Self-organizing Bluetooth Scatternets

by

Godfrey Tan

Bachelor of Science in Electrical Engineering and Computer Science,
University of California at Berkeley (1999)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

January 2002

© Massachusetts Institute of Technology 2002. All rights reserved.

Author . . . .
Department of Electrical Engineering and Computer Science
January 18, 2002

Certified by . . . .
John Guttag
Professor
Thesis Supervisor

Accepted by . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Self-organizing Bluetooth Scatternets
by
Godfrey Tan

Submitted to the Department of Electrical Engineering and Computer Science
on January 18, 2002, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

There is increasing interest in wireless *ad hoc* networks built from portable devices equipped with short-range wireless network interfaces. This thesis 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 develop a set of online algorithms to form scatternets and to schedule point-to-point communication links. Our efficient online topology formation algorithm, called TSF (Tree Scatternet Formation), builds scatternets by connecting nodes into a tree structure that simplifies packet routing and scheduling. Unlike earlier works, our design does not restrict the number of nodes in the scatternet, and also allows nodes to arrive and leave at arbitrary times, incrementally building the topology and healing partitions when they occur. We have developed a Bluetooth simulator in *ns* which includes most aspects of the entire Bluetooth protocol stack. It was used to derive simulation results that show that TSF has low latencies in link establishment, tree formation and partition healing. All of these 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. Our scheduling algorithm, called TSS (Tree Scatternet Scheduling), takes advantage of the tree structure of the scatternets constructed by TSF. Unlike previous works, TSS coordinates one-hop neighbors effectively to increase the overall performance of the scatternet. In addition, TSS is robust and responsive to network conditions, adapting the inter-piconet link schedule effectively based on varying workload conditions. We demonstrate that TSS has good performance on throughput and latency under various traffic loads.

Thesis Supervisor: John Guttag
Title: Professor

Acknowledgments

This thesis would not have been materialized without constant supervision from my adviser, John Guttag. It has been an incredible experience and an invaluable opportunity to work with John. John has been a mentor, tutor and teacher: always providing me with mental support and thoughtful advice, helping me improve my writing, presentation and technical skills with weekly one-on-one sessions, and showing me, at every chance, how to reason about research issues and solving them effectively. Another individual critical in helping me shape the research theme of this thesis is Hari Balakrishnan. After all, the thesis research grew from a class project while I was taking the graduate networking course taught by Hari during Fall, 2000. Hari has treated me as one of his own students, always giving me sound advice and support. Both John and Hari have put great efforts in turning parts of the thesis work into conference paper submissions. I feel fortunate to be a member of the Network and Mobile Systems (NMS) research group, led by John and Hari, at the Laboratory for Computer Science (LCS). My close friend and colleague, Allen Miu, has been an ideal collaborator as we worked together on several issues in completing our paper submissions. It wouldn't have been as much fun working at LCS without Allen. Many of my friends in the NMS group, Magdalena Balazinska, Kyle Jamieson, Greg Harfst and Bodhi Priyantha, have also participated in several exciting discussions that usually led me to deeper, if not complete, understanding of the problems at hand. Lastly, regular warm love and support from my parents, brothers and sister, living in California, and my fiancee‘ have given me the strength needed to complete this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The growing use of information intensive consumer devices such as cell phones, personal digital assistants (PDAs), and laptop computers have called for a new networking paradigm for interconnecting them. The goal is to create a personal area network (PAN) that accommodates seamless information transfer between different devices with varying capacity in an *ad hoc* manner without the need for manual configuration or wired infrastructure. In December, 1999, an industry consortium known as the Bluetooth Special Interest Group (SIG) standardized Version 1 [1] of a short-range, low-power radio frequency (RF) technology called Bluetooth motivated in part by the need for suitable link-layer PAN technologies [8, 16, 9]. The principal design goals of Bluetooth were i) *ad hoc* connectivity, ii) robust communication, iii) simplicity, iv) energy efficiency and v) cost efficiency. The Bluetooth communication substrate consisting of Radio, Baseband, Link Controller and Link Manager specifies mechanisms for establishing connection with nearby devices from different manufactures in an *ad hoc* manner. Up to 8 Bluetooth devices can form a centralized network, called *piconet*, controlled by a master node which allocates transmission slots to all other nodes (slaves) in the piconet. Bluetooth achieves robustness against interference from nearby devices by employing a Frequency Hopping Code Division Multiple Access (FHCDMA) technique. 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. This thesis addresses challenges related to realizing self-organizing scatternets and present a novel set of solutions that can be implemented within the existing Bluetooth specification.

The communication substrate of Bluetooth is different from that of other existing wireless technologies. Unlike traditional wireless LANs which rely on distributed contention resolution mechanisms, Bluetooth is based on a centralized master-slave scheme. A Bluetooth piconet consists of one master and up to seven slaves. The master allocates transmission slots (and therefore, channel bandwidth) to the slaves

[1]SIG published an updated Version 1.1 in February 2001.

in the piconet. The master transmits in every even-numbered time slot of $625\mu s$. A slave transmits data only in an odd-numbered slot, and only if it receives a message (which might simply be an empty "poll" message) from the master in the previous (even) slot. This MAC protocol is an example of a *time-division duplex (TDD)* scheme.

With low power requirement goal in mind, the Bluetooth specification defines three different classes of radio powers, 1mW, 2.5mW and 100mW, covering maximum ranges between 10 to 100 meters. In contrast, the typical transmit power of 802.11b Wireless LAN technology is 1000mW. Bluetooth links also have low bandwidth capacity of 1Mbps compared to 802.11b's 11Mbps. Both Bluetooth and 802.11b operate in the same 2.4 GHz frequency band. With different design goals in mind, Bluetooth and 802.11b are poised to serve different applications and have potential to complement each other.

The combination of TDD and FHCDMA schemes has made internetworking piconets interesting and challenging. The rest of this chapter gives an overview of these challenges and a summary of our solutions. We begin by discussing the motivation behind our research in the next section.

1.1 Motivation

One of the fundamental goals of the Bluetooth technology is to eliminate the need for cables when connecting up everyday devices. Figure 1-1 shows various Bluetooth usage models. By replacing cables with Bluetooth, a typical desktop will become a cordless computer. The Bluetooth enabled headset can be used to send voice commands or messages to and from any other Bluetooth devices including stationary or mobile phones, laptops, refrigerators, CD players, or even toasters. Bluetooth technology promises many future applications such as instant postcard and interactive conference applications. One can take pictures with a Bluetooth-enabled digital camera and sends out images instantly to a Bluetooth-equipped cell phone which then transmits those images to a desired location over the Internet. Similarly, participants of an interactive conference can exchange voice and data using Bluetooth enabled laptops and PDAs. Hence, Bluetooth is appealing as the personal area networking technology to connect everyday devices without cables or manual configuration.

1.1.1 Usage for *Ad Hoc* Scatternets

Bluetooth technology promises much more beyond cable-free connectivity between a small number of devices [2] in an isolated piconet. Bluetooth can be extended to interconnect multiple piconets to form a larger *ad hoc* scatternet consisting of hundreds of devices. Multi-hop scatternets can provide connectivity over distances greater than the short radio range as long as there are sufficient relay nodes between overlapped

[2] Recall that a Bluetooth piconet consists of a maximum of 8 devices.

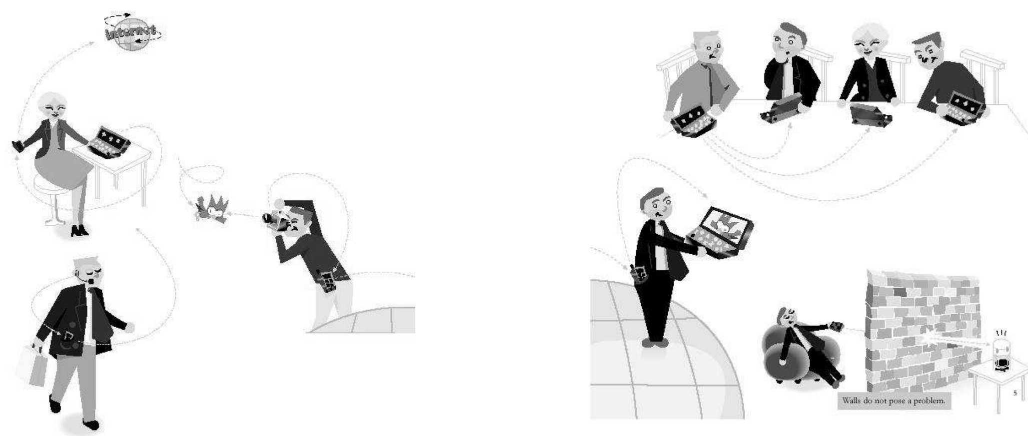


Figure 1-1: Various Bluetooth Usage Models (adapted from [1]).

piconets. An example usage model is the interactive conference scenario where participants in the conference, each with his own piconet, can exchange data via a connected scatternet. Bluetooth scatternets are also suitable for crowded malls. Many shoppers, each carrying a piconet, can exchange data including digital coupons while receiving advertisements from Bluetooth devices stationed in the shops. The frequency hopping technique employed by Bluetooth allows multiple piconets to communicate within the radio range with little interference between them.

Broadly speaking there are two approaches to connecting piconets. In environments with a pre-existing network infrastructure such as office buildings and malls, Bluetooth access points, connected to each other and to the Internet via the wired network, can be stationed across the buildings to provide connectivity to residents. On the other hand, Bluetooth technology appeals to many application scenarios where no communication infrastructure exists. Bluetooth scatternets can provide seamless connectivity between devices in places such as open football, baseball and soccer fields, parking lots, and inside cars and subways. The capacity of a single scatternet in these scenarios can range from tens to hundreds of nodes.

Furthermore, Bluetooth technology can also be used in wireless sensor networks. Sensor networks are widely used in industrial automation such as oil and gas processing, chemical production and automobile manufacturing. Sensor networks provide reliable monitoring of various environments for both civil and military applications ranging from patients monitoring and home energy management to land mines detection and tactical information imaging in battlefields. Many of these scenarios call for wireless sensor networks as opposed to wired ones. Wireless sensor networks can also be much more cost and time effective than the wired counterparts. [3] Generally, a

[3] In oil and gas processing, the cost of mounting and wiring sensors on large refinery tanks can be as high as $2,000 per foot. [7]

wireless sensor network is energy-constrained, has low bandwidth, and contains hundreds of sensors. Thus, the low cost, low power Bluetooth technology is suitable for creating *ad hoc* scatternets connecting hundreds of Bluetooth sensors [4].

1.2 Challenges and Solutions

A number of challenges exists before Bluetooth scatternets become a reality. The goal of this thesis is to identify those challenges and present solutions for them. More specifically, given an *ad hoc* network where,

1. Nodes arrive and depart at arbitrary times,
2. Various subsets of nodes are within radio communication range of each other, and
3. Every node uses a single radio chip and the communication is based on the master-slave TDD scheme.

One must provide a set of online algorithms that satisfy the following properties:

1. Each node is in at least one piconet,
2. The piconets overlap in such a way that there are sufficient relays to create a reasonably short path between each pair of nodes,
3. Routing packets is efficient, and
4. There exists an efficient schedule for wireless channel transmissions.

We identify the three main challenges involved in satisfying the aforementioned properties as: i) topology formation, ii) link scheduling, and iii) packet routing. In broadcast based wireless LANs such as 802.11b, the network topology is determined by the physical distance between nodes. In Bluetooth, an explicit topology formation process is required since nearby devices need to discover each other and explicitly establish point-to-point links. During the link formation process, the Bluetooth nodes synchronize the frequency hopping sequence and gather necessary clock information. This essential *ad hoc* discovery process could be lengthy. Therefore, algorithms are required to quickly form a network topology that spans across all nodes within the transmission proximity.

A link scheduling mechanism is necessary since relay nodes need to communicate in multiple channels or links. This is because Bluetooth devices, each with a single radio chip, can only be active on one channel at a time and thus, nodes must communicate in different channels on a time division basis. Hence, a coordinated

[4] We note that the current Bluetooth technology may not be suitable for some wireless sensor networks with very stringent power requirements.

link scheduling mechanism is required for neighboring nodes to carry out successful packet transfers. The overall performance of the scatternet-wide communication critically depends on the scheduling mechanism which dictates the bandwidth utilization, end-to-end latency and the energy usage.

A routing mechanism is also essential to route packets over a multi-hop scatternet. Small Bluetooth packet size and low memory and energy requirements dictate the design of *ad hoc* scatternet routing protocols.

In developing a set of online distributed mechanisms for self-organizing scatternets, we aim to achieve the following goals in a prioritized manner:

1. The protocols must be incremental, i.e. be able to adapt to arbitrary node arrivals and departures,

2. Must be scalable up to a few hundreds nodes,

3. Must be energy efficient,

4. Must have low memory requirements, and

5. Must be reasonably efficient in terms of communication latency, bootstrap delay, and system bandwidth.

With these goals in mind, we summarize our approach to each of the three problems (topology formation, scheduling, routing) in detail in the following sections.

1.2.1 Topology Formation

Since Bluetooth is a relatively new technology, little related work exists in the area of constructing Bluetooth scatternets. The scatternet formation schemes presented in [22, 2, 30] use randomized strategies to form scatternets with certain properties. Specifically, the schemes in [22, 30] attempt to minimize the number of piconets in a scatternet for reasons varying from simplicity to possible interference between multiple piconets. It is not clear, however, that minimizing the number of piconets is always desirable. A scatternet with k piconets can have at most k simultaneous master-slave communications. A scatternet with a higher number of piconets can potentially have higher overall system capacity. In addition, the earlier protocols are limited to scenarios where all nodes arrive over a small window of time and do not deal with dynamic environments where nodes arrive and leave arbitrarily.

In contrast, our scatternet formation algorithm, called TSF (for Tree Scatternet Formation), connects nodes in a tree structure while dynamically assigning master/slave roles to each node. Our algorithm is both decentralized and self-healing, in that nodes can join and leave at any time without causing long disruptions in connectivity. We have chosen a tree topology, in contrast to the approach proposed in [22, 2, 30], 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. We believe that a tree topology simplifies the scheduling of Bluetooth communication links. The hierarchical nature of a tree allows links at the same level to be scheduled simultaneously without conflicts. Furthermore, links at non-consecutive levels can also be scheduled simultaneously. Finally, observe that although a common objection to trees is that they are more likely to cause disconnections, our healing algorithm makes the disruptions associated with nodes leaving small. In Chapter 2, we describe TSF in detail, analyze its performance, and examine its salient properties.

1.2.2 Link Scheduling

There has been relatively little work done in the area of scatternet-wide link scheduling. It is impractical to coordinate all the nodes in the scatternet to arrive at a link schedule that maximizes the system's overall efficiency. Racz et al. presents a pseudo random scheduling scheme (called PCSS) in which nodes randomly choose communication checkpoints for adjacent links [28]. Although PCSS avoids scatternet-wide coordination, and therefore, has low scheduling overheads, it is not responsive to bursty traffic because of the lack of coordination at each level.

In contrast, our scatternet scheduling algorithm called TSS (for Tree Scatternet Scheduling) adapts to various (local) workload conditions quickly while attempting to increase the system-wide throughput and reduce the end-to-end communication latency. TSS exploits the fact that the scatternets produced by TSF have tree topologies, and coordinates one-hop neighbors to schedule communication tasks efficiently. In Chapter 3, we present TSS in detail and show that it yields good performance in terms of throughput and end-to-end packet latency under various traffic loads.

1.2.3 Packet Routing

Much research has been done in the area of routing packets in mobile *ad hoc* networks especially based on the 802.11b wireless LAN technology [29]. Among them, Dynamic Source Routing [19] and Ad hoc On-Demand Distance Vector Routing [27] stand out because of their ability to cope with mobility with reasonable overheads [10]. However, it is not clear whether those *ad hoc* routing protocols are suitable for Bluetooth networks where energy efficiency is critical, packets are rather small and thus sensitive to bytes overheads incurred by routing protocol, and scatternet-wide broadcast is considered an expensive primitive. Bhagwat al et. presents a routing scheme for Bluetooth scatternets called Routing Vector Method (RVM) that can be considered a variant of *ad hoc* source routing protocols [5]. RVM addresses the issues particular to routing in Bluetooth scatternets.

For our thesis research, we do not propose a particular *ad hoc* scatternet routing scheme. We note that our topology formation scheme simplifies routing by forming a tree scatternet in two ways. First, tree topology guarantees loop-freedom and thus, *ad hoc* routing protocols can eliminate overheads associated with detecting duplicate broadcast packets due to routing loops. Second, tree topology guarantees a unique path between any node pair and therefore, it is easy to encode the (link layer) path information between node pairs efficiently. This is useful for source routing protocols such as RVM and DSR that encode the entire path information in the packet headers.

1.2.4 Bluetooth Protocol Stack

In this section, we summarize the functionality of each layer in the Bluetooth protocol stack. Figure 1-2 compares the Bluetooth protocol stack (on the right) with the Open Systems Interconnect (OSI) standard reference model [9]. The radio and part of Baseband layer perform modulation, demodulation and channel coding for actual transmission and reception of data on air, and thus, are comparable to the Physical layer. The other part of Baseband and some part of Link Controller layer are responsible for framing, error checking and correction, and thus, are equivalent of the Data Link layer. The rest of Link Controller combined with Link Manager carries out the duties of the combined Network and Transport layer by setting up, maintaining, and tearing down links, and by providing reliability and multiplexing of data transfers across Bluetooth links. The Bluetooth specification defines a Host Controller Interface (HCI) to provide a uniform interface for accessing the lower layers. The specification provides flexibility to implement the four layers, Radio, Baseband, Link Manager, and HCI as a Bluetooth module on a single processor, and the upper layers on a separate host processor. The host protocol stack consists of L2CAP and RFCOMM/SDP layers that provides management and data flow control services and a common representation for applications data.

Figure 1-3 shows where in the protocol stack topology formation, scheduling and routing agents can be implemented. Instead of using HCI commands, the agents could also access directly to each lower layer. In fact, the current HCI specification may not expose all the capability of lower layers needed by a link scheduling scheme and, to a less extent, a topology formation scheme.

1.2.5 Bluetooth Simulator

To evaluate the effectiveness of our algorithms, we have developed a Bluetooth simulator as an extension to the well-known Network Simulator (*ns*) [26]. Our development efforts were eased by the *bluehoc* simulator developed by IBM [6]. Our simulator implements most aspects of the Bluetooth protocol stack according to the Bluetooth specification (version 1.1). Figure 1-4 shows various modules of our simulator. The Baseband module of the simulator implements the pseudo-random frequency-hopping technique the Inquiry, Inquiry Scan, Page, Hold, and Role-Change as specified in the Bluetooth Baseband specification. The LMP module implements detailed LMP protocols and the LC module implements link control operations such as the Automatic Repeat Request (ARQ) scheme. As shown in the figure, every link has unique instances of the LC and LMP modules which are responsible for carrying out all the Bluetooth primitive link establishment, control and communication operations. The simulator also includes the HCI layer to allow various scatternet formation, scheduling and routing schemes to use the functionality of the lower Bluetooth layers easily.

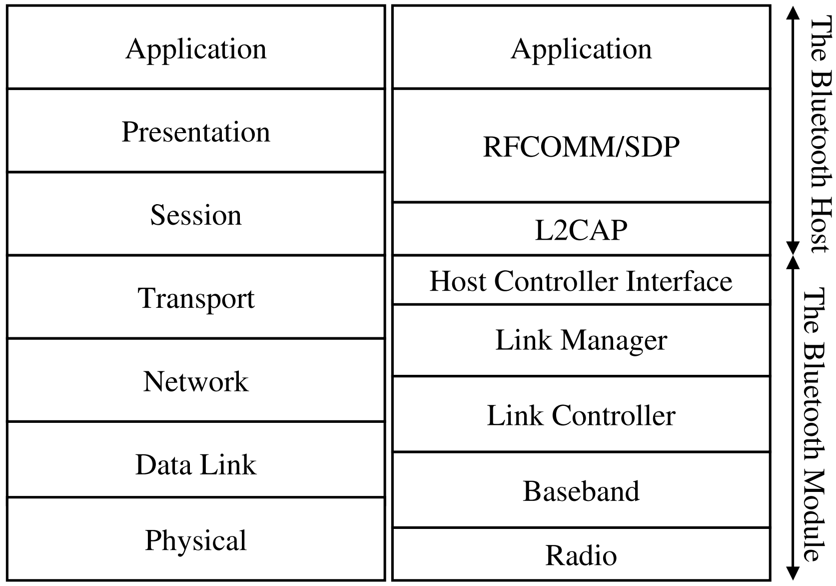


Figure 1-2: OSI reference model v.s. the Bluetooth protocol stack.

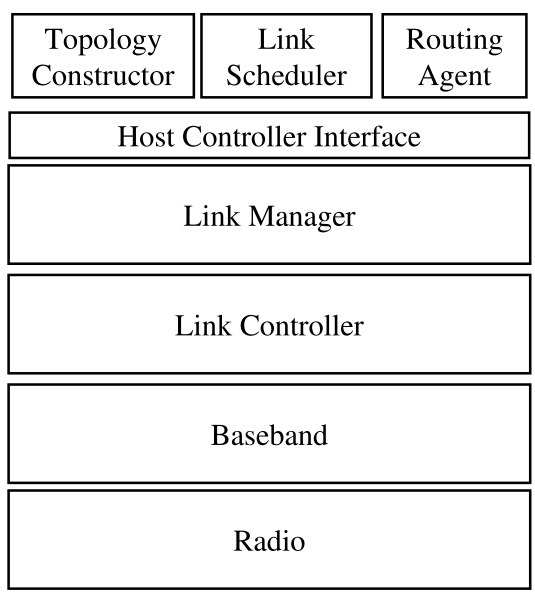


Figure 1-3: Interface between the scatternet formation, scheduling and routing modules and the Bluetooth core protocol stack.

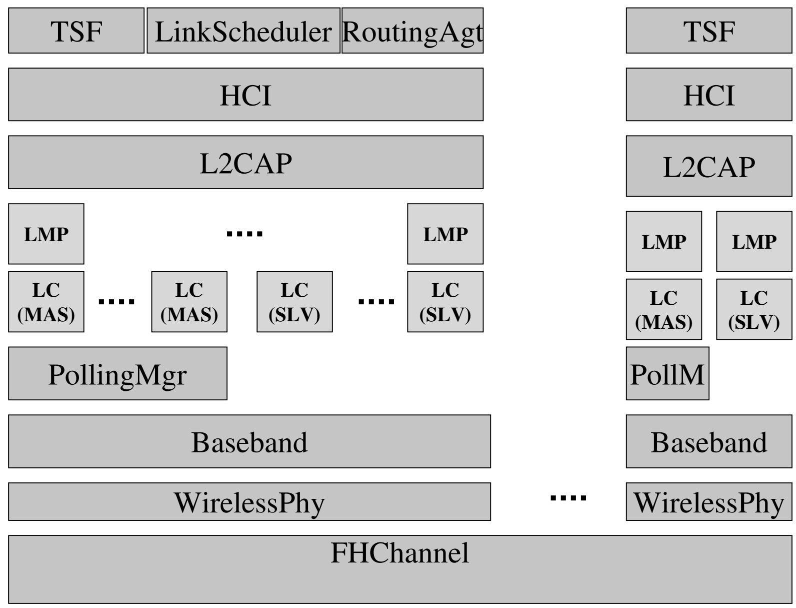


Figure 1-4: The Bluetooth Simulator's Modules

Each device in the simulator is equipped with a Bluetooth protocol stack through which it discovers and communicates with other devices. Note that devices are capable of assuming both master and slave roles on a time-division basis. All the devices share the wireless medium consisting of 79 channels, which is simulated by the FHChannel module. Our simulator implements many important aspects of the Bluetooth protocol stack in detail and hence gives us insights in understanding engineering difficulties as well as performance aspects.

1.2.6 Evaluation

We have implemented the topology formation (TSF) and link scheduling (TSS) schemes in the simulator and conducted several experiments to evaluate various performance aspects of our algorithms. Sections 2.3 and 3.3 present the detailed evaluation of TSF and TSS respectively.

For TSF, we measure connection setup, scatternet formation and healing latencies, and the time a node spends searching for neighboring nodes. We also compare our results with the previous results published in [30, 22]. In addition, we analyze the properties of resulting scatternet topologies. We evaluate TSS based on average and total throughput available to applications and average end-to-end packet delay perceived by applications. We conduct several experiments with varying workload conditions and compare some of our results with the optimum.

1.3 Contributions and Thesis Structure

In this thesis, we identify the three main challenges in realizing scatternets: i) scatternet formation, ii) link scheduling and iii) packet routing. We have developed an efficient online scatternet formation algorithm, TSF, which connects nodes in a tree structure. Unlike earlier work, our design does not restrict the number of nodes in the scatternet, and also allows nodes to arrive and leave at arbitrary times, incrementally building the topology and healing partitions when they occur. TSF 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.

We have also developed an online scatternet scheduling algorithm, TSS, that coordinates communication tasks on various links efficiently. Unlike previous work, TSS coordinates one-hop neighbors effectively to increase the overall performance of the scatternet. In addition, TSS is robust and responsive to network conditions, adapting the inter-piconet communication schedule effectively based on varying workload conditions.

In an *ad hoc* environment where nodes come and go arbitrarily, a scatternet must attempt to self-organize while still allowing nodes to communicate efficiently at the same time. Thus, interoperatability between the scatternet formation scheme and the link scheduling scheme is important. In our integrated approach, the tree topology serves as the centerpiece as it simplifies link scheduling and packet routing. Our link scheduling algorithm TSS not only schedules communication tasks efficiently but also ensures that scatternet formation tasks are carried out in a timely fashion. To our knowledge, we are the first to present an integrated and complete solution to realize self-organizing scatternets for dynamic environments where nodes arrive and depart at any time.

We have also developed a Bluetooth simulator in *ns* which includes most aspects of the Bluetooth protocol stack. We implemented both TSF and TSS in our simulator and ran numerous simulations to evaluate their performance. This also demonstrates that both schemes can be implemented within the existing Bluetooth specification (Version 1.1). Our simulation results 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. The simulation results also show that TSS achieves high throughput and low packet latency for various traffic loads.

Chapter 2 and 3 discuss topology formation and link scheduling in detail respectively. Each chapter begins with the identification of the problem, followed by background materials describing related works and relevant aspects of Bluetooth, and a performance evaluation and summary. We conclude in Chapter 4 with a summary of the thesis work and a discussion of future work.

Chapter 2

Topology Formation

Bluetooth-like link technologies are a recent development, and one can only speculate on how they might be used. Broadly speaking, we believe there are two distinct ways in which Bluetooth-based scatternets will be used. Some environments, e.g., a network connecting household appliances, will be largely static. In these environments, it will be reasonable to statically configure scatternets in the way many wired (and wireless) networks are configured today. Our work is aimed at more dynamic environments, in which the relatively frequent arrival and departure of nodes and node mobility make manual configuration problematic.

We consider three usage scenarios: i) an interactive conference scenario, ii) an outdoor network scenario, and iii) a sensor network scenario. In the interactive conference scenario, a few dozens of participants equipped with Bluetooth laptops and headsets arrive in succession over a small time period. Once the meeting has started, participants rarely move or leave the meeting, and new participants seldom arrive. The scatternet ceases to exist when the meeting ends in a few hours.

The outdoor network scenario is intended for large *ad hoc* meetings taking place in open baseball and football fields. In this scenario, several participants each carrying a group of Bluetooth devices arrive within a relatively short period and form a connected scatternet among their devices. The scatternet remains active for a few hours during which a small number of participants may leave or join the network. The scatternet then disappears as all remaining participants leave rapidly. The scatternet capacity in this scenario can range from tens to hundreds of devices.

Lastly, the sensor network scenario represents a generic Bluetooth sensor network to be used for reliable monitoring of various civil and military environments. In this scenario, hundreds of groups of sensors arriving in rapid succession may be connected in a single scatternet. Sensor nodes rarely move or leave the network and the network exists for days or months.

We can capture the dynamics of the aforementioned scenarios with two different environments: i) where nodes arrive *en masse* and no nodes leave, and ii) where nodes arrive and depart in incremental fashion. Our efficient scatternet formation algorithm , called TSF (for Tree Scatternet Formation), is designed to work well in both of these modes of dynamic operation. TSF assigns master/slave roles to nodes while connecting them in a tree structure. Our algorithm is both decentralized

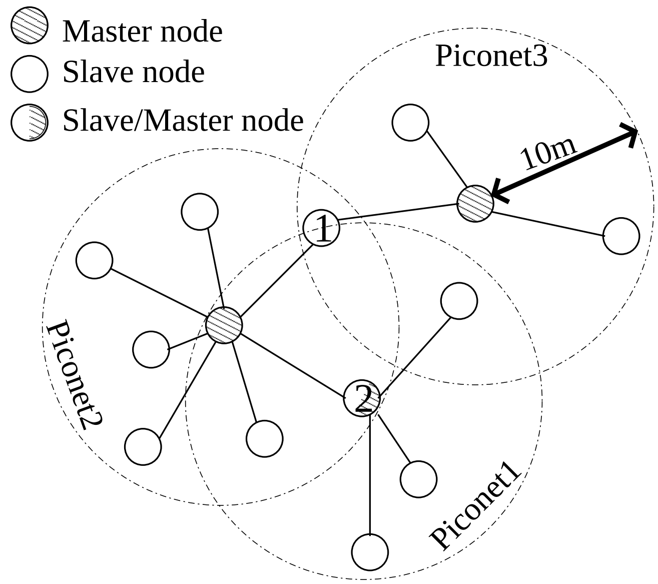


Figure 2-1: A Bluetooth scatternet with two types of relay nodes: node 1 acts as slave in both piconet 2 and 3 ("slave relay"), while node 2 is master in piconet 1 and slave in piconet 2 ("master relay").

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.

A major problem that needs to be solved in forming scatternets is deciding which nodes should act as relays that interconnect piconets. Judiciously choosing relays, such as nodes 1 and 2 in Figure 2-1, is important because it determines the nature of the resulting topology. We chose a tree topology, in contrast to the approaches proposed by Salonidis et al. [30] and Law et al. [22], because it simplifies both the routing of messages and the scheduling of communication events.

Compared to previous approaches, 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. In contrast, more general *ad hoc* routing protocols [27, 29, 10], either incur per-packet overhead as in Dynamic Source Routing (DSR) [19] or Routing Vector Method (RVM) [5], or increase memory requirements as in Ad-hoc On-Demand Distance Vector (AODV) [27].

A tree topology also simplifies the scheduling of Bluetooth communication links. TSF achieves the minimum number of average piconets per bridge node by ensuring that every bridge node (internal node) participates in exactly two piconets. The

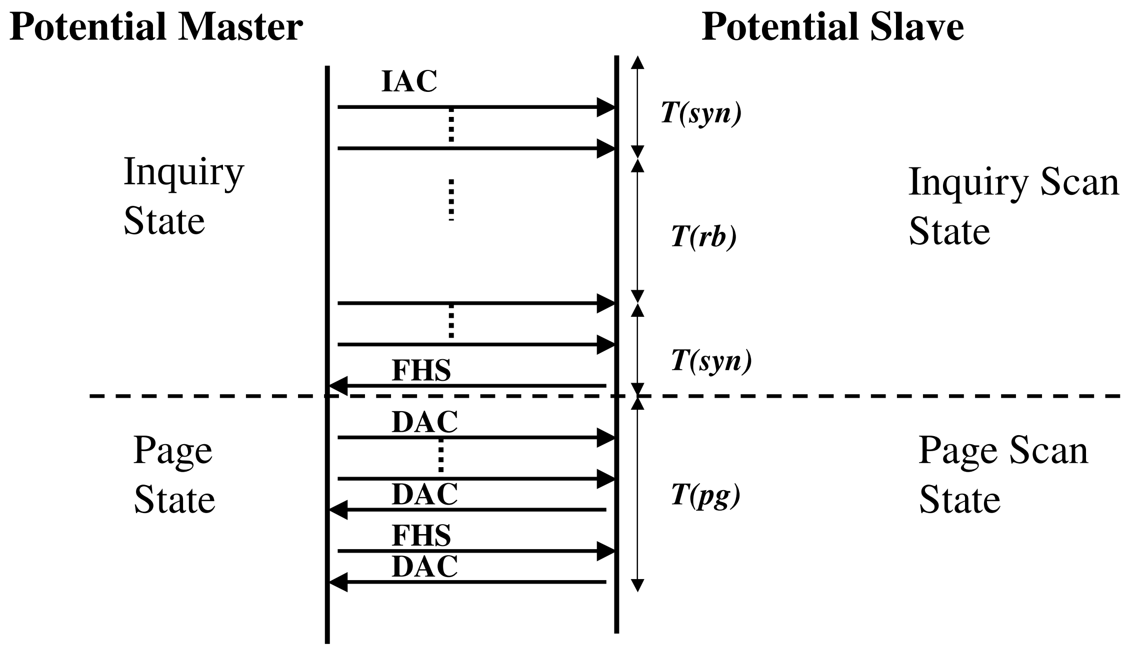


Figure 2-2: Bluetooth link formation process.

hierarchical nature of a tree allows links at the same level to be scheduled simultaneously without conflicts. Furthermore, links at non-consecutive levels can also be scheduled simultaneously. Finally, observe that although a common objection to trees is that they are more likely to cause disconnections, our healing algorithms make the disruptions associated with nodes leaving small.

In Section 2.1, we explain the Bluetooth link formation process and prior work on scatternets. Section 2.2 presents the details of our algorithm, TSF. Section 2.3 evaluates the efficiency of TSF and analyzes the properties of resulting topologies.

2.1 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, which is part of the Bluetooth specification, is necessary to understand our topology formation algorithm. We then discuss previous work.

2.1.1 Bluetooth Link Formation

The link formation process specified in the Bluetooth Baseband specification consists of two processes: *Inquiry* and *Page* [8]. 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

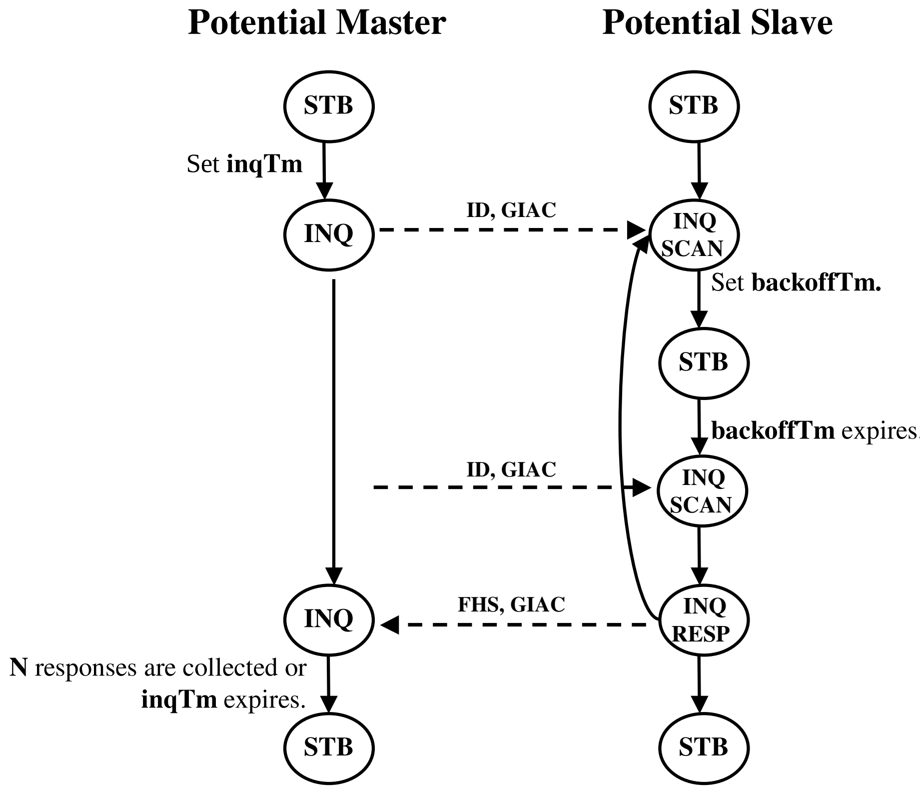


Figure 2-3: State transitions during the Inquiry process.

those neighbors. 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. Figure 2-2 illustrates the Bluetooth link formation process.

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.

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_{rb}, between 0 and 1023 time slots before responding with the signaling information. Let T_{sync} be the delay before two nodes can synchronize their frequencies during the Inquiry process. The time taken to complete the Inquiry process is given by

$$T_{inq} = 2T_{sync} + T_{rb} \tag{2.1}$$

T_{sync} varies according to the differences in clock values between nodes. If the two nodes have synchronized clocks, T_{sync} will be very short and thus, T_{rb} dominates T_{inq}.

However, if they are not synchronized, T_{sync} will dominate. We also note that in some cases the second synchronization delay will be much shorter than the first one. This is because after the backoff timer expires, the scanning node listens on the same frequency that it did before it backed off. Hence, if the backoff timer is relatively short, the scanning node may be able to receive the ID packets from the same node performing Inquiry in a short period. However, for simplicity, we consider the first and second synchronization delays to be the same.

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.[1] 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 2-3 and 2-4 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 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

$$T_{conn} = T_{inq} + T_{pg} \tag{2.2}$$

T_{inq} is typically much larger than T_{pg} and dominates the delay to enter the Connection state.[2]

2.1.2 Related Work

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 [17, 11, 24, 20, 13, 23]. None of these, however, deal with the complications introduced by the master-slave frequency hopping TDD MAC layer used in Bluetooth.

The Bluetooth link formation mechanism requires that each node is pre-configured to carry out either master or slave role. Recall that potential master carries out Inquiry and Page operations while potential slave conducts Inquiry Scan and Page Scan

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

[2] 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.

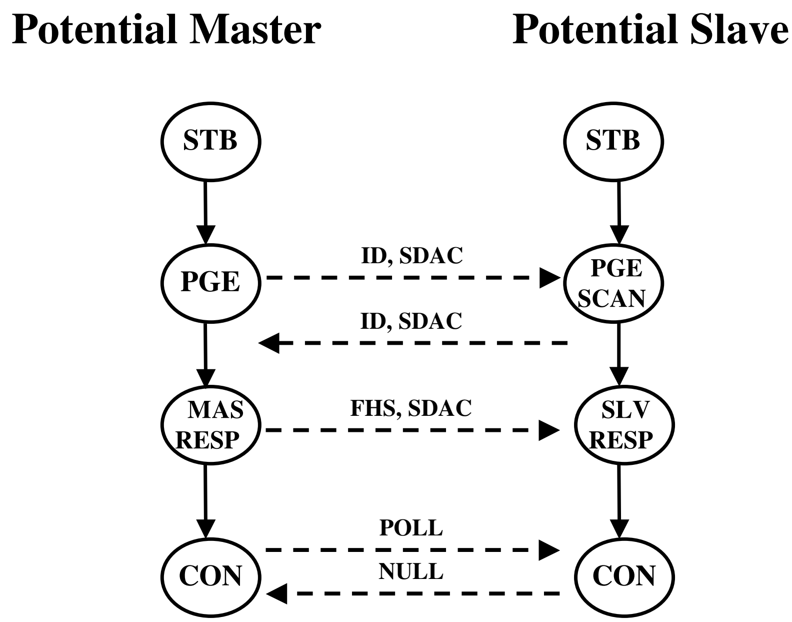


Figure 2-4: State transitions during the Page process.

operations. 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. Algorithms are required to automatically assign roles to nodes as they attempt to connect to each other.

Salonidis *et al.* present a symmetric link formation scheme where no configuration of potential master or slave roles is necessary [30]. 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 of time. Also, it does not provide a mechanism for healing the network when nodes disappear (e.g., because they have moved or failed). Finally, the scheme described limits the maximum number of nodes involved in the scatternet formation to be 36. Our scheme has none of these limitations.

Law *et al.* have developed a randomized distributed scatternet formation protocol and evaluated the performance [22]. 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. They show that the protocol runs in *O(log n)* time and sends $O(n)$ messages. It is not clear that minimizing the number of piconets is always desirable. A scatternet with k piconets can have at most k simultaneous master-slave communications. A scatternet with a higher number of piconets can potentially have higher overall system capacity. Certainly there are trade-offs between the increase in system's capacity and the increase in collisions as a result of having an additional piconet. Like the scheme developed by Salonidis et al., Law et al.'s protocol does not deal with dynamic environments where nodes may arbitrarily join or leave the network. For simplicity, their protocol relies on synchronized rounds where devices discover their neighbors simultaneously. In contrast, our approach is asynchronous as nodes attempt to build up a connected scatternet based on their own state machines, and thus, is more efficient and robust. Lastly, their protocol constantly moves nodes from one piconet to another as it forms a larger scatternet of desired properties. This causes disruptions in communication between connected nodes. In contrast, TSF builds up a tree scatternet incrementally without requiring relocation of nodes.

2.2 TSF: Tree Scatternet Formation

Bluetooth-like link technologies are a recent development, and one can only speculate on how they might be used. Broadly speaking, we believe there are two distinct ways in which Bluetooth-based scatternets will be used. Some environments, e.g., a network connecting household appliances, will be largely static. In these environments, it will be reasonable to statically configure scatternets in the way many wired (and wireless) networks are configured today. Our work is aimed at more dynamic environments, in which the relatively frequent arrival and departure of nodes and node mobility make manual configuration problematic.

In some dynamic environments, such as in a scheduled meeting, most nodes arrive *en masse*. In other environments, e.g., a shopping mall, nodes arrive and leave in incremental fashion. Our algorithm is designed to work well in both of these modes of dynamic operation.

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

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, 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 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.

2.2.1 State Machines

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, called *free nodes*, that are seeking to join another tree to form a larger component and reduce the number of components. We denote the *root node* of tree T_k by r_k. We refer all other nodes in a component other than root as *tree nodes*. Every 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 provide loop-freeness, TSF distinguishes between two kinds of component merges: i) merges of trees each having more than one nodes, 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 single node from each tree, either root node or tree node, to be the *coordinator* which is responsible for discovering coordinators from neighboring trees as explained in later sections. All other tree nodes that are not coordinators spend a small of time looking for free nodes to connect to. 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 simple 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 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 as explained earlier.

The pseudo-code for different state-machines running at various types of nodes is shown in Figure 2-5. *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. We explain the use of Inquiry Access Code in detail in Section 2.2.4. 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, given by Equation 2.1. 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. This is presented in Section 2.3.2.

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. It is important for a tree node to spend more of its time involved in communication when it is handling high traffic volume. On the other hand, it is also important for these nodes to perform the social task of forming bigger trees and improving the overall connectivity

```
PROCEDURE TSF-FREE() {
   iac ← GIAC
   state_pair ← (Inquire, Comm/Scan)
   state ← random state from state_pair with 0.5 probability
   do forever
      t_state ← random(E[T_inq], D)
      Run(state, t_state, iac)
      state ← opposite state from state_pair
}
PROCEDURE TSF-ROOT() {
   if(designated as coordinator)
      TSF-COORDINATOR()
   else
      Run(Comm, ∞, null)
}
PROCEDURE TSF-TREE() {
   if(designated as coordinator)
      TSF-COORDINATOR()
   else
      iac ← GIAC
      state_pair ← (Comm, Comm/Scan)
      state ← random state from state_pair with 0.5 probability
      do forever
         if(state = Comm)
           t_state ← f_comm × random(E[T_inq], D)
         else
           t_state ← random(E[T_inq], D)
         Run(state, t_state, iac)
         state ← opposite state from state_pair
}
PROCEDURE TSF-COORDINATOR() {
   iac ← LIAC
   state_pair ← (Inquire, Comm/Scan)
   state ← random state from state_pair with 0.5 probability
   do forever
      t_state ← random(E[T_inq], D)
      Run(state, t_state, iac)
      state ← opposite state from state_pair
}
```

Figure 2-5: Pseudo-code of various TSF state-machines.

of the scatternet. An ideal value for f_{comm} should strike a balance between these two factors. But to find an ideal f_{comm} value, a node needs an accurate measurement of its current communication load and the number of disconnected components within the radio range. In the interest of simplicity, we approximate the ideal f_{comm} value as a function of how many children a node has. Let d be the number of adjacent links.

$$f_{comm} = d \tag{2.3}$$

We find this approximation produces efficient communication topologies with short average path length as later demonstrated in Section 2.3.5.

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 slave. Although roots are responsible for merging components, they do not spend any time in either Inquire or Scan states to discover neighboring components. 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 is important. It ensures that root nodes are not overburdened with the energy-intensive task of performing Inquiry, and thus, can spend almost 100% of the time communicating. Roots spend a small amount of time in Page and Page Scan modes to merge their components together. In the next section, we explain in detail how coordinators are elected and how component merges take place, and prove the correctness of TSF.

2.2.2 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. Recall that the Page process takes a much smaller of time compared to the Inquiry process. 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 a root node joins another node as a child, the child 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 former root. We use this master-relay strategy because it is simple, and because it minimizes the number of piconets in which a relay node participates (at most two, the minimum possible). This in turn reduces the scheduling and piconet-switching overhead, both of which are significant in Bluetooth.

Tree nodes alternate between the Comm and Comm/Scan states. They may connect to free nodes as slaves. Once a connection is established, the tree node (slave) initiates the role-switch procedure and becomes master. Note that the role-switch procedure only requires a few single slot messages to exchange clock information of the two devices involved.

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

1. Free nodes may only connect to other free nodes, or to tree 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 former free node becomes the slave.

2. Root nodes of trees with more than one node may only connect to other root nodes. One of them becomes the master and the other the slave in the master's Bluetooth piconet.

3. Tree nodes do not attempt to form larger trees with nodes that are not free nodes.

Theorem 2.2.1. *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 tree 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. □

Mas/Slave	Root	Tree	Free	Coordinator
Root	1	0	0	0
Tree	0	0	1	0
Free	0	1	1	0
Coordinator	0	0	0	1

Table 2.1: Link formation combination: entries with 0 are invalid. Root and Tree entries are meant for root and tree nodes which are not assuming coordinator role.

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 2.1 shows the valid combination of master-slave connection establishment between different types of nodes.

We do not allow the connection between tree nodes and root nodes since this has the potential to create self-loops or multi-hop loops. It would be possible to allow the connection and check for loops. But doing so would involve a significant amount of control message overhead within the scatternet. In fact, TSF produces trees with very little communication between nodes already in the scatternet, and is well-suited to a Bluetooth implementation as explained in Section 2.2.4.

We also note that no loops will form if we allow free nodes to join root nodes. 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 tree nodes help free nodes join the scatternet, works well.

To avoid periodic synchronization effects, TSF randomizes the state residence time from an interval $[E[T_{inq}], D]$. 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 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.

2.2.3 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 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 and change it to the appropriate node type. When a slave loses the connectivity to its parent, it updates its node type as follows. 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 only experienced a few instances of this particular situation in hundreds of simulations involving 80 or fewer nodes. There are two 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, when the two root nodes merge, the root node assuming 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.

2.2.4 Bluetooth Implementation

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 2-6 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.

Nodes running TSF state machines 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 coordinator nodes, coordinators only transmit and listen to ID packets containing LIAC. All other nodes (tree and free) 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.

There is a rare 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

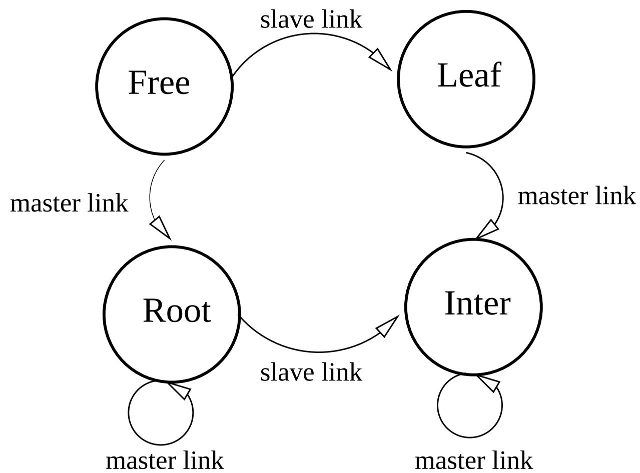


Figure 2-6: Node state transitions during topology construction.

state does not know whether an inquiry response (FHS packet) is in response to that node's inquiry. Consider a coordinator node A and a free node B both performing Inquiry. A tree node C from the same subtree where A is attached to hears the ID packets transmitted with GIAC from B, and responds with an FHS packet. By chance, both A and B happen to hear the replied FHS packet simultaneously and attempt to page C which has entered the Page Scan state as described in Section 2.1.1. Note that A has no way of knowing that C is a tree node in its own subtree. If A is successful before B in establishing a link with C, it will produce a cycle.

This problem can easily be avoided by including one extra bit of information, stating whether the node sending the response is a coordinator 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. The child node will verify whether the parent node is a compatible node according to Table 2.1. If not, the child will tear down the link by sending appropriate HCI commands to the Baseband module. Both nodes then resume their previous roles. Again, we note that this kind accidental loop creation is a rare event.

2.3 Performance Evaluation

To evaluate the effectiveness of our algorithms, we have developed a Bluetooth simulator as an extension to the Network Simulator (*ns*) [26]. Our simulator implements most aspects of the entire Bluetooth protocol stack according to the Bluetooth specification Version 1.1 as explained in 1.2.5. We implemented TSF in the simulator and conducted several simulations to evaluate the performance of our algorithms on two different environments: *en masse* arrivals, and *incremental arrivals and departures*. In this section, we present the detailed models for evaluation and simulation results on link establishment, scatternet formation, and healing latencies, and discuss salient properties of the resulting topologies.

2.3.1 Models

As mentioned in Section 1.1, Bluetooth *ad hoc* scatternets are useful in many scenarios ranging from interactive conferences to sensor networks. Our work is focused on true *ad hoc* scatternet usage scenarios where no network infrastructure exists. We use a group of nodes rather than an individual node as our basis unit. This stems from an observation that during an interactive *ad hoc* meeting, each participant arrives with a group of personal devices which may have already been connected in a piconet. However, only a fraction of these nodes, usually one, will be interested in connecting with neighboring nodes from different groups. The rest of the nodes in the group may only communicate with a group leader or the master as slaves and thus, do not attempt to connect with other nodes outside the group.

We consider two phases during a period of T seconds: i) the formation phase where many nodes arrive rapidly to form a scatternet, and ii) the communication phase where many connected nodes communicate while some new nodes arrive. During both phases, some existing nodes may leave. At the end of T seconds, the scatternet ceases its operation as all remaining nodes leave. Both arrival and departure rates may differ during the periods before the meeting starts and after the meeting has started. For instance, most people arrive within the first 15 minutes before the meeting takes place but only a few people arrive during the meeting. The arrival and departure processes are defined as follows:

1. A group consists of n_a connected nodes, among which n_f nodes attempt to connect to neighboring nodes outside the group, where $1 \leq n_a \leq 8$ and $n_f \leq n_a$.

2. Groups of nodes arrive at rate λ_f and depart at rate μ_f during the period $[0, T_f]$,

3. Groups of nodes arrive at rate λ_c and depart at rate μ_c during the period $[T_f, T_c]$, where T_c is the time when the network ceases its operation, and

4. Both arrival and departure events follow uniform distribution.

Nodes in a group are connected to each other via TSF or manual configurations. Without loss of generality, we assume $n_f = 1$. Recall that TSF designates a single coordinator node from a component to discover neighboring components and that the state machine running at each coordinator is identical to the one running at a free node except for using different Inquiry Access Codes. For simplicity, we use a free node to represent a group of n_a connected nodes. Since $n_a - 1$ nodes do not participate in forming scatternet, they do not affect the performance in any way. We evaluate the performance of TSF in terms of various latencies related to forming connections and healing partitions in two different environments: i) where nodes arrive *en masse*

Parameter	En-masse	Incremental
n	$2 \le n \le 80$	30
n_a	$1 \le n_a \le 8$	$1 \le n_a \le 8$
n_f	1	1
T_c (seconds)	60	60
T_f (seconds)	1	30
λ_f (groups per second)	n	1.07
λ_c (groups per second)	0	0
μ_f (groups per second)	0	0
μ_c (groups per second)	0	$5 \le \mu_c \le 24$

Table 2.2: Various parameters for *en masse* and incremental environments.

$T_{inqscan}^{per}$	T_{scan}^{win}	T_{pg}^{to}	T_{pgscan}^{per}
1.28	0.01125	1.28	0.32

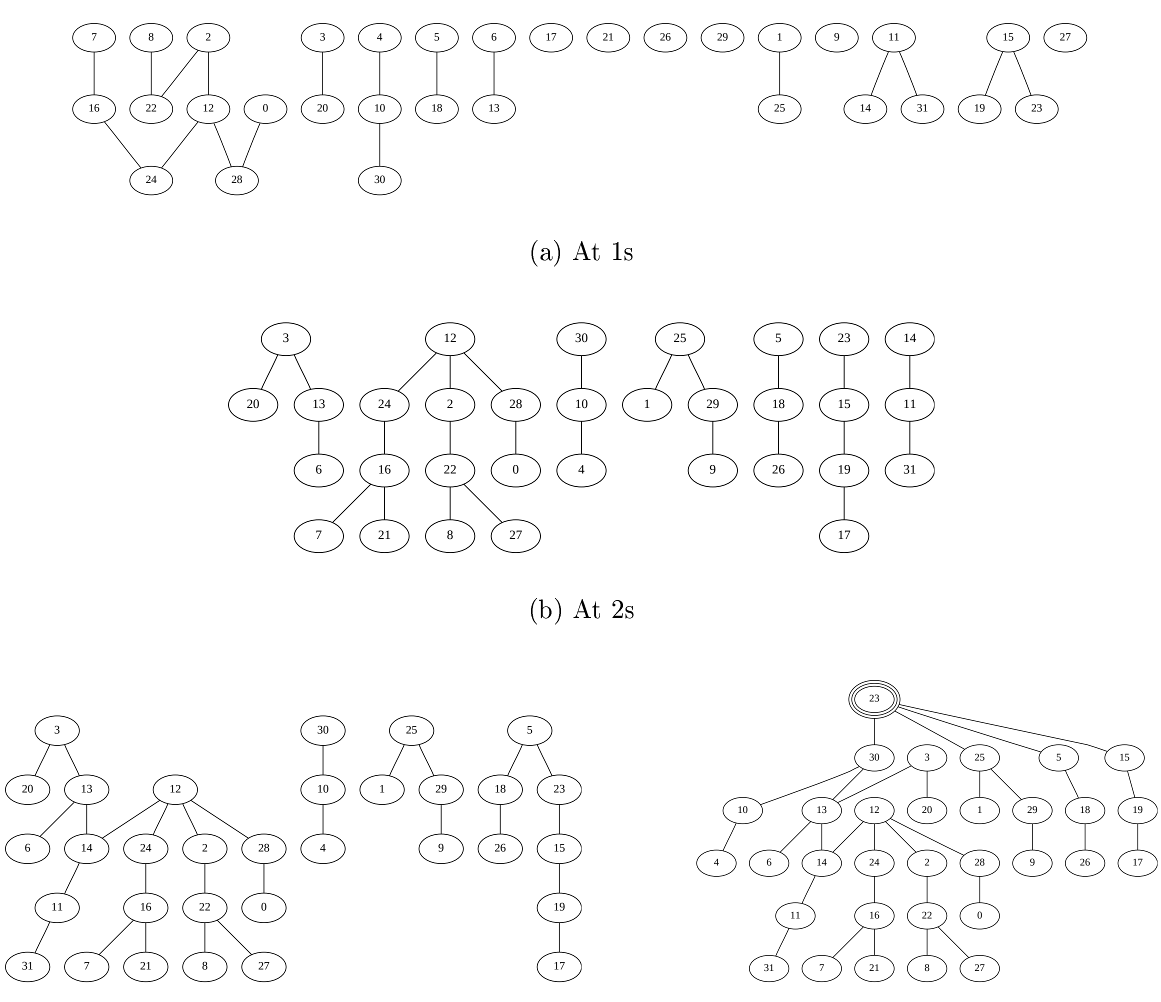
Table 2.3: Bluetooth Baseband parameters for Inquiry and Page processes (in seconds).

and no nodes leave, and ii) where nodes arbitrarily arrive and depart. For each environment, we ran simulations for various number of nodes. Table 2.2 shows the parameters we used during simulations.

2.3.2 Configurations

In all the experiments, nodes are assigned to a random clock value between 0 and $2^{27} - 1$ since every Bluetooth device has a free-running 27-bit clock where each tick lasts $312.5\mu s$. Every data point shown in all figures is the average of 100 independent trials. Table 2.3 shows the Bluetooth Baseband parameters we used for the Inquiry and Page procedures. $T_{inqscan}^{per}$ and T_{pgscan}^{per} are the periods between consecutive Inquiry Scan and Page Scan operations respectively. T_{scan}^{win} is the scanning window for both scanning operations, and T_{pg}^{to} is the timeout for both Page and Page Scan operations.

Through simulation, we determined that the expected time to complete the Inquiry process, $E[T_{inq}^{opp}] = 1.8s$, when two nodes are performing opposite discovery operations namely Inquiry and Inquiry Scan. Note that this value varies according to the Baseband parameters (see Table 2.3) used during the simulation. We also ran numerous simulations with two nodes running TSF to determine a good value for D. Recall that D is the parameter deciding the size of the random interval, which governs how long the node is resident in a given state. We found that setting $2.2 * E[T_{inq}^{opp}] \le D \le 4.4 * E[T_{inq}^{opp}]$ would give an average connection delay of 6-8s and chose $D = 3.8 * E[T_{inq}^{opp}]$ for all the simulation runs.



(a) At 1s

(b) At 2s

(c) At 3s

(d) At 10s

Figure 2-7: Evolution of a 32-node scatternet

2.3.3 *En masse* Arrivals

We start by evaluating the performance of TSF when nodes are arriving *en masse* and no nodes are leaving. We analyze the performance of TSF by measuring two important delays: link establishment delay and scatternet formation delay. Figure 2-7 shows an example of the evolution of a scatternet containing 32 nodes that all arrived *en masse* at time $0s$.

Link Establishment Delay

We define the connection setup or link establishment delay as the time taken before a free node can establish its first communication link with another node. This is an important metric because it gives a sense of how fast a (free) node can, on average, talk to its first neighbor. In an ideal case, the two nodes will be configured to perform

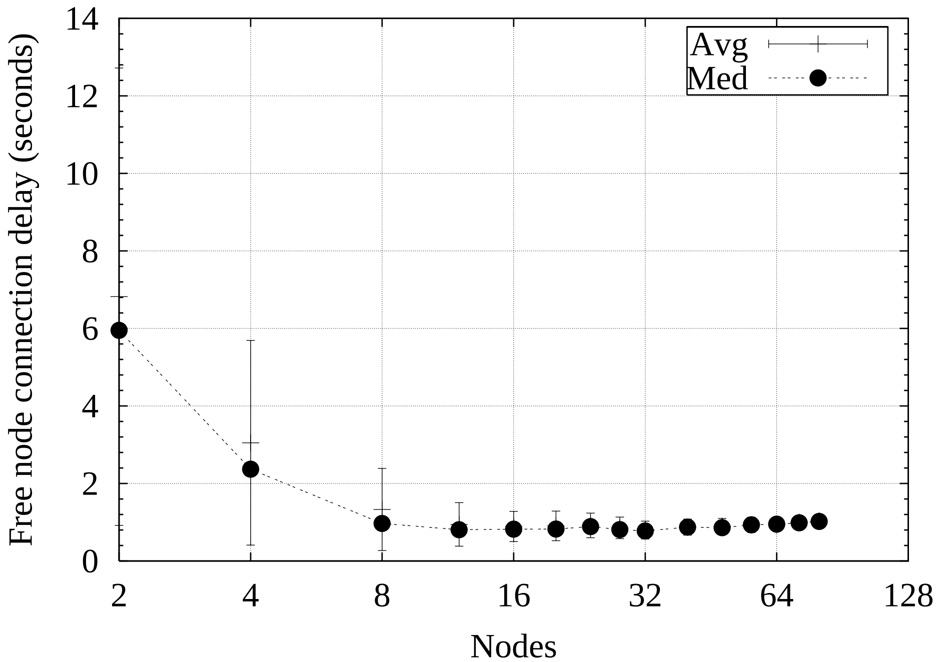


Figure 2-8: Connection delay for free nodes (*en masse*).

Inquiry and Inquiry Scan respectively at the same time and thus, the connection setup delay will be close to $E[T_{inq}^{opp}]$. On the other hand, if both nodes happen to choose the same state, they will have to wait until they reside in opposite states resulting in longer connection delay.

Figure 2-8 shows the average and median delays for a free node to setup a connection as a function of the number of nodes arrived. As the number of arriving nodes increases, the delay gets smaller. There are two reasons for this behavior. First, as more and more pairs of nodes perform Inquiry and Inquiry Scan, it becomes faster for several pairs to get connected. Second, the chances for a free node to get connected to a non-root node increase as more and more non-root nodes are performing Inquiry Scan. Recall that every non-root node periodically conducts Inquiry Scan to establish a communication link with a free node. Therefore, as the number of non-root nodes in multiple subtrees increases, it becomes faster and faster for a free node to get connected to an existing non-root node. This is apparent in Figure 2-7 where all free nodes are connected to different subtrees in just $2s$.

TSF achieves an average connection delay of $1s$ for an en masses environment with 12 nodes or more. The delay begins to increase slowly for scenarios with more than 32 nodes. This is due to the contention created by increased number of nodes. For instance, when there are multiple nodes conducting Inquiry Scan, more than one node may hear the ID packets sent out by a particular inquiring node. Both scanning nodes will then backoff randomly and attempt to scan the packets from the same inquiring node. However, only one of them will succeed prolonging the time taken for the other

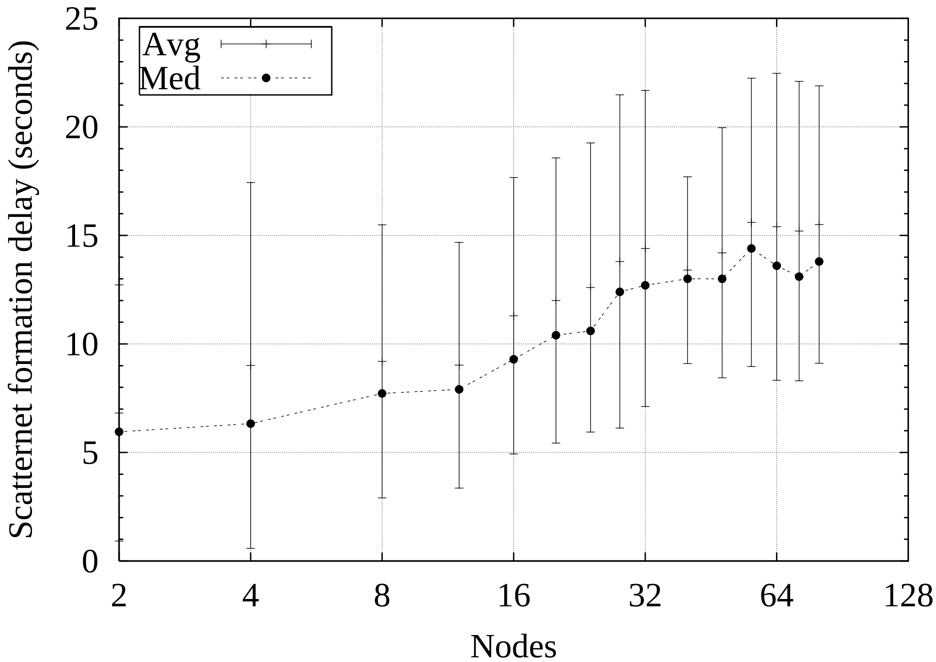


Figure 2-9: Scatternet formation delay (en masse).

node to connect to a different node. This kind of missed opportunities becomes more frequent with a larger number of free nodes. Note that the connection delay of $1s$ achieved by TSF is not too far off from the ideal expected time $E[T_{conn}^{ideal}] = 0.34s$ to establish a connection[3].

Scatternet Formation Delay

As discussed in Section 2.2, 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. Figure 2-9 plots the median and average delays, with error bars that show the standard deviation, taken by TSF to build a scatternet for n nodes arriving en masse and 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.

[3]When the clocks of two nodes are synchronized, $E[T_{conn}^{ideal}]$ is dominated by the random backoff interval T_{rb} between 0 and $0.64s$.

Although one expects the delay curve to be non-decreasing, there are a few small anomalies (e.g., at 56 nodes). This is because of the high variance in the data sets, as seen by the large error-bars on the graph.

Time Spent in Discovery Modes

Figure 2-10 plots the percentage of time that a node spends in discovering neighbors (Inquiry process) and establishing connections with them (Page process) before and after a connected scatternet is formed. Not surprisingly, the percentage decreases with the increase in scatternet-size. As explained in Section 2.1, 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. We also note that our coordinator selection scheme presented in Section 2.2.2 periodically elects a new coordinator and thus, distributes the social task of discovering other coordinators over all the nodes.

Comparison with Other Schemes

It is difficult to quantitatively compare TSF's performance with the two previous schemes [30, 22] 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. Salonidis et al. assume that the time average taken to complete the Inquiry process $E[T_{inq}^{opp}]$ is $0.34s$ which is equal to $E[T_{inq}^{ideal}]$. This assumption implies that nodes have synchronized clocks and therefore, T_{inq}^{opp} is clearly dominated by the maximum random backoff interval between 0 and $0.64s$. We decide not to use this assumption for two practical reasons. First, even when nodes are arriving *en masse*, the only way to have all the 27-bit clocks synchronized will be to turn off all the devices and turn them on at the same time. Second, absolute clock synchronization is undesirable since the phase in the Inquiry hopping sequence is determined by native clocks of the devices. Therefore, the higher the number of nodes with synchronized clocks performing Inquiry simultaneously, the more collisions and longer connection setup delay. We do not, however, see any discussion on collisions in [30]. We believe that Law et al. [22] make a similar assumption of synchronized clocks as Salonidis et al. do. For the aforementioned reasons, we decide to assign a random 27-bit clock value for each device during the simulation.

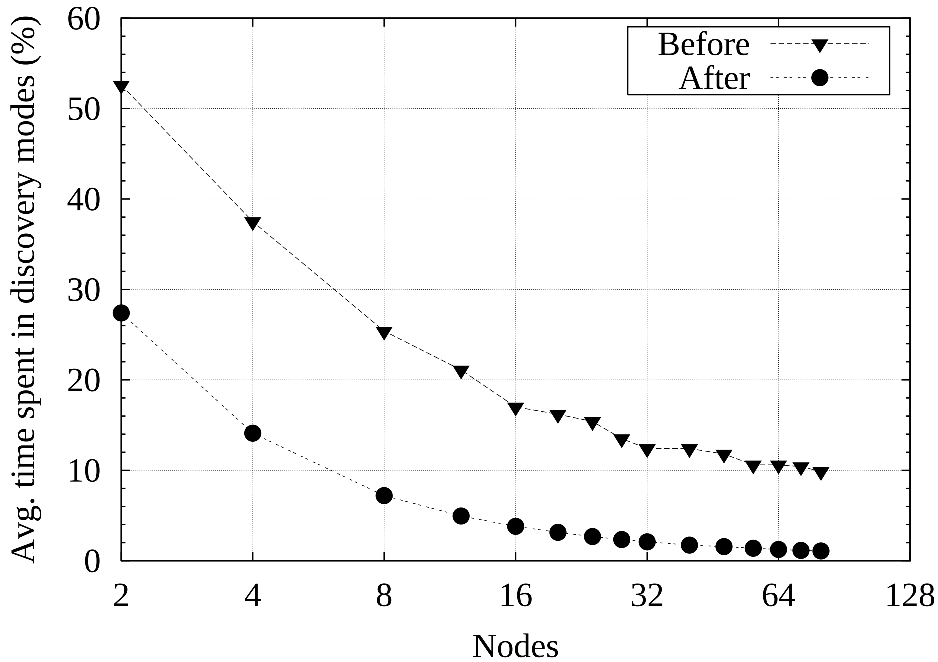


Figure 2-10: Avg. time spent in Inquiry and Page related modes as a percentage of total time before and after the connected scatternet is formed.

We conclude that a sensible comparison between the three schemes will be to compare the scatternet formation delay in terms of round which is simply defined as $E[T_{inq}^{opp}]$. As discussed in 2.3.2, through simulation, we found that the expected time to complete the inquiry process, $E[T_{inq}^{opp}] = 1.8s$. For the other two schemes, we use $E[T_{inq}^{opp}] = 0.34s$ as explained before. Figure 2-11 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 [30] and [22]. Based on our assumptions, 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 [30]. 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 [22], 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.

2.3.4 Incremental Arrivals and Departures

In this Section, we analyze the performance of TSF in dynamic environments where nodes arrive and depart arbitrarily. In dynamic environments such as stores and

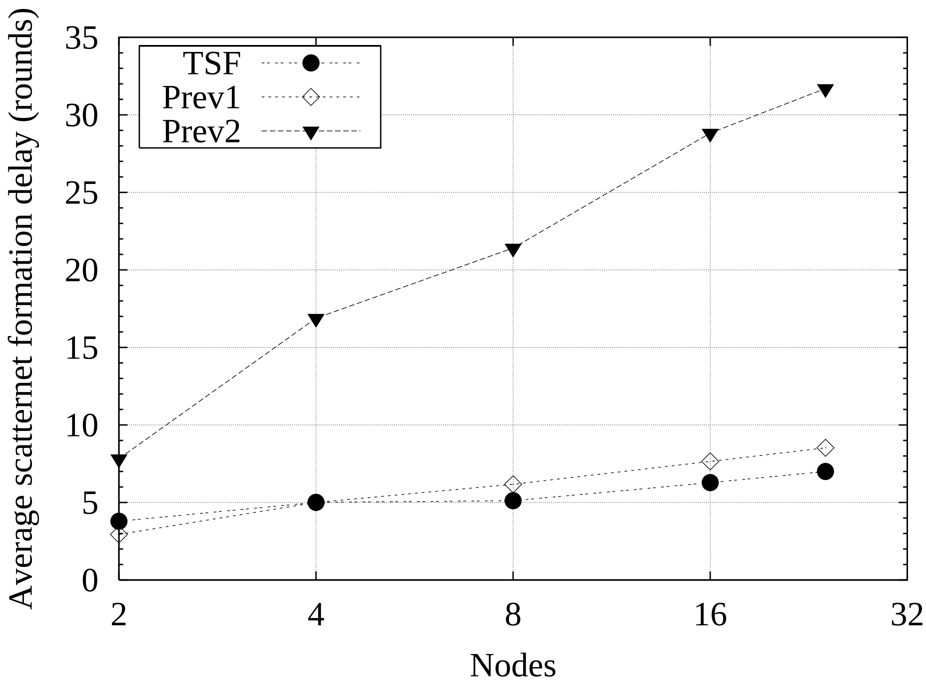


Figure 2-11: Comparison between three scatternet formation schemes. Prev1 and Prev2 are developed by Salonidis et al. and Law et al. respectively.

coffee shops in malls and airports, 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 works do not handle dynamic environments where nodes are arriving and departing arbitrarily.

Link Establishment Delay

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 2-9, 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. An interesting observation here is that the link establishment delay of $2.5s$ is significantly larger than the average delay of $1s$ experienced by 16 or more free nodes arriving en masse. The reason is, when nodes are arriving *en masse*, free nodes in the Inquiry Scan mode may get connected to some other free nodes in the Inquiry mode. However, in the incremental arrival scenario, half of the free nodes are expected to start in the Inquiry Scan mode and will not get connected to a non-root node until they change to the Inquiry mode. This also explains a relatively high variance in the data set since the other half of the nodes are expected to start in the Inquiry mode establishing connections very quickly.

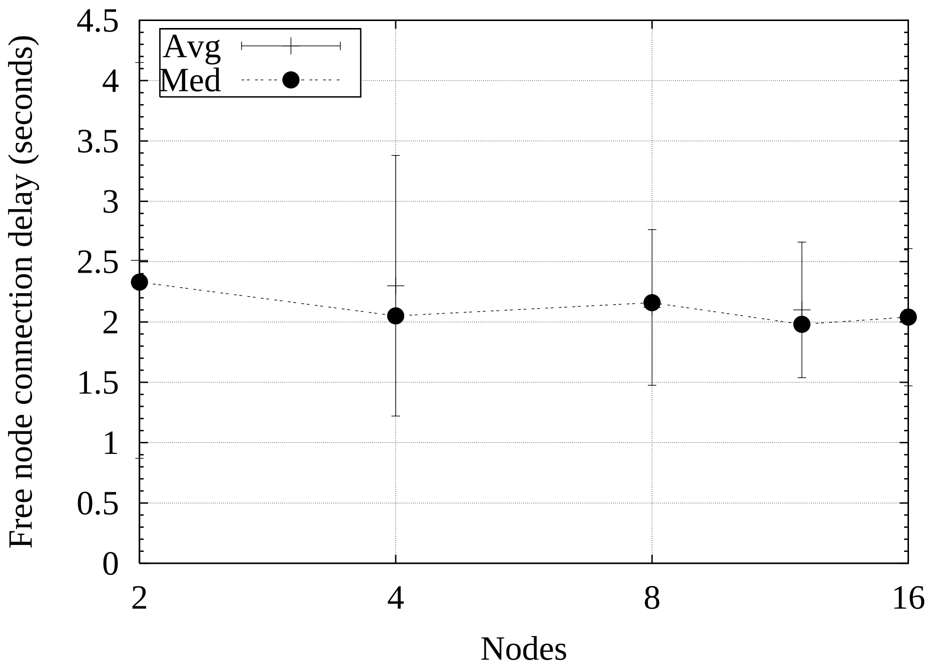


Figure 2-12: Delay for a free node to connect to a 32-node scatternet.

Healing Delay

When nodes arbitrarily leave, the scatternet will be partitioned into several smaller networks. In this Section, we measure how quickly TSF heals network partitions. As explained in Section 2.2.3, 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 2-13 shows that TSF heals the network partitions logarithmically with the number of partitions.

For simplicity, we assume that network partitions are detected by various nodes simultaneously. This is, of course, not true in reality where neighboring nodes of a departing node may detect the link connectivity loss at different times. Nevertheless, the variation is relatively small compared to the healing delay and does not significantly affect our analysis.

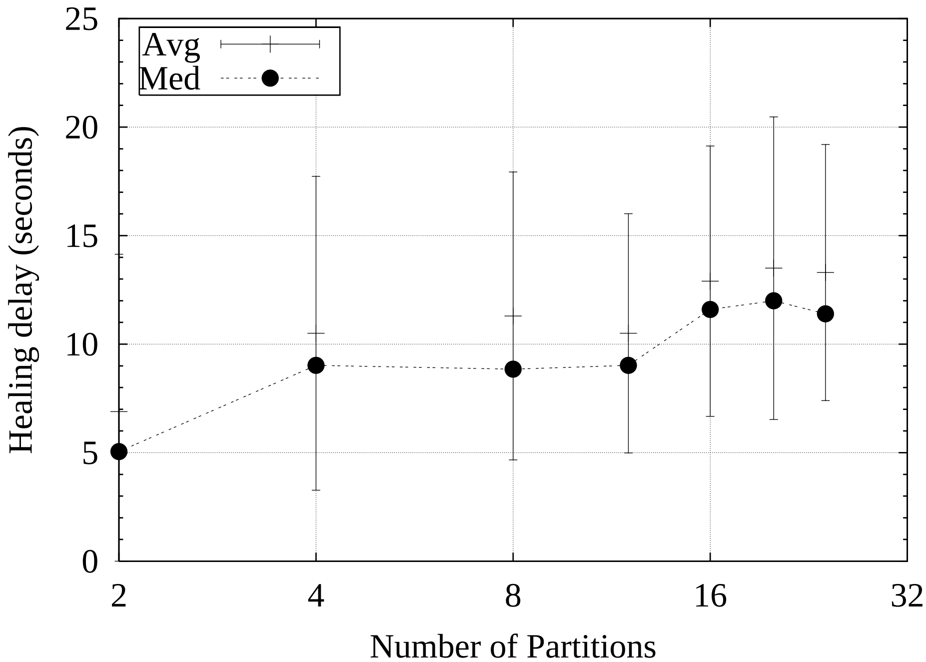


Figure 2-13: Delay for merging multiple scatternet trees.

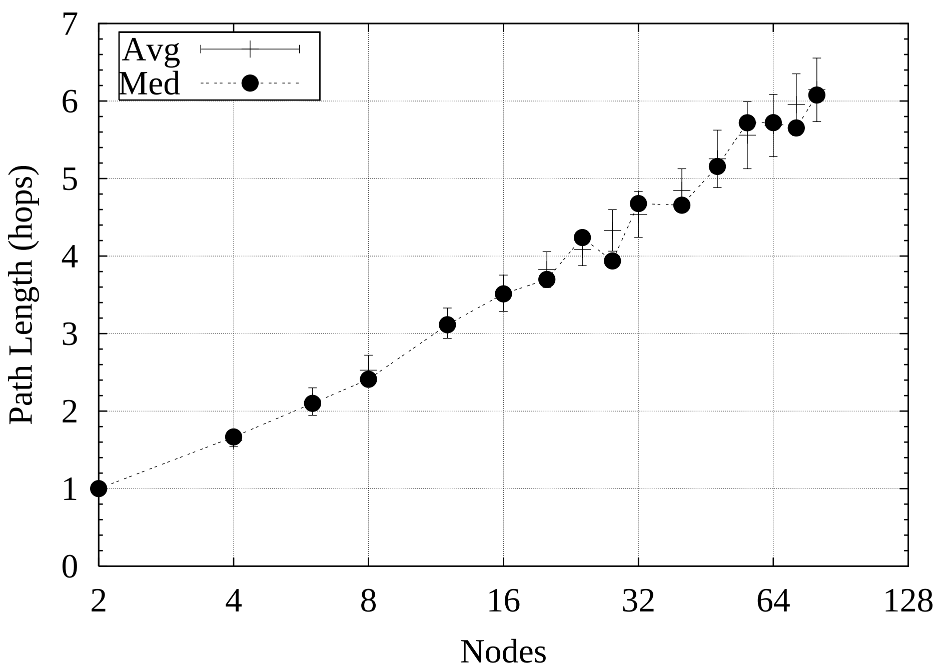


Figure 2-14: Path length for all source and destination pairs.

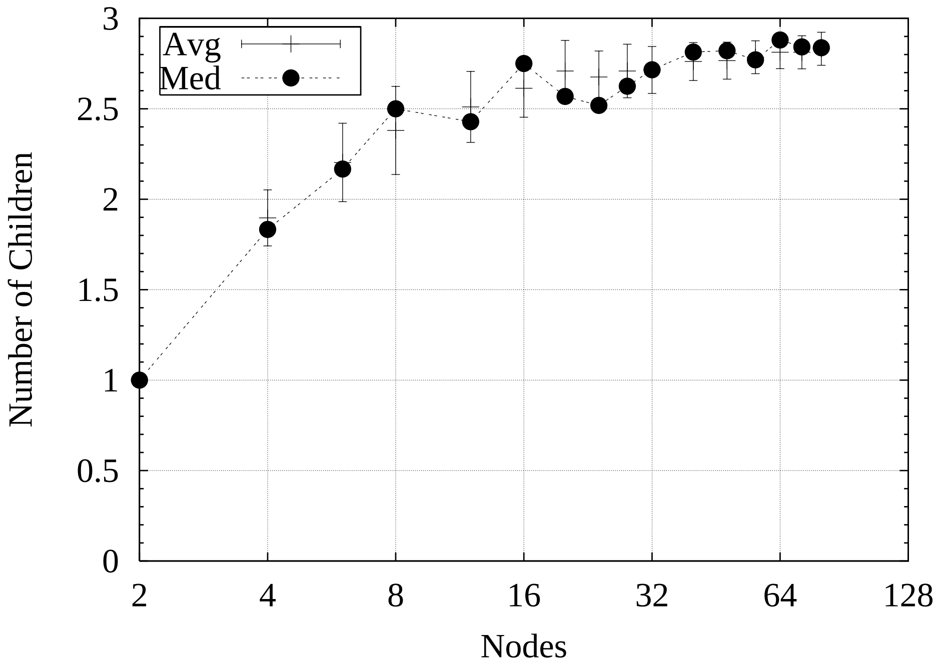


Figure 2-15: Number of children in tree scatternets.

2.3.5 Topology Properties

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., throughput, goodput and latency. We choose communication latency as an important metric to determine the efficiency of Bluetooth scatternets made up of low-bandwidth links. The link scheduling algorithms are critical in optimizing the communication latencies between various node pairs. Nevertheless, in multi-hop networks, the path length or hop count between communicating nodes dictates the end-to-end latency.

Figure 2-14 shows that the average path length grows logarithmically with the number of nodes contained in the scatternet. We plot the average number of children in scatternet trees of various sizes in Figure 2-15. For scatternets with 16 nodes and more, the average number of children is around 2.7.

2.4 Summary

This chapter 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 previous work, our design does not restrict the number of nodes in the network, and also allows nodes to arrive and leave at arbitrary times, incrementally building the topology and healing partitions when they occur. The incremental scatternet formation process allows an arriving node to begin communication with neighboring nodes in the component as soon as it is connected to it.

Our simulation results show that TSF has low tree formation latency, logarithmic in the number of nodes, and better than previous schemes. In addition, TSF achieves low average link establishment delay of $1s$ and $2.5s$ with respect to nodes arriving both *en masse* and incrementally while requiring each node to spend just a small amount of time discovering neighbors. Our results also show that TSF generates tree topologies where the average path length between all node pairs grows logarithmically with the number of nodes. Furthermore, we describe how TSF can be implemented within the Bluetooth specification.

Chapter 3

Link Scheduling

Scheduling communication links in Bluetooth scatternets presents an interesting challenge. Two properties of scatternets make this task a difficult one. First, devices in a Bluetooth piconet communicate using a centralized polling scheme organized by the master in a time-slotted system. A slave is allowed to transmit only if it has been polled by the master in the preceding time slot. Thus, the master not only resolves contention but also allocates bandwidth among slaves. The master may also broadcast to slaves in its piconet. However, unlike unicast packets where every packet sent is acknowledged using the Automatic Repeat Request (ARQ) scheme, broadcast packets are unreliable since they are not acknowledged by any receivers. Second, piconets are interconnected via relay nodes that participate in multiple piconets on a time-division basis. Therefore, every Bluetooth node needs to multiplex its time over all adjacent master-slave communication links. An efficient scheduling algorithm must take into account intra-piconet scheduling as well as inter-piconet scheduling.

Since Bluetooth networks are not yet widely deployed, no one knows what the traffic patterns will be on these networks. Nevertheless, we believe that a scatternet scheduling algorithm must work well for two general classes of application: i) delay sensitive and ii) throughput sensitive applications. Delay sensitive applications range from simple applications controlling Bluetooth enabled mice to voice and streaming applications. These applications send fixed-size application data units (ADUs) or packets [1] periodically and have worst case per-packet delay requirements. We define the per-packet delay perceived by applications as the time between when the source application hands over a data packet to the Bluetooth module for transmission and when the destination application receives that packet. The delay sensitive applications are usually not bandwidth-intensive since ADUs tend to be small in size and are not generated rapidly by the applications.

Throughput sensitive applications include various file transfer applications such as FTP and email applications. Throughput is defined as the number of application payload bytes transmitted per second. Throughput sensitive applications are concerned with transmitting large ADUs as quickly as possible. They tend to use reliable transport protocols and are bursty in nature. These applications are more

[1] Note that an ADU may be segmented into multiple Bluetooth Baseband packets.

concerned with the average throughput available than the per-packet delay perceived. For instance, an FTP application does not really care about how long each packet takes to get to the destination but only on when the last packet arrives the destination. A throughput sensitive connection, however, may time out if a packet is not received by the destination in a certain finite period.

Considering the differing needs of applications, we define the following four main criteria to evaluate the effectiveness of a scheduling algorithm:

1. Average throughput available to applications,

2. Average, worst-case, and 90-percentile delay perceived by applications,

3. Power efficiency, and

4. Fairness.

We measure power efficiency as the ratio of the number of application bytes a node transmitted (and received) successfully to the time the node has been active in seconds. Assuming that a constant amount of energy is required for a node to be active, this metric measures the amount of useful work done per energy unit. For simplicity, we do not distinguish between energy usage related to computation and that of communication. A scheduling algorithm is considered efficient if it maximizes the average available throughput and power efficiency while minimizing delay. The scheduling algorithm must also preserve some form of fairness among all nodes and applications. In particular, we require the scheduling algorithm to provide fair bandwidth allocation among all traffic flows between different source and destination pairs sharing a particular link.

We first examine the difficulty in developing an efficient scheduling algorithm and discuss how to allocate the link bandwidth fairly later. To achieve the overall efficiency of the scatternet, we must minimize the scheduling delay at every link. We identify the following four main factors that hamper the efficiency of a scheduling algorithm:

1. Missed communication events when only an end node is active on a link while the other is busy communicating on another link,

2. Missed communication opportunities when two end nodes fail to recognize that they could communicate,

3. Wasteful communication events when no application data is transmitted, and

4. Frequent piconet switches conducted by bridge nodes. This is undesirable because of the so called bridging overhead as a result of the master clocks not being synchronized.

We examine how these factors affect the throughput, latency, and energy usage of the end nodes. In the case of missed events and opportunities on a communication link, we are assuming that there is data to be exchanged on that link. During a missed communication event, an end node wastes energy by being active on the link where no communication happens. Similarly, both end nodes, which are not doing any useful work, may fail to recognize that they could communicate. Such missed opportunity results in loss of throughput and increased delay since useful data could have been transmitted. On the other hand, both nodes may be active on a communication link yet they have no useful data to be exchanged wasting energy. Lastly, a bridge node, which participates in more than one piconet, may frequently switch between multiple piconets when forwarding data.

Thus, it is important to coordinate end nodes communicating on multiple links to improve efficiency. We distinguish between two possible ways of coordination: static and dynamic schemes. In a static scheme, a scatternet-wide link schedule is setup so that end nodes on a Bluetooth link communicate based on the fixed schedule. The static scheme coordinates the scatternet-wide communication by dictating how each node communicates with its neighboring nodes. The static schedule also determines the fixed duration for two nodes to communicate. Under some conditions, the static scheme can reduce missed communication events and opportunities as well as the number of piconet switches. However, this is not true under varying traffic conditions since the static schedule does not take into account the traffic present on each link. This can result in missed communication events and opportunities alongside with wasteful events. Thus, the static scheduling schemes are not suitable for a majority of the real world scenarios, where network traffic conditions vary. Hence, a responsive and dynamic online scheduling algorithm is necessary to improve the overall performance of the scatternet.

3.1 Background

As explained in the previous section, every Bluetooth node communicates with another node on its adjacent link on a time division basis. Thus, the greater the number of parallel communications in a connected scatternet, the higher the overall throughput of the system. On the surface, the Bluetooth link scheduling problem is similar to the online version of finding a matching with maximum weight. Although much theoretical research has been done on the offline version of the problem [14, 15, 25, 3], relatively few online algorithms exist [21, 4]. We first examine related research work and explain the differences between the problems it addresses and our problem later in the section.

Kalyansundaram and Pruhs [21] provide an optimum strategy for the online maximum weighted matching of complete bi-partite graphs. In their setup, one bi-partition of the graph is designated as the server vertices and the other the request vertices, with each bi-partition having cardinality k. The weights on the edges are then revealed online at k different times. During the i^{th} time interval, the weights of all the edges incident on the i^{th} request vertex are revealed, and one unmatched server vertex is selected to match the request. Their proposed greedy strategy picks the edge with the maximum weight to handle the current request. Its performance is *3-competitive* meaning that their online algorithm is 3 times worse than the offline optimal algorithm in the worst case.

Berenbrink *et al.* [4] present several competitive strategies for scheduling real-time requests in distributed data servers. A set of requests arrives at the system every round consisting of an equal number of time steps. Each request arriving at round, t, specifies exactly two distinct resources but requires to get access to one of them at the latest during round $t + d - 1$. The goal is to maximize the number of requests fulfilled before the deadline expires. Of course, every resource can only fulfill at most one request per round. They present several global strategies as well as two local strategies where every new request does not know anything about other requests and their alternative resources at the beginning. To make reasonable decisions, the requests and resources are allowed to communicate with each other by exchanging fixed size messages during the communication rounds. They assume that up to d messages can reach a resource in each communication round and that the senders of the dropped messages will be informed of the failures in the same round. The performance of these local strategies is measured in terms of communication rounds. The two strategies, A_{local_fix} and A_{local_eager}, require 2 and 9 communication rounds and are *2-competitive* and *5/3-competitive* respectively.

The link scheduling problem in Bluetooth scatternets is not the same as the maximum weighted bi-partite matching problem tackled by Kalyansundaram *et al.* The difference is Kalyansundaram *et al.*'s assumption of communication between the request and the server vertices during matching. When scheduling communication links in a scatternet, node j does not know the arrival of request $r_{i,j}$ at node i and vice versa, and the two nodes may not be synchronized for communication. Thus, some communication mechanism needs to be set up during bootstrap and throughout the network lifetime and its overhead must be considered when evaluating the performance of the scheduling schemes. Similarly, the local scheduling problem solved by Berenbrink *et al.* assumes that the system can carry up to d messages to a resource in each communication round. Again, in our case, we need to consider how the two nodes will be synchronized to be active on the common channel simultaneously before any data transfer can occur.

There has been little research work done for scatternet-wide link scheduling. Racz et al. presents a pseudo random scheduling scheme (called PCSS) for Bluetooth scatternets [28]. In PCSS, every node randomly chooses a communication checkpoint for a particular link. When both end nodes show up at a checkpoint simultaneously, they can communicate until one of the nodes leaves to attend to another checkpoint. Checkpoints are randomly selected based on the current master's clock, the slave's device address and the base checking period, which is the expected interval until the next checkpoint occurs. In order to adapt to various traffic conditions, PCSS measures the link utilization in a simple way and adjusts the checking period accordingly. The advantages of PCSS is the lack of scatternet-wide coordination in scheduling links and its dynamic adjustment of the checking period according to link utilization. However, since PCSS is based on a randomized scheme, there will be occasional collisions among various checkpoints resulting in missed communication events. More importantly, PCSS is not very responsive to bursty traffic. PCSS can only increase or decrease the interval between two successive communication events on a particular link by a multiple of 2. This rigid behavior combined with the lack of coordination between end nodes on a particular link makes PCSS respond poorly to varying traffic conditions.

In contrast, our scheduling algorithm TSS is responsive to various traffic loads and dynamically schedules the communication links based on current traffic conditions and heuristics. TSS adjusts the durations for end nodes to communicate on a particular link as well as the intervals between two successive communication events on that link according to the changing traffic loads. TSS also has low communication overheads since coordination is only done locally at the piconet level instead of at the scatternet level. In addition, TSS tolerates disruption in connectivity by providing a fall-back communication mechanism without requiring nodes to coordinate explicit. Finally, TSS inter-operates effectively with our scatternet formation algorithm, TSF, providing a complete solution to realize Bluetooth scatternets.

3.2 TSS: Tree Scatternet Scheduling

TSS coordinates communication among neighboring nodes while allocating bandwidth judiciously based on current local traffic conditions. TSS improves the overall efficiency of the scatternet by significantly reducing missed communication events and opportunities, piconet switches, and wasteful communication events.

TSS is based on the concept of scheduled meetings or appointments. End nodes on a particular link meet according to their appointment and exchange data during the meeting. Before the meeting is terminated, the two nodes schedule when their next meeting will occur and how long it will last at the minimum.

Before we continue with the details of the algorithm, we review the properties of scatternets produced by our topology construction algorithm, TSF. The scatternet is a tree structure, which guarantees loop-freedom and the existence of at most one parent for any node in the scatternet. In addition, a parent node always acts as master and its children nodes are slaves. TSS exploits this hierarchy to converge quickly to a mutually agreed appointment between two nodes. Specifically, at the end of every meeting, a master suggests a list of possible meeting times and durations to a slave. The slave replies with desired start and finish times of a future meeting, which fall between one of the meeting periods suggested by the master.

Each node negotiating for a future meeting attempts to achieve the following goals:

1. The meeting will begin at the earliest time possible, when there is data to exchange, without canceling existing meetings.

2. The duration of the meeting is long enough to exchange the existing data queued at each node.

3. The meeting is scheduled so that it preserves some form of fairness among all neighboring nodes that want to communicate with this node.

In the following sections, we discuss in detail how to schedule a future meeting to achieve the aforementioned goals.

3.2.1 Bootstrap and Termination

As soon as two nodes establish a new link, they remain active on that link for a fixed period. During this period, the nodes follow connection establishment procedures required by a higher Bluetooth layer such as L2CAP for future data exchange by various applications. The nodes communicate until the current meeting is terminated for one of two reasons: i) an end node needs to attend to another meeting or ii) there is no more data to exchange. In the first case, the meeting has lasted for at least the agreed duration since TSS does not schedule overlapped meetings. In the second case, the agreed duration is longer than necessary and both nodes enter the Standby mode to save power. In either case, nodes negotiate their next meeting as explained in the next subsection. It is important that they do this even if they have no data to exchange so that they can meet again.

3.2.2 Negotiating a Future Meeting

The critical part of TSS is how to schedule future meetings. Nodes negotiating for a future meeting must eventually agree to i) the *start* time and ii) the *duration* of the meeting. It is also important that the scheduling of a new meeting does not result in cancellation of existing meetings committed by both end nodes since doing so will result in missed communication events.

To keep track of existing meetings, each node keeps an ordered list of tasks. A *task* is defined as a 2-tuple (s^l, f^l) where l is the communication link with which the task is associated and (s^l, f^l) denote the start and finish times of the task. To negotiate a future meeting between master A and slave B, A sends out a list of possible future meetings denoted by their start and finish times. During negotiation, start and finish times are always based on the master's clock since native clocks are not synchronized. B then picks a suitable meeting without requiring it to cancel any existing meeting, and informs A of the desired meeting time and duration. Both nodes modify their task lists accordingly and terminate the meeting. Sometimes, B may not be able to pick a meeting from a list suggested by A without canceling the existing meetings. To avoid this situation, A includes the finish time of the last task from its task list. Thus, B can always pick a meeting time later than the latest finish time of A's tasks.

It is clear that how master A selects future meeting times and durations effects the overall efficiency of the scheduling algorithm. The start time of a future meeting and its duration depend not only on the current and future traffic loads associated with the link but also on that of other adjacent links at the end nodes. Scheduling the meeting at the earliest time may result in a waste of resources when there is no data to exchange. A timely meeting is, therefore, necessary to reduce end-to-end packet latency. Similarly, scheduling a long meeting could result in a waste of bandwidth if there is not enough data to be exchanged during that period. On the other hand, a short meeting increases the bridging overheads as nodes switch between piconets more frequently.

To deal with dynamic traffic conditions, each node monitors the utilization of every adjacent link and makes an informed decision for the start time and the duration of a future meeting. *Utilization* is the ratio of the number of time-slots used to transmit data packets to the duration of a meeting. TSS maintains two moving averages for each link: i) the number of data units transmitted during a meeting, tx_slots, and ii) the interval between two successive meetings, *period*. Both variables are in time-slots and are updated before the negotiation for a future meeting begins as follows:

$$tx\_slots = cur\_tx \times K_c + (1 - K_c) \times tx\_slots \tag{3.1}$$

$$period = cur\_per \times P_c + (1 - P_c) \times period \tag{3.2}$$

cur_tx and cur_per correspond to the number of slots used to transmit data during the on-going meeting and the interval between the last two meetings respectively. K_c and P_c are scaling constants (between 0 and 1) for the corresponding exponentially weighted moving averages. To negotiate, the master picks the start time of the earliest future meeting as follows:

$$start = min(P_{max}, max(P_{min}, period)) \tag{3.3}$$

P_{min} and P_{max} represent the lower and upper bounds of the interval between two successive meetings. A link will become active at least every P_{max} time-slots even if there is no data to exchange. Although doing so could potentially result in wasteful communication events, this is necessary to ensure that nodes can communicate again. P_{min} dictates the earliest time before the future meeting could begin.

The future meeting duration depends on whether there is any data left in either end node's queue as shown in Equation 3.4. If there is no data left, the duration is simply the sum of the minimum duration, D_{min}, and the average time slots needed to exchange data. Otherwise, the duration is increased according to the current utilization of the link denoted by *util* which is the ratio of cur_tx to the duration of the on-going meeting. α is a constant between 0 and 1 and dictates how fast a link's duration gets increased as it becomes busier.

$$duration = \begin{cases} D_{min} + tx\_slots & \text{if no data in the queues} \\ D_{min} + tx\_slots + (util \times cur\_tx \times \alpha) & \text{otherwise} \end{cases} \tag{3.4}$$

As mentioned before, either master or slave may request to terminate the on-going meeting to attend another meeting or when there is no data left. The latter is detected by the master node when it receives a *NULL* packet from a slave to which it has sent a *POLL* packet (referred to as the POLL-NULL sequence). In either case, the negotiation begins by sending a *REQ* packet. If master receives a *REQ* packet, it ignores the packet and sends a *REQ* packet back to the slave for reasons explained shortly. The *REQ* packet sent by the master contains N_m future meeting times and the duration which is the same for all meetings. The slave then picks the most suitable meeting time and duration and replies with a *REP* packet. For each pair of start and finishing times suggested by the master, the slave checks its task list to see whether there are at least D_{min} free time-slots within that period. The slave may choose the earliest meeting among all the possible ones. However, the earliest meeting may be much shorter than the duration requested by the master. Instead, the slave chooses the meeting with the longest duration. The intuition for this decision is that choosing the meeting with the longest duration potentially reduces the gaps between successive tasks in a node's task list, which translate to loss of bandwidth. Recall that scatternets constructed by TSF have tree structures. The design decision to use the longest duration combined with the requirement to have the master suggest meeting times allows nodes in tree scatternets to converge to an efficient inter-piconet link schedule. The root schedules its adjacent links in succession while its child nodes attempt to schedule their child links accordingly. Overtime, all relay nodes begin to settle on an efficient link schedule as different subsets of links are coordinated to communicate simultaneously in a top-down fashion. Intuitively, TSS strives to select a maximal set of links for simultaneous communication to increase the tree scatternet's overall performance.

3.2.3 Tolerating Faults

In practice, links may temporarily fail due to crashes or mobility. Therefore, an end node may not show up during the agreed upon meeting period. A master node detects the absence of a slave node when it does not receive any response after polling it for N_{poll} times consecutively. Similarly, a slave node assumes that the master is absent after not being polled in N_{poll} consecutive even slots [2]. This grace period is necessary to cater for clock drifts especially when two nodes haven't met for a long period.

When end nodes do not meet in the agreed meeting period, they both reschedule the future meeting automatically based on the last agreed meeting start time, s_{last}. Each node chooses an appropriate future meeting beginning at $s_{last} + k * T_{wait}$, where T_{wait} is the waiting period and $k > 0$. Thus, at some point in the future, end nodes meet again and communicate as before. If a node detects that the other end node misses the meeting for N_{miss} times, it will assume that its counterpart has disappeared and disconnect the Bluetooth link. If a node desires to re-connect to the scatternet, it must again go through the discovery procedures required by the topology formation scheme.

3.2.4 Fairness

Several existing algorithms such as Fair Queuing [12] and Deficit Round Robin [31] are used to provide fairness among all competing flows going through routers. Both schemes require proper classification of flows and significant amount of memory and thus, may not be suitable for Bluetooth devices with small memory. Fair bandwidth allocation in Bluetooth is further complicated by the facts that forwarding nodes communicate on its adjacent links on a time division basis. We are investigating in integrating a suitable fair queuing algorithm with our scheduling scheme. In particular, we plan to modify DRR to suit Bluetooth scatternets. We currently use RED

[2] Recall that master transmits in even slots and slaves in odd slots.

(Randomly Early Detection) queues to detect congestion early and perform random drops. This results in packets being dropped from several flows, as opposed to a single flow in some cases if packets were dropped from tail.

3.2.5 Interoperatability with Scatternet Formation Schemes

It is important for a link scheduling scheme to inter-operate well with scatternet formation schemes. As described in Chapter 2, a scatternet formation scheme dictates how nodes carry out discovery operations such as Inquiry and Inquiry Scan to establish communication links with their neighbors. The scatternet formation scheme specifies the start and finish times of each of these operations for the lower layers to carry out.

TSS coordinates both communication tasks, called Comm tasks, and scatternet formation related tasks, called Form tasks. Since scatternet formation schemes operate independently from TSS, scheduling conflicts may arise between different types of tasks. For instance, a scatternet formation scheme may force a node to enter the Page Scan mode as soon as it discovers a neighbor node in the Inquiry Scan mode. However, the existing communication events may prevent a node from carrying out the Page Scan operation right away. TSS uses the following rules to resolve conflicts between different types of tasks:

1. The ongoing task is never interrupted or pre-empted by another task.
2. A future Form task is scheduled as soon as the current task's finish time has reached. If necessary, all the existing tasks which fall in between the start and finish times of the Form task are rescheduled.

The rules guarantee that every Form task will be scheduled no later than D_{max} since any Comm task takes at most D_{max} slots. Form tasks are given higher priority since carrying discovery operations out in a timely fashion shorten the time required to create a connected scatternet. We note that various policies can be set between a scatternet formation scheme and TSS to resolve scheduling conflicts. For instance, a scatternet formation scheme may only require TSS to apply the second rule if the Form task is to carry out Page or Page Scan operations instead of the Inquiry and Inquiry Scan operations. The existing tasks that are canceled according to the second rule are rescheduled in the same way when nodes are detected absent.

3.2.6 Bluetooth Implementation

In this section, we discuss the implementation of TSS in Bluetooth in detail. TSS plays two different roles: i) as generic task scheduler and ii) as communication link scheduler. TSS serves as a generic task scheduling algorithm by arranging various tasks in a sequential order and interacts with lower Bluetooth layers to carry out appropriate operations at different times. TSS is also an inter-piconet scheduling agent which provisions bandwidth available to each communication link based on traffic loads. We note that a generic task scheduling mechanism is required for third party implementation of various scatternet formation and link scheduling schemes.

The current Bluetooth specification (version 1.1) lacks such a mechanism and make it hard if not impossible for third parties to implement these schemes in an interoperatable way. We also note that the Bluetooth SIG has been working for some time to develop a new version of the Bluetooth specification that can fully support scatternet operations.

On the other hand, vendors of Bluetooth firmware composed of lower layers such as Radio, Baseband and LMP can implement TSS within the existing Bluetooth specification. The only mechanism required by TSS is to activate and deactivate communication links as needed. The Bluetooth specification describes two mechanisms to achieve that goal: *Hold* and *Sniff*. A link can be put on hold by entering the *Hold* mode. To do so, end nodes must negotiate for the interval before the link can be active again, and the negotiation process is carried out by the LMP layers. TSS can be implemented using the *Hold* mode although doing so will require a couple of more packets to exchange for each negotiation process. The *Sniff* mode reduces the overhead of repeated negotiation by requiring nodes to negotiate for the interval between two successive communication events. Thus, end nodes agree to communicate on a particular link at every fixed period. The end nodes need to negotiate again to change the agreed interval. TSS can use the *Sniff* mode to schedule communication links especially when the *period* associated with a particular link is not changing too rapidly. Although TSS can be implemented within current Bluetooth specification, we note that a more responsive link management mode such as the one suggested in [18] is desirable for efficiently scheduling communication links in Bluetooth scatternets.

3.3 Performance Evaluation

We have implemented TSS in our Bluetooth simulator and integrated it with TSF. We conducted simulation runs under various traffic loads and measured average and total throughput available, and average per-packet delay perceived by both throughput intensive and delay sensitive applications. The results show that TSS responds quickly to dynamic traffic conditions and scales well as the number of applications increases. The rest of this section discusses simulation setup and performance results in detail.

3.3.1 Configurations

Table 3.1 shows the TSS parameters we used for all the simulation runs. Note that only D_{min}, D_{max} and P_{max} can significantly affect the communication efficiency of TSS. D_{min} and P_{max} dictate the amount of overhead for maintaining active links with no data to exchange whereas D_{max} determines the maximum communication period on any link. We kept the same Baseband parameters (see Table 2.3) used for TSF's simulation runs. In every simulation run, nodes arrive en-masse. TSF attempts to form a connected scatternet while TSS coordinates link schedule. Applications are started in two different ways since we are interested in the performance of TSS with or without interaction from TSF. For all the simulation runs except for the ones conducted in Section 3.3.5, applications are started only when the steady state is

D_{min}	D_{max}	P_{min}	P_{max}	N_{poll}	K_c	P_c	α
20	100	5	1000	3	0.25	0.2	0.25

Table 3.1: TSS parameters in time-slots.

reached and every node has connectivity to every other node, at which time, TSF stops performing Inquiry and Inquiry Scan operations. Thus, the performance of TSS solely depends on the traffic pattern and the topology of the scatternet. In Section 3.3.5, we analyzed the communication efficiency of TSS when each application is started as soon as there is connectivity between source and destination while TSF continues to conduct discovery operations throughout the simulation period. Recall that TSF's continuous operation is necessary in dynamic environments where nodes come and go arbitrarily. Every data point in the figures is an average of at least 10 runs.

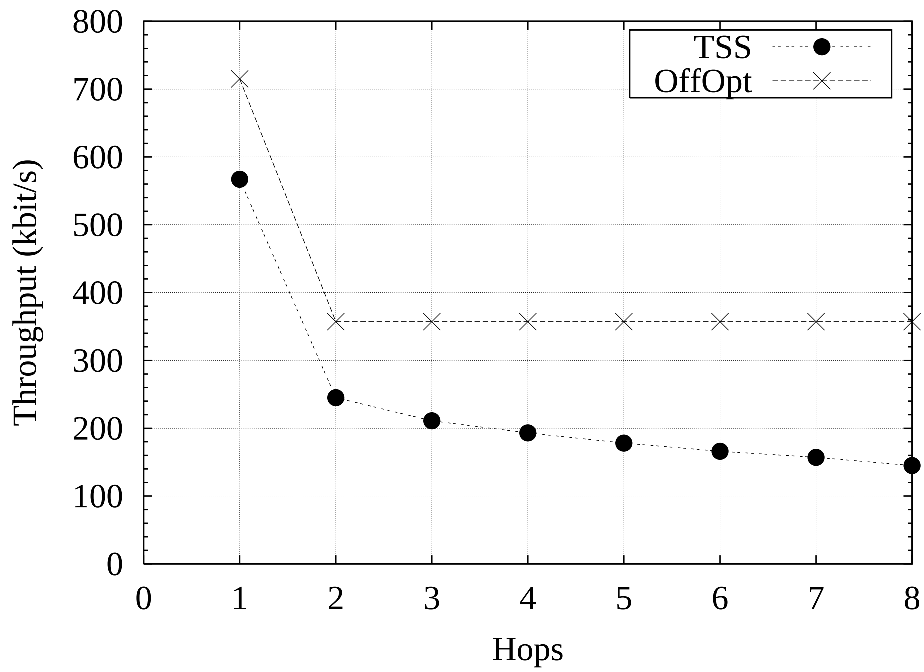


Figure 3-1: Throughput of a CBR application over varying number of forwarding hops.

3.3.2 Effects of Number of Forwarding Hops

In this subsection, we analyze how the number of hops along the communication path impacts the available throughput. We setup a CBR application, using UDP

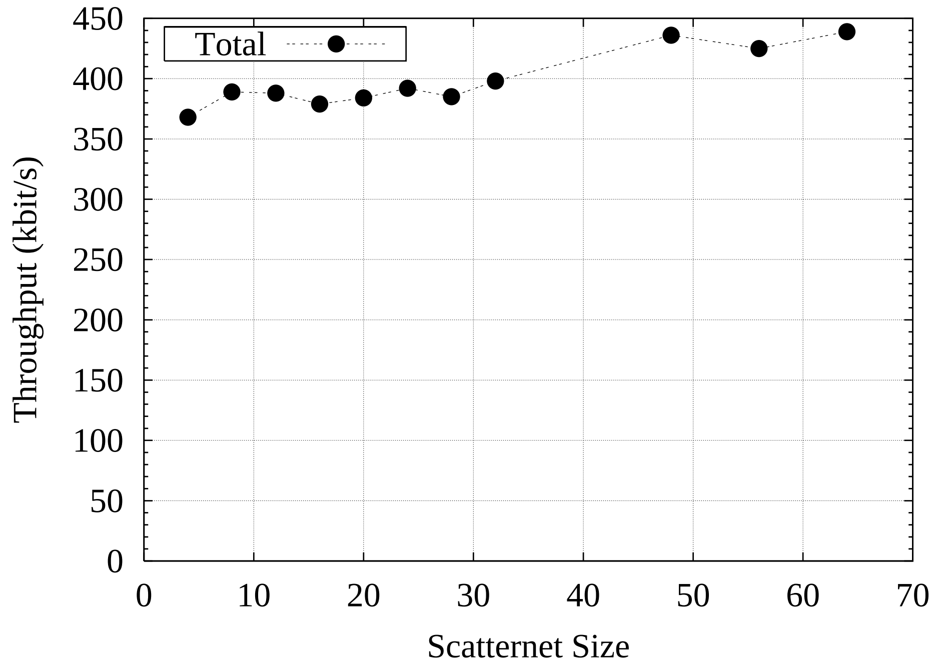


Figure 3-2: Total throughput of FTP applications from a source to all other nodes.

as transport, between a random source and destination pair from a scatternet containing 20 nodes. The application sends data at the maximum data rate, and each packet contains 335 bytes of application data, which is the maximum size permitted by Bluetooth without requiring segmentation and reassembly. Figure 3-1 compares the throughput available to the application (noted as TSS) and the optimal offline throughput (noted as OffOpt). For one hop case, the optimal throughput is calculated as: $335 * 8/(6 * 0.000625) = 715kbps$ since it requires 6 time-slots to transmit one data packet. We can see that TSS achieves 79% of the optimal throughput. The main overhead of TSS in this case is in maintaining other active links that do not have any data to exchange.

The optimal throughput is halved when the number of forwarding hops is more than one. This is because every relay node forwarding packets needs to divide its time between receiving packets from one link and transmitting it over to a different link. Ideally, all forwarding nodes will either be receiving or forwarding data simultaneously and hence, the optimal multi-hop throughput available will be one-half of the optimal one-hop throughput. As shown in the figure, the throughput achieved by TSS decreases slowly as the number of forwarding hops increases. As the number of links adjacent to each forwarding node varies, it becomes harder for TSS to line up perfectly the communication tasks associated at each node, thus increasing the overhead.

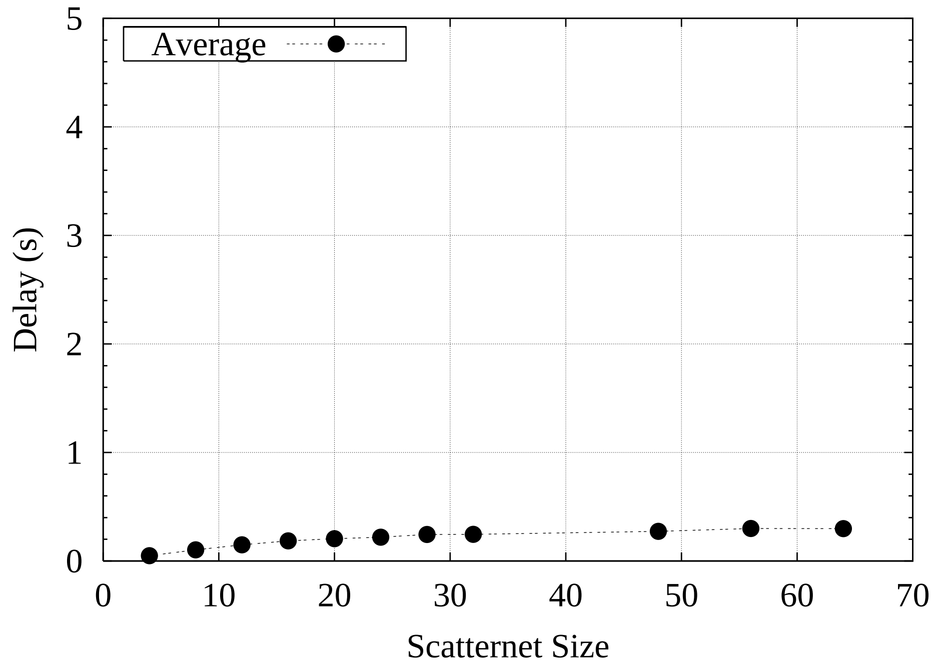


Figure 3-3: Average delay of a FTP packet from a source to all other nodes.

3.3.3 Effects of Scatternet Size

We examine how the scatternet size impacts the total throughput available to a particular source node sending to multiple destinations. We set up FTP applications from a random source node to all other nodes in a scatternet. Again the size of each application data packet is 335 bytes. We ran simulations with several random source nodes on each of several different topologies of the same size to get the average. Figure 3-2 plots the total throughput achieved by the source node. The throughput remains around $375kbps$ for scatternets containing 32 nodes or less and around $470kbps$ for scatternets containing 48 nodes or more. We explain shortly the reasons behind this behavior.

It is less clear to calculate the optimal throughput of an application sending data to multiple destinations. Without loss of generality, let's assume that node A is sending data to all other nodes in the scatternet, which are not sending any data. The critical factor impacting the optimal throughput that A can achieve is the number of neighbors with more than one adjacent link, i.e. non-leaf neighbors. Note that A can achieve the optimal throughput when sending data to its one-hop destinations. If A does not have any leaf neighbor, like 3 in Figure 3-4, the data transmission to its multi-hop destinations, 2, 4 and 7, can be arranged so that 3 achieves the optimal total throughput of $715kbps$. In an ideal situation, a maximal matching of busy communication links are scheduled simultaeneously for a short period of time called *round*. In the first communication round, the links between 3 and 1, 5 and 2, and 6 and 0, are be scheduled simultaeneously. Similarly, the links between 3 and 5, 1 and

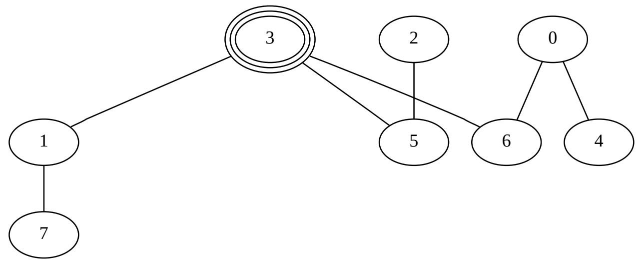


Figure 3-4: A scatternet containing 8 nodes where 3 is the root.

7, and 0 and 4 are scheduled in the second communication round. Lastly, the link between 3 and 6, along with other links already scheduled in the previous rounds, if they still have data, are scheduled at the same time. Clearly, in three rounds each of the multi-hop destinations will receive packets transmitted in a single round. Thus, 3 achieves the optimal total throughput. On the other hand, if A only has a neighbor with more than one adjacent link, like 1 in Figure 3-4, the optimal throughput will be close to the optimal multi-hop data transfer rate of $375.5kbps$ as its neighbor 3 needs to divide its time between receiving packets from 1 and forwarding to other nodes. As the scatternet size increases, tree becomes bushier and there exist less nodes (like 1) with just one non-leaf neighbor. This is the reason why the total throughput achieved by a node sending data to all other nodes in the scatternet increases with the size of the scatternet.

Figure 3-3 plots the average end-to-end per-packet latency perceived by applications and the average number of forwarding hops against the scatternet size. The average delay slowly increases as the scatternet size increases, demonstrating again that TSS scales well.

3.3.4 Interaction with a Mixture of Applications

In this section, we analyze how well TSS accommodates a mixture of bandwidth intensive and delay sensitive applications. We use FTP and CBR applications to represent bandwidth intensive and delay sensitive natures respectively. We setup an equal number of FTP and CBR applications between random node pairs in a scatternet containing 24 nodes. As before, each FTP application sends out data packets of 335 bytes as fast as the underlying data transport, TCP, allows. Each CBR application, on the other hand, sends out 100-byte data packets at a small rate of $10kbps$ using UDP as data transport. Figure 3-5 depicts how average and total throughput available to each class of application varies as the number of applications increases. CBR applications on average achieve throughput close to the sending rate of $10kbps$. Expectedly, the average TCP throughput goes down as the number of FTP application increases. However, the combined throughput available to the FTP applications increases. This is because as more links become busy, there is less overhead to maintain active links with no data to exchange.

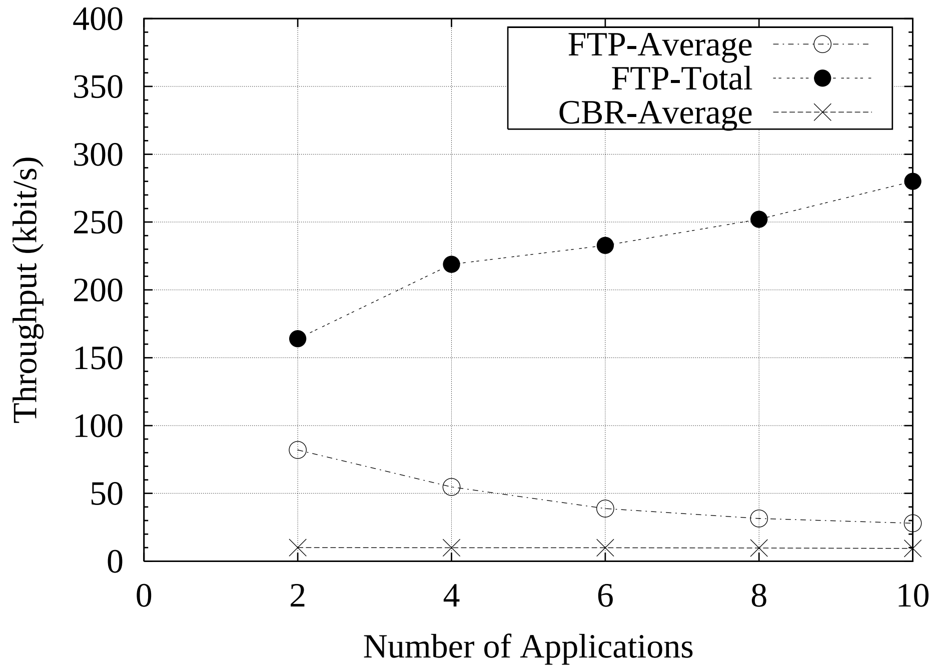


Figure 3-5: Total and average throughput of applications between multiple node pairs.

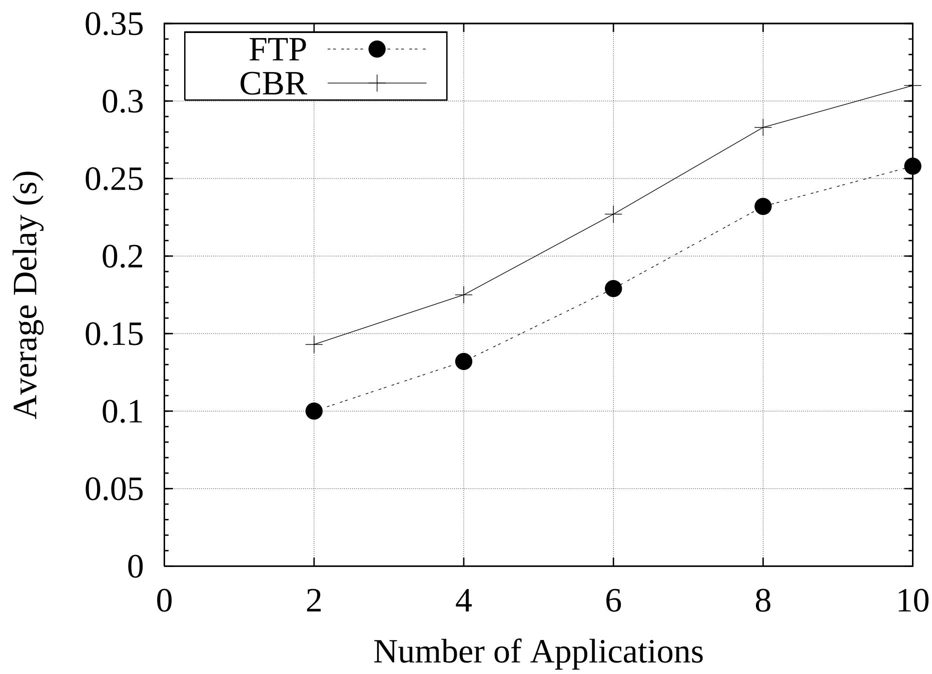


Figure 3-6: Average per-packet delay of applications between multiple node pairs.

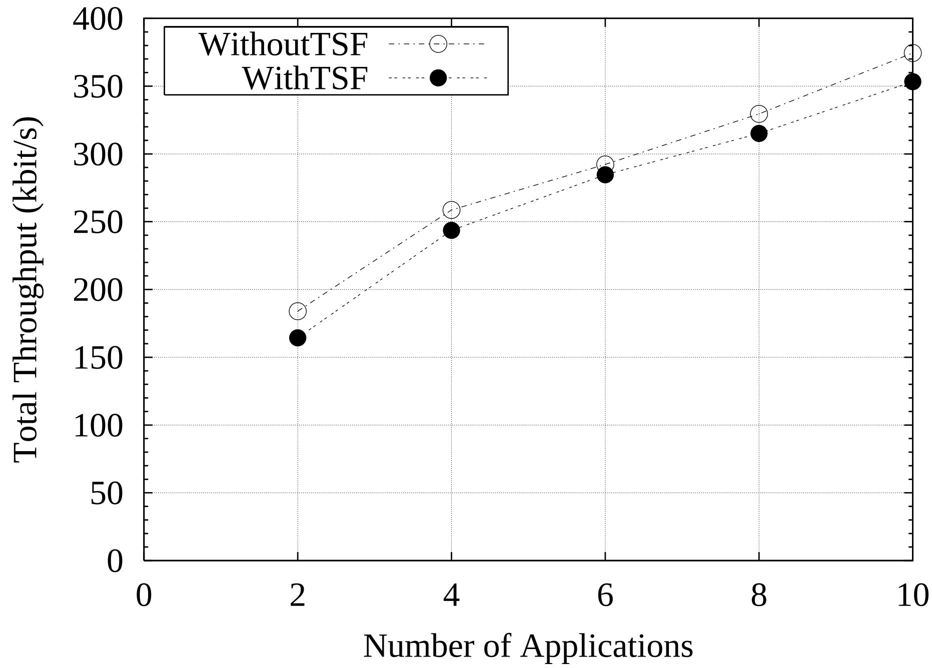


Figure 3-7: Total throughput of applications between multiple node pairs, with or without interaction with TSF.

Figure 3-6 shows that the average per-packet latency increases as the number of applications increases. The reason for this is the increased buffering delay at each forwarding node as more packets are queued. The per-packet latency for CBR applications are higher than that of FTP applications as the underlying data transports react differently to congestion. TCP responds to congestion by holding off the packets and sending it as bandwidth becomes available. On the other hand, UDP sends out data as soon as it becomes available regardless of congestion. Thus, the average buffering delay perceived by a UDP application is larger than that of a TCP one.

3.3.5 Interoperatability with TSF

In this section, we examine how well TSS interacts with TSF. We setup a mixture of TCP and CBR applications between random node pairs in the same way we did in Section 3.3.4 with two exceptions. First, each application is started as soon as there is connectivity between source and destination. Second, each node continues to carry out Inquiry and Inquiry Scan operations even after a connected scatternet has formed. We ran simulations and compare the performance of TSS in this environment against the one in the environment mentioned in Section 3.3.4. Figure 3-7 plots the total throughput available to applications in two different environments: with and without interaction from TSF. Clearly, the difference in total throughput at any point is not significant, showing that TSS inter-operates well with TSF. Figure 3-8

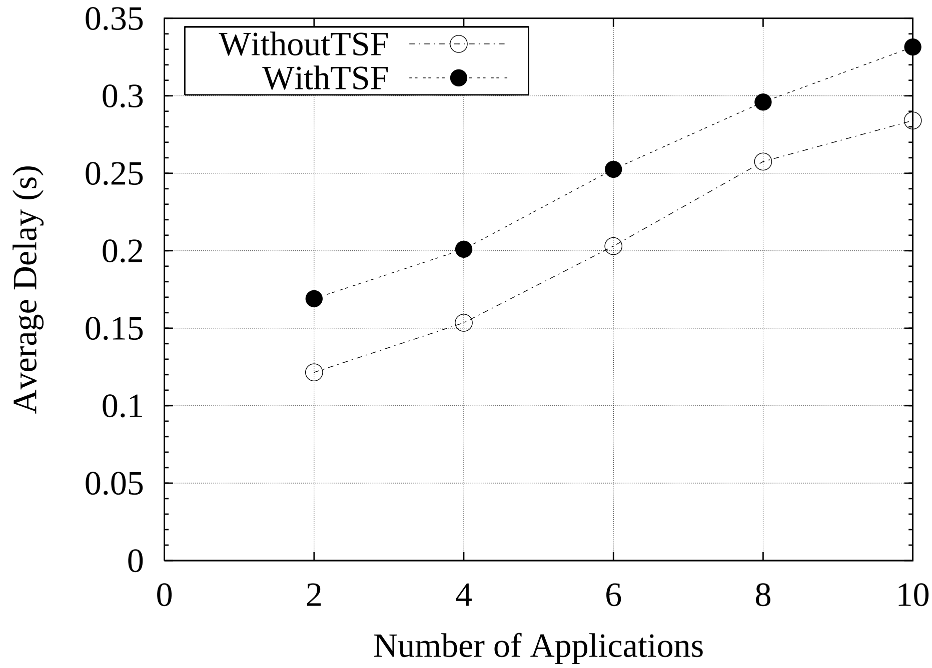


Figure 3-8: Average per-packet delay of applications between multiple node pairs, with or without interaction from TSF.

plots the average per-packet delay applications perceived in both environments. The average delay increases by $0.05s$ when TSF interacts with TSS. This is because during the early stage, nodes on average are spending more time carrying out scatternet formation tasks. As more packets are getting queued in the buffers, the delay for each packet increases. The average per-packet delay before and after a connected scatternet is formed is different with the former being larger. Since applications are started only after all nodes are connected in the $WithoutTSF$ environment, the average delay perceived by applications is smaller than that in the $WithTSF$ environment.

3.4 Summary

In this chapter, we discuss the challenges in scheduling Bluetooth communication links efficiently and present a new coordinated online link scheduling algorithm, TSS. Although TSS can coordinate all communication links in a particular scatternet, its performance is optimized for tree scatternets. TSS exploits the hierarchical nature of the tree by arbitrating scheduling conflicts in a top-down fashion. Unlike the earlier work, TSS is responsive to various traffic conditions by dictating how long and how often end nodes communicate on a particular link according to current and previous traffic loads. TSS also has low overhead since coordination is only done among one-hop neighbors. In addition, TSS tolerates disruption in connectivity by providing a fall-back communication mechanism when nodes are not able to communicate during the agreed upon periods. TSS inter-operates well with scatternet formation schemes by providing an arbitrating mechanism when scheduling conflicts arise between various discovery and communication operations. Finally, TSS coupled with TSF provides a complete novel solution to realize Bluetooth scatternets within existing Bluetooth specification.

Chapter 4

Conclusions

In this thesis, we have addressed challenges in internetworking personal area networks using an emerging low-power, low-cost RF technology, Bluetooth. Bluetooth's use of Time Division Duplex and Frequency Hopping schemes for communication coupled with its point-to-point link formation mechanism makes internetworking Bluetooth piconets different from internetworking traditional wired and wireless LANs. We give a summary of our work in the next section and concludes with future direction in Section 4.2.

4.1 Summary

We identified the three main challenges in realizing self-organizing scatternets.

1. Topology formation and healing.
   In broadcast based wireless LANs such as 802.11b, the physical distance between nodes determine the network topology. Bluetooth, however, requires an explicit topology formation process because nearby devices need to discover each other and establish point-to-point links before they can communicate. In addition, algorithms are required to heal network partitions as a result of arbitrary node arrivals and departures.

2. Link scheduling.
   When multiple piconets form a connected scatternet, certain nodes must participate in more than one piconet and relay packets between piconets. Since relay nodes must communicate in multiple links on a time division basis, a link scheduling mechanism is necessary for successful packet transfers between neighboring nodes.

3. Packet routing.
   A routing mechanism is essential to forward packets over a multi-hop scatternet. Small Bluetooth packet size and low memory and energy requirements make this problem potentially challenging.

We developed an online scatternet formation algorithm, called TSF (Tree Scatternet Formation), and a dynamic link scheduling algorithm, called TSS, (Tree Scatternet Scheduling) to create connected scatternets and enable efficient communication respectively. Although we do not propose a new routing algorithm, the scatternets produced by TSF simplify routing by guaranteeing absence of loops and a unique path between any node pair. Both TSF and TSS are distributed, incremental and adapt to arbitrary node arrivals and departures without causing long disruptions in network connectivity. This novel feature, absent in earlier works, makes our solutions appealing to real world environments such as football matches or shopping malls.

TSF connects nodes in a tree structure while deciding dynamically and in a distributed fashion which node acts as master and as slave, thus avoiding centralized decision making. Unlike earlier work, our design does not restrict the number of nodes in the scatternet. TSF also allow an arriving node to begin communication with neighboring nodes in the component as soon as it is connected while requiring it to spend a small amount of time discovering neighboring nodes to improve the connectedness of the scatternet.

TSS exploits the tree structure of resulting scatternets constructed by TSF to efficiently coordinate communication tasks on various links. Unlike previous work which lacks coordination at any level, TSS coordinates one-hop neighbors effectively to increase the overall performance of the scatternet. In addition, TSS is robust and responsive to network conditions, adapting the inter-piconet communication schedule based on varying traffic loads.

We also developed a Bluetooth simulator in *ns* that includes most aspects of the Bluetooth protocol stack. We implemented both TSF and TSS in our simulator and ran numerous simulations to evaluate their performance. This demonstrated that both schemes can be implemented within the existing Bluetooth specification (Version 1.1). Our simulation results 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. The simulation results also show that TSS achieves high throughput and low packet latency for various traffic loads.

The performance of our algorithms stems from several engineering decisions. TSF exploits the asymmetric nature of two Bluetooth primitives to discover neighbors: Inquiry and Inquiry Scan. Nodes conducting Inquiry transmit and listen for a long period of time (in seconds) making Inquiry an expensive operation. On the other hand, nodes conducting Inquiry Scan only listen for a short window (in milliseconds). TSF requires each node (other than a dynamically designated coordinator) in the component tree to conduct Inquiry Scan periodically while forcing free nodes to divide their time equally between Inquiry and Inquiry Scan operations. Although nodes running TSF on average spend just 4% of their time discovering neighbors, the delay required for a node to connect to another node is a mere $1s$, not too far from the ideal expected time of $0.34s$ if two nodes were to spend on average 25% of their time discovering each other.

TSS uses hierarchy to resolve scheduling conflicts. Only a master node (parent) can suggest future communication periods to a slave node (child), which chooses a suitable period from the suggested ones. Since tree scatternets are loop-free, nodes settle on an efficient communication schedule dynamically in a top-down fashion.

In dynamic *ad hoc* environments, it is important both to improve the connectedness of the scatternet and to enable communication efficiently within the connected components at the same time. Thus, the scatternet formation scheme and the link scheduling scheme must inter-operate well. In our integrated approach, the tree topology serves as the centerpiece as it simplifies link scheduling and packet routing. Our link scheduling algorithm TSS not only schedules communication tasks efficiently by exploiting the tree structure but also ensures that scatternet formation tasks are carried out in a timely fashion. To our knowledge, we are the first to present an integrated and complete solution to realize self-organizing scatternets for dynamic environments where nodes arrive and depart at any time.

4.2 Future Work

Challenges remain before Bluetooth scatternets can be usefully deployed.

First, although TSF guarantees to produce a connected scatternet when nodes are within radio proximity, TSF 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 fails to create a single connected scatternet when either coordinators or roots cannot hear each other. We offer the following suggestions to extend TSF so that it will work well for networks with diameter larger than one. Tree nodes can enter the Inquiry Scan mode periodically to listen for packets transmitted by coordinators in addition to the ones transmitted by free nodes. As soon as a tree node establishes a connection with a coordinator, both nodes will break the link and inform corresponding roots to merge component trees. To guarantee that the resulting topology is loop-free, a tree node must not connect to the coordinator from its own tree. Since there are 64 reserved Inquiry Access Codes, only two of which are in use, we can assign random Inquiry Access Code for each tree. Whenever a new root or coordinator is elected, the root node needs to broadcast the Inquiry Access Code and the root identifier to all other nodes in the tree. Thus, tree nodes can detect the cycle during link creation and break it immediately.

Second, TSF should be extended to scenarios where an infrastructure exists. In places like airports, Bluetooth base stations, that are connected to each other and to the Internet via a wired network, can be installed throughout the airport to provide connectivity to travelers. Each base station can be configured as the root node to which newly arrived nodes attach. Free nodes running TSF should be configured to only attach to the existing scatternet tree and not to each other. Doing so will eliminate the healing protocol since scatternets can only exist with base stations configured as roots. In addition, TSF can dictate the shape of the tree accurately by controlling which nodes conduct discovery operations when. We believe that TSF can be tailored in simple ways to work well with infrastructure based Bluetooth systems.

Lastly, an efficient routing algorithm should be developed specifically for tree scatternets. 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. The packet routing protocol works by simply having each node look at the destination identifier and forward it along one of its links. The main challenge in realizing such protocol is in dealing with mobility since the identifiers for nodes will change as they leave. The address resolution scheme must update its table entries appropriately as nodes come and go.

4.3 Conclusion

This thesis has addressed challenges in internetworking personal area networks using Bluetooth. The designers of Bluetooth have chosen a centralized TDD scheme for simplicity and a Frequency Hopping technique to provide robustness against interference. The combination of these two schemes makes internetworking Bluetooth piconets challenging. In this thesis, we presented efficient solutions for forming scatternets and scheduling communication links. Both problems are not limited to the Bluetooth technology. As battery-powered devices are becoming abundant, the fundamental goal of the algorithms enabling communication between them becomes clear: power efficiency. One way to achieve that goal is to form an efficient topology for communication where communication paths are short and routing is simple. Another way is to develop an effective schedule for communication so that nodes are only active to exchange useful data and sleep, otherwise. Both TSF and TSS are designed to achieve these goals while dealing with the complications introduced by Bluetooth.

It remains to be seen whether Bluetooth can deliver us the promised land of self-organizing scatternets. A lot of it depend on the availability of "killer" applications and cheaper products. Nevertheless, Bluetooth has taken a huge step in convincing many of us that "network is the computer." All Bluetooth layers can be implemented on a single chip. For most everyday devices, only a small application will be required to provide the necessary functionality and it can potentially use the Bluetooth chip for computation in addition to communication. If the next phase of technological advancements is solely to enhance our ways of lives by reducing complexity, the future belongs to those intelligent devices equipped with technologies to self-organize, self-heal and use power efficiently.

Bibliography

[1] Bluetooth Special Interest Group Web Site. `http://www.bluetooth.com/`.

[2] Alok Aggarwal, Manika Kapoor, Lakshmi Ramachandran, and Abhinanda Sarkar. Clustering Algorithms for Wireless Ad Hoc Networks. In *4th International Workship on Discrete Algorithms and Methods for Mobile Computing and Communications*, Boston, MA, August 2000.

[3] D. Avis. A survey of heuristics for the weighted matching problem. *Networks*, 13:475–493, 1983.

[4] Petra Berenbrink, Marco Riedel, and Christian Scheideler. Simple competitive request scheduling strategies. In *11th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 33–42, 1999.

[5] 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.

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

[7] Sensor Networks in Industrial Automation. `http://www.xbow.com/crossnet/Applications/applications_frame.htm/`.

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

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

[10] 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.

[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*, pages 85–96, Rome, Italy, July 2001.

[12] A. Demers, S. Keshav, and S. Shenker. Analysis and Simulation of a Fair Queueing Algorithm. *Internetworking: Research And Experience*, 1:3–26, April 1990.

[13] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next Century Challenges: Scalable Coordination in Sensor Networks. In *Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom '99)*, pages 263–270, August 1999.

[14] H. N. Gabow and R. E. Tarjan. Almost optimal speed-ups of algorithms for matching and related problems. In *20th Symposium on Theory of Computing (STOC)*, pages 514–527, 1988.

[15] H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for network problems. *Journal of Computing*, 18:1013–1036, 1989.

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

[17] 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)*, January 2000.

[18] Niklas Johansson, Fredrik Alriksson, and Ulf Jonsson. JUMP Mode- A Dynamic Window-based Scheduling Framework for Bluetooth Scatternets. In *ACM Symposium on Mobile Ad Hoc Networking and Computing*, Long Beach, CA, October 2001.

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

[20] J. Kahn, R. Katz, and K. Pister. Mobile Networking for Smart Dust. In *Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom '99)*, August 1999.

[21] B. Kalyanasundaram and K. Pruhs. Online weighted matching. *Journal of Algorithms*, 14:478–488, 1993.

[22] Ching Law, Amar K. Mehta, and Kai-Yeung Siu. Performance of a New Bluetooth Scatternet Formation Protocol. In *ACM Symposium on Mobile Ad Hoc Networking and Computing*, Long Beach, CA, October 2001.

[23] C. Lin and M. Gerla. Adaptive Clustering for Mobile Wireless Networks. *IEEE Journal on Selected Areas in Communications*, 15(7):1265–1275, September 1997.

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

[25] R. Motwani. Average-case analysis of algorithm for matching and related problems. *JACM*, 41:1329–1356, 1994.

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

[27] 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*, pages 90–100, New Orleans, LA, February 1999.

[28] Andras Racz, Gyorgy Milklos, Ferenc Kubinszky, and Andras 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.

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

[30] 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.

[31] M. Shreedhar and George Varghese. Efficient Fair Queuing using Deficit Round Robin. In *Proc. of ACM SIGCOMM*, August 1995.