path latencies for the PROB graph topologies grow much higher than TSF topologies.

We attribute these result to the cost of inter-piconet scheduling. For TSF topologies, the relay nodes belong to exactly 2 piconets. For PROB topologies, the relay nodes belong to 2 or more piconets, which increases the inter-piconet scheduling latency. Figure 12 illustrates the effect of increasing the inter-piconet scheduling penalty, β_α, for a scatternet with a fixed size of 50. The TSF topologies are clearly less sensitive to inter-piconet scheduling delay than are the PROB topologies.

V. SUMMARY

This paper described TSF, a scatternet formation algorithm for networks constructed of devices communicating using Bluetooth. TSF efficiently connects nodes in a tree structure that simplifies packet routing and scheduling. Unlike earlier work, our design does not require that all devices be within radio range of each other, nor does it restrict the number of nodes in the network. It also allows nodes to arrive and leave at arbitrary times, incrementally building the topology and healing partitions when they occur.

Our simulation results show that TSF has low tree formation latency. The average connection delay, three seconds, is independent of the number of nodes.

[6] When both u and v are relay nodes, we assume that they define only one unidirectional master-slave relationship. That is, u and v communicate with each other only in one piconet, and they do not later exchange the master-slave relationship to form another piconet in which they communicate.

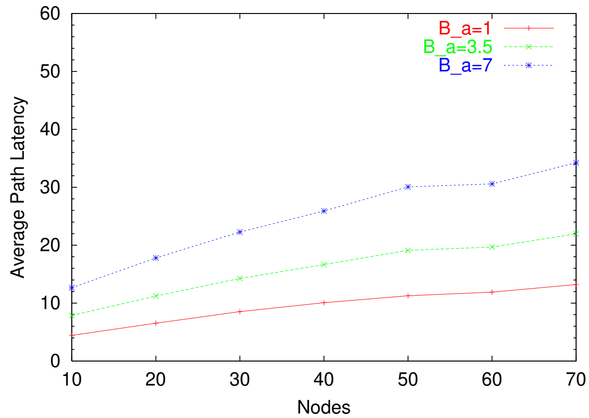

Figure 10. Average path latency of scatternets formed by TSF vs network size.

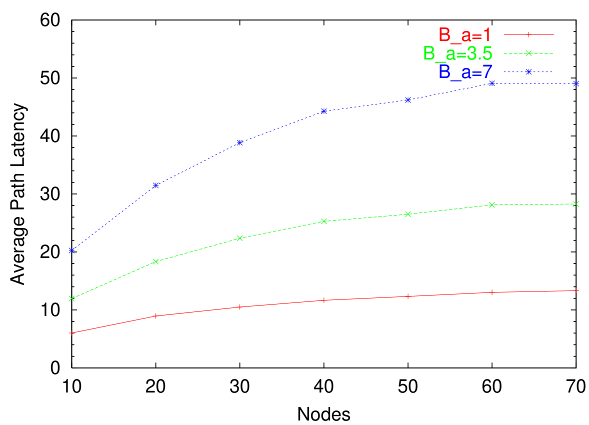

Figure 11. Average path latency of scatternets formed by PROB vs network size.

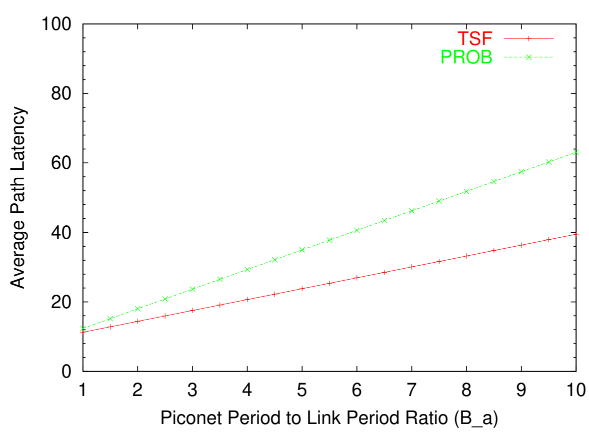

Figure 12.

We also present a model for analyzing the efficiency of Bluetooth scatternet topologies. The model takes into account intra-piconet and inter-piconet scheduling overhead. Using this model we show that TSF yields efficient topologies, i.e., the communication latency between nodes in the scatternet is low.

References

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

[2] Jaap Haartsen, "The Bluetooth Radio System," *IEEE Personal Communications Magazine*, pp. 28–36, February 2000.

[3] Jennifer Bray and Charles Sturman, *Connection Without Cables*, Prentice Hall, 2001.

[4] Theodoros Salonidis, Pravin Bhagwat, Leandros Tassiulas, and Richard LaMaire, "Distributed topology construction of Bluetooth personal area networks," in *Proc. IEEE INFOCOM*, Anchorage, AK, April 2001.

[5] Charles E. Perkins and Elizabeth M. Royer, "Ad hoc On-Demand Distance Vector Routing," in *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, New Orleans, LA, February 1999, pp. 90–100.

[6] Elizabeth M. Royer and C.-K. Toh, "A Review of Current Routing Protocols for Ad-Hoc Mobile Wireless Networks," *IEEE Personal Communications Magazine*, pp. 46–55, April 1999.

[7] Josh Broch, David A. Maltz, David Johnson, Yih-Chun Hu, and Jorjeta Jetcheva, "A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols," in *Proceedings of the Conference on Mobile Computing and Networking*, Dallas, TX, 1998.

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

[9] Pravin Bhagwat and Adrian Segall, "A routing vector method (RVM) for routing in bluetooth scatternets," in *6th IEEE International Workshop on Mobile Multimedia Communications (MOMUC)*, San Diego, CA, November 1999.

[10] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-Efficient Routing Protocols for Wireless Microsensor Networks," in *Proc. 33rd Hawaii Int'l Conf. on System Sciences (HICSS '00)*, Jan. 2000.

[11] Benjie Chen, Kyle Jamieson, Hari Balakrishnan, and Robert Morris, "Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks," in *Proc. of the 7th ACM Int'l Conf. on Mobile Computing and Networking*, Rome, Italy, July 2001, pp. 85–96.

[12] Nancy A. Lynch, *Distributed Algorithms*, Morgan Kaufmann Publishers, Inc., 1996.

[13] "ns-2 Network Simulator," `http://www.isi.edu/vint/nsnam/`, 2000.

[14] "Bluetooth Extension for ns," `http://oss.software.ibm.com/developerworks/opensource/bluehoc/`, 2001.