行政院國家科學委員會專題研究計畫　期中進度報告

藍芽個人無線區域網路上通訊議題之設計實作與分析(2/3)

計畫類別： 個別型計畫
計畫編號： NSC93-2213-E-009-030-
執行期間： 93 年 08 月 01 日至 94 年 07 月 31 日
執行單位： 國立交通大學資訊工程學系(所)

計畫主持人： 曾煜棋

報告類型： 精簡報告

處理方式： 本計畫可公開查詢

中　華　民　國 94 年 5 月 19 日

摘要

藍芽（Bluetooth）技術係一新興崛起之個人區域網路標準，並預期在未來無線通訊領域發展中將扮演著重要的角色；藍芽當初原本是設計用來取代纜線設備（例如：滑鼠，鍵盤與個人電腦之連接線），但在後來的演進中發現，藍芽的應用面並不僅止於此；隨著市面上低價藍芽模組的陸續出現，大規模的藍芽設備佈署將很有可能在近期內實現。針對上述趨勢，我們將分別就藍芽相關通訊議題，提出一系列的解決方案或分析結果，並實作出真正具有通訊功能的藍芽散網(scatternet)與微網(piconet)，評估不同網路結構對整體通訊效能的實際影響，以達到驗證理論分析之目的。

本計畫的主要目標，是建立有效率的藍芽通訊網路並以理論加以評估。在第一年當中，針對藍芽的主從式網路架構做改善。在藍芽微網的環境下，探討主裝置詢問（poll）從屬裝置之機制策略，儘量填滿所有的負載欄位（payload field），以及空的數據封包，期望更有效率地使用每一個藍芽時槽（time slot），藉此提昇藍芽微網最大可容納之生產量（throughput），使得更多的通訊活動能夠同時被支援。

本報告將提出第二年的研究成果。第二年的工作重點在於將數個微網連結成一個更大的網路結構，稱為散網。我們提出一個藍芽散網的環形結構，稱為藍環(BlueRing)。進而探討並實作藍環之形成(formation)，繞徑(routing)、維持管理(maintenance)以及提供一種回復機制（recovery mechanism），使藍環網路拓樸結構具有容錯能力，在出現錯誤時能迅速回復網路結構。此外，我們針對藍芽標準 V1.1 版及 V1.2 版，分析其頻率匹配(frequency-matching)的延遲時間，並針對分析的結果，提出了三種可以提升藍芽裝置搜尋時間的方法。

關鍵詞： 藍芽，散網，微網

目錄

報告內容 …… 1

I. 前言 …… 1

II.研究目的……2

III.研究方法…… 2

IV.研究成果……7

附錄…… 8

I. 發表論文全文 …… 8

報告內容

I. 前言

藍芽係採主從裝置之網路結構（master-slave configuration），其基本的網路單元稱為微網（piconet），將數個微網連結起來可以形成一個更大的網路結構，稱為散網（scatternet），然而，在藍芽現有的標準規格裡，散網的結構與運作方式並未被清楚定義，端視各家廠商如何設計; 因此，一個簡單且具通訊效率的藍芽散網拓樸結構能否被提出，便成為藍芽是否能成為個人無線區域網路（Wireless Personal-Area Network，WPAN）上主要標準技術元件的關鍵所在;在本計畫執行的第二年中，我們提出一個藍芽散網之環形結構，稱為藍環（BlueRing），係將數個微網以環狀的方式串接起來，連接微網與微網之藍芽主機稱為橋接器（bridge），橋接器負責跨微網（inter-piconet）之封包轉送；除此以外，在這項研究裡，我們設計藍環（BlueRing）形成（formation）、繞徑（routing）、及維持管理（maintenance）機制，並評估其在通訊效能（communication efficiency）及容錯能力（fault tolerance）上，和其它拓樸形狀表現的差異。此外，搜尋藍芽裝置的時間過長，將會嚴重的影響到許多行動應用，因此，我們針對其頻率匹配(frequency-matching)的延遲時間進行分析，其結果適用於藍芽標準 V1.1 版及 V1.2 版，並根據分析的結果，提出了三種可以提升藍芽裝置搜尋時間的方法。

II. 研究目的

本年度的工作重點在於將數個微網連結成一個更大的網路結構，稱為散網。這個部分目前藍芽現有的標準規格裡並沒有清楚的定義，端視各家廠商如何設計。而在藍芽設備日益增加的情況下，欲延伸藍芽的使用，進而形成成個人無限區域網路（Wireless Personal-Area Network，WPAN），藍芽散網建立與維持將是關鍵技術。而我們將提出一個藍芽散網的環形結構，稱為藍環(BlueRing)。進而探討並實作藍環之形成(formation)，繞徑(routing)、維持管理(maintenance)以及提供一種回復機制（recovery mechanism），使藍環網路拓樸結構具有容錯能力，在出現錯誤時能迅速回復網路結構。此外，我們針對藍芽標準 V1.1 版及 V1.2 版，分析其頻率匹配(frequency-matching)的延遲時間，針對分析的結果，提出了三種可以提升藍芽裝置搜尋時間的方法。

III. 研究方法

以下我們將對藍芽散網的環形結構藍環(BlueRing) ，分別詳述其形成(formation)，繞徑(routing)、維持管理(maintenance)等機制，並分析藍芽裝置搜尋時間且提出如何改善的方法。

1. 藍環(BlueRing)的形成協定(Formation Protocol)

為了建立藍環，採用集中形成機制（Centralized formation mechanism）。假設所有藍環裝置皆在無線電波涵蓋的範圍內，為每個藍芽裝置定義一個藍環成員（RING_MEM）參數，以判別是否為藍環之中的裝置（若值為 1 則是，若為 0 則否），剛開始 RING_MEM 設為 0。建立過程分為兩個階段：

在第一階段中，每個藍芽裝置以機率 p 查詢（I）及以機率 1-p 查詢掃描（IS）。當某個 I 與某個 IS 相符合，則兩個藍芽裝置建立臨時微網，於其中有三個參數被交換：藍環成

員、搜尋到的藍芽裝置數目以及藍芽裝置位址。首先，比較藍環成員參數。假使某個藍芽裝置的藍環成員參數為1，而其它的為0，則它贏了。

如果發生平手的狀況，則看哪一個藍芽裝置收集較多的其他藍芽裝置資訊就贏了。假使無法決定勝利者，則由唯一的藍芽裝置位址參數決定，輸家要提供贏家所有到收集的藍芽裝置資訊，在交換所有資訊後，臨時微網就被拆掉。假如在查詢時間終止（inquiry timeout)（IT）內沒有接收到其它查詢／查詢掃描訊息，贏家可宣稱自己為領導者。接著領導者進入呼叫狀態，試著去收集其它非領導者，其它非領導者必須進入呼叫掃描狀態，等待被呼叫。

在第二階段中，根據所希望建立的環狀拓樸，領導者藉由設立臨時微網，指定數個藍芽裝置為主裝置。對於每個指定的主裝置，領導者並提供該主裝置旗下從屬裝置的資訊，包括指定下游和上游橋接器。擁有這樣的資訊後，每一個主裝置就可以各自去呼叫旗下的從屬裝置，且建立自己的微網。提供橋接器服務的裝置應確定裝置本身的下游和上游主裝置已成功完成連結。當變成環的一部份，藍芽裝置將藍環成員參數設定為1。

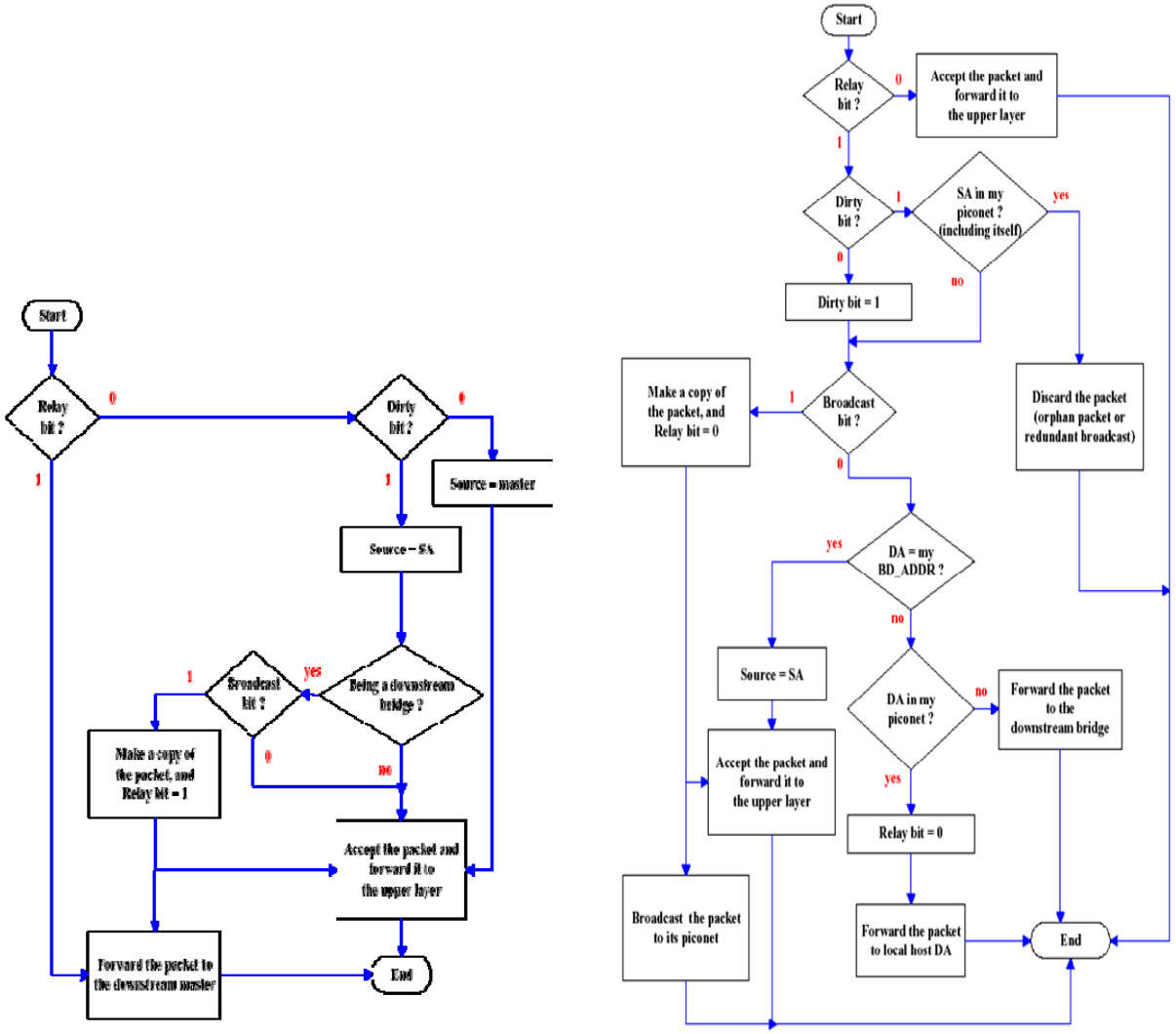
2. 藍環(BlueRing)的繞徑協定(Routing Protocol)

在這個項目中，主要是在藍環中提供可支援點對點傳送（unicasting）及廣播（broadcasting）之繞徑協定。項目一中提到，資料封包會沿著藍環的方向進行繞徑，由於封包沿著藍環流動最後到達目的微網，不需要繞徑搜尋處理（route discovery process）程序。因此藍環中的繞徑沒有紀錄之前狀態，所以沒有繞徑表需要進行維護；相反的，大多相對映兩點直接資料傳輸網路（Ad hoc network）之繞徑協定需要維持繞徑表。

藍環之封包負載格式依據封包繞徑方式分為三種：封包負載格式a、封包負載格式b以及封包負載格式c。封包負載格式a係表示在單點跳躍單點傳送傳遞，封包負載格式b係表示多點跳躍單點傳送傳遞，而封包負載格式c係表示廣播傳遞。為了在藍環上繞徑，必須增加三個控制位元於封包負載標頭之後，根據不同傳輸方式而有不同表現方式。三個控制位元分別是：廣播位元（Broadcast bit)、中繼位元（Relay bit)、髒位元（Dirty bit)。

在封包負載格式 52 中另外包括 48 位元之來源藍芽裝置位址（SA，Source BD_ADDR）欄位，及 48 位元之目的藍芽裝置位址（DA，Destination BD_ADDR），紀錄來源藍芽裝置之位址及目的藍芽裝置之位址；而封包負載格式 53 包括 48 位元之來源藍芽裝置位址，紀錄來源藍芽裝置之位址。

下面是藍環從屬裝置以及主裝置繞徑之示意圖，說明當從屬裝置以及主裝置接收到封包該如何運作。



接下來是藍環的橋接原則。採用以臨界為基礎（threshold-based）之策略以實施停放／非停放（park/unpark）要求，使用下述三個參數：(1)T_b：計算橋接器中等候封包數之臨界值；(2)T_m：計算主裝置中等候封包數之臨界值；以及(3)T_{out}：計算橋接器切換微網之時間終止值。

在一般情況下，由於大多數時間，橋接器直覺地將連結至其上游微網，在某些臨界條件為真，將切換至下游微網。當連結至下游微網，橋接器將被下游主裝置視為高優先權以便儘快排除緩衝區的封包。

3. 藍環(BlueRing)的維持管理(Maintenance Protocol)

在封包繞徑時，容錯是基本的問題，特別是在行動環境下。在藍環中，當任何主裝置或橋接器離開網路，環會中斷且變成線性路徑，新的藍芽裝置可能會加入。以下敘述如何解決在單點錯誤和多點錯誤之情況。

單點錯誤（Single-Point Failure）：假設某一為主裝置之主機或橋接器失去作用，因為在藍環中有初始繞徑方向，假使封包總是以正向傳送，則主機無法到達另一在反向的主機。在封包負載標頭後增加一個新的稱為“方向（Direction）”之控制位元，這個位元協助主機決定繞徑遵循方向（正向／反向）。

多點錯誤（Multi-Point Failure）：上述之容錯繞徑協定確定繞徑不受影響，但留下損壞點沒有修復，在此我們提供了一個回復機制可以重新連結藍環網路。新的主機也可加入現行的藍環，只需區域重新連結。只要沒有兩個關鍵點同時失去作用，協定可正確運作。

在藍芽規格裡保留 63 個專屬查詢存取碼（DIACs，Dedicated Inquiry Access Code）以發現範圍內某些特定的裝置，在此提出使用兩個保留的專屬查詢存取碼，分別為第一專屬查詢存取碼和第二專屬查詢存取碼，以幫助藍環恢復。同樣地，將使用一般查詢存取碼（GIAC，General Inquiry Access Code）以吸引主機加入現行藍環。以下範例說明如何處理橋接器離開（bridge leaving），主裝置離開（master leaving），以及如何藉由產生更多微網以擴展藍環規模。

另外，為了使藍環網路更具擴充性，希望可以納入更多新加入之藍芽裝置數目，我們可以適當增加環內微網的數目，由於藍芽規格本身的限制，使得單一微網所能容納的從屬裝置數目有限（≦7），因此，增加微網數目意味著增加所能管理的藍芽裝置數目。當某一主裝置旗下的從屬裝置數目超過預設之臨界值，會將原本屬於同一個微網之主裝置及從屬

裝置拆成兩個微網，以容納更多欲加入藍環網路之藍芽裝置。

4. 藍芽裝置搜尋(Bluetooth Device Discovery)分析及其加速機制

搜尋藍芽裝置的時間過長，將會嚴重的影響到許多行動應用，因此，我們針對藍芽標準 V1.1 版及 V1.2 版，分析其頻率匹配(frequency-matching)的延遲時間，根據我們分析的結果，其平均的裝置搜尋時間為 23.55 秒；除此之外，更針對分析的結果，發現主要造成過長的裝置搜尋時間原因，在於必須長時間的等待從主置發出來的詢問時窗(inquiry window)，對此，我們提出了三種可以改善藍芽裝置搜尋時間的方法，分別為：

- Half Inquiry Interval (HII)：將詢問的週期縮減為一半，其效果可以減少前述的等待時間。
- Dual Inquiry Scan (DIS)：從屬裝置將其做詢問頻率增為兩倍，可以增加每次詢問時窗會被詢問到的機會，因而大量地減少頻率匹配的延遲。
- Combination of HII and DIS：結合上述兩種方法，其結果將可以大量的使搜尋時間從 23.55 秒縮短到 11.38 秒。

IV. 研究成果

本計畫依時程順利進行，並獲致豐碩成果，相關論文發表如下，論文全文詳如附錄：

- T.-Y. Lin, Y.-C. Tseng, and K.-M. Chang, “A New BlueRing Scatternet Topology for Bluetooth with Its Formation, Routing, and Maintenance Protocols”, ***Wireless Communications and Mobile Computing***, Vol. 3, No. 4, June 2003, pp. 517-537. (SCIE)
- J.-R. Jiang, B.-R. Lin, and Y.-C. Tseng, "Analysis of Bluetooth Device Discovery and Some Speedup Mechanisms", ***Int'l J. of Electrical Engineering***, Vol. 11, No. 4, Nov. 2004, pp. 301-310. (EI)

A New BlueRing Scatternet Topology for Bluetooth with Its Formation, Routing, and Maintenance Protocols

Ting-Yu Lin[1], Yu-Chee Tseng[1], and Keng-Ming Chang[2]

[1]Department of Computer Science and Information Engineering
National Chiao-Tung University, Hsin-Chu, 300, Taiwan
E-mail:{tylin, yctseng}@csie.nctu.edu.tw

[2]Department of Computer Science and Information Engineering
National Central University, Chung-Li, 320, Taiwan

Corresponding author: Professor Yu-Chee Tseng
E-mail: yctseng@csie.nctu.edu.tw

Abstract

The basic networking unit in Bluetooth is *piconet*, and a larger-area Bluetooth network can be formed by multiple piconets, called *scatternet*. However, the structure of scatternets is not defined in the Bluetooth specification and remains as an open issue at the designers' choice. It is desirable to have simple yet efficient scatternet topologies with well supports of routing protocols, considering that Bluetooths are to be used for *personal-area networks* with design goals of simplicity and compactness. In the literature, although many routing protocols have been proposed for *mobile ad hoc networks*, directly applying them poses a problem due to Bluetooth's special baseband and MAC-layer features. In this work, we propose an attractive scatternet topology called *BlueRing* which connects piconets as a ring interleaved by bridges between piconets, and address its formation, routing, and topology maintenance protocols. The BlueRing architecture enjoys the following nice features. First, routing on BlueRing is stateless in the sense that no routing information needs to be kept by any host once the ring is formed. This would be favorable for environments such as Smart Homes where computing capability is limited. Second, the architecture is scalable to median-size scatternets easily (e.g., around 50∼70 Bluetooth units). In comparison, most star- or tree-like scatternet topologies can easily form a communication bottleneck at the root of the tree as the network enlarges. Third, maintaining a BlueRing is an easy job even as some Bluetooth units join or leave the network. To tolerate single-point failure, we propose a protocol-level remedy mechanism. To tolerate multi-point failure, we propose a recovery mechanism to reconnect the BlueRing. Graceful failure is tolerable as long as no two or more critical points fail at the same time. As far as we know, the fault-tolerant issue has not been properly addressed by existing scatternet protocols yet. In addition, we also evaluate the ideal network throughput at different BlueRing sizes and configurations by mathematical analysis. Simulations results are presented, which demonstrate that BlueRing outperforms other scatternet structures with higher network throughput and moderate packet delay.

Keywords: ad hoc network, Bluetooth, mobile computing, personal-area network (PAN), piconet, routing, scatternet, wireless communication.

1 Introduction

Wireless communication is perhaps the fastest growing industry in the coming decade. It is an enabling technology to make computing and communication anytime, anywhere possible. Depending on whether base stations are established or not, a wireless network could be classified as *infrastructure* or *ad hoc*. According to the radio coverage and communication distance, it can be classified as *wide-area*, *local-area*, *personal-area*, or even *body-area*.

This paper focuses on Bluetooth [1], which is an emerging PAN (Personal Area Network) technology, and is characterized by indoor, low-power, low-complexity, short-range radio wireless communications with a frequency-hopping, time-division-duplex channel model. Main applications of Bluetooths are targeted at wireless audio link, cable replacement, and ad hoc networking. The basic networking unit in Bluetooth is called *piconet*, which consists of one master and up to seven active slaves. For a larger wide-spread deployments, multiple piconets can be used to form a *scatternet*. A host may participate in two piconets to relay data, to which we refer as a *bridge* in this paper.

In the Bluetooth specification, the structure of scatternets is not defined, and it remains as an open issue at the designers' choice. In the literature, although many routing protocols have been proposed for *mobile ad hoc networks* based on wireless LAN cards [10], directly applying them poses a problem due to Bluetooth's special baseband and MAC-layer features [3]. It is desirable to have simple yet efficient scatternet topologies with well supports of routing protocols, considering that Bluetooths are to be used for PAN with design goals of simplicity and compactness. According to [4, 5], Bluetooth-based mobile ad hoc networks need routing protocols closely integrated with underlying scatternet topologies. The reason stems from the physical and link-level constraints of Bluetooth technology, making legacy routing protocols for mobile ad hoc networks (e.g., [10]) unsuitable for scatternets.

Several previous papers [6, 9, 12] have addressed the performance issues, which motivate studies of the scatternet formation problem. References [7, 11, 13, 14] propose various scatternet formation mechanisms (refer to the review in Section 2.4). However, all these works fail to provide clear and efficient routing protocols to run over the proposed scatternet topologies. Until recently, reference [8] proposes a routing protocol based on a 2-level hierarchical scatternet. Two types of local networks are defined: *PAN (Personal Area Network)* and *RAN (Routing Area Network)*. RAN is responsible of interconnecting PANs. All traffic from a PAN needs to go through the RAN to reach another PAN. However, this approach suffers from two drawbacks. First, the number of

participating Bluetooth units is limited. Second, the RAN may become the bottleneck of the whole network, in terms of both communication delays and fault tolerance capability.

In this work, we propose an attractive topology called *BlueRing* for scatternet structure, and address its formation, routing, and maintenance protocols. While similar to the IEEE 802.5 token-ring in topology, our BlueRing differs from token ring in several aspects due to Bluetooth's special baseband features. First, the ring consists of multiple piconets with alternating masters and slaves and thus can de facto be regarded as a *ring of trees* since each master can connect to multiple active slaves. Second, no token is actually running on the ring. Third, since each piconet has its unique frequency hopping sequence, multiple packets may be relayed on the ring simultaneously. Routing protocols to support unicast and broadcast on BlueRings are proposed. For bridges (slaves connecting two piconets), a bridging policy is clearly defined so as to relay packets efficiently.

The BlueRing architecture enjoys the following nice features. First, routing on BlueRing is stateless in the sense that no routing information needs to be kept or constructed by any host once the ring is formed. This would be favorable for environments such as Smart Homes where computing capability is limited. Second, the architecture is scalable to median-size scatternets (e.g., around 50$\sim$70 Bluetooth units). In comparison, most star- or tree-like scatternet topologies can easily form a communication bottleneck at the root of the tree as the network enlarges. Third, maintaining a BlueRing is an easy job even if some bluetooth units join or leave the network. To tolerate single-point failure, we propose a protocol-level remedy mechanism. To tolerate multi-point failure, we propose a recovery mechanism to reconnect the BlueRing. Graceful failure is tolerable as long as no two or more critical points fail at the same time. As far as we know, the fault-tolerant issue has not been properly addressed by existing scatternet protocols yet.

The rest of this paper is organized as follows. Preliminaries are in Section 2. The formation, routing, and maintenance protocols for BlueRing are proposed in Section 3, Section 4, and Section 5, respectively. In Section 6, we present some analysis and simulation results. Finally, Section 7 summarizes the paper and points out our future work.

2 Preliminaries

2.1 Bluetooth Protocol Stack

Bluetooth is a master-driven, short-range radio wireless system. The smallest network unit is a *piconet*, which consists of one master and up to 7 active slaves. Each piconet owns one frequency-

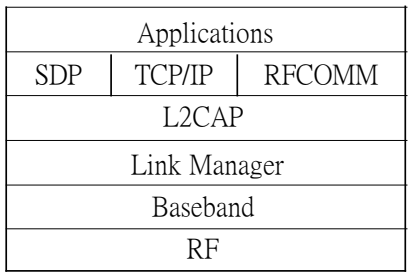


Figure 1: Bluetooth protocol stack.

hopping channel, which is controlled by its master in a time-division-duplex manner. A time slot in Bluetooth is 625μs. The master always starts its transmission in an even-numbered slot, while a slave, on being polled, must reply in an odd-numbered slot.

Fig. 1 shows the Bluetooth protocol stack. On top of RF is the Bluetooth Baseband, which controls the use of the radio. Four important operational modes are supported by the baseband: *active*, *sniff*, *hold*, and *park*. The active mode is most energy-consuming, where a bluetooth unit is turned on for most of the time. The sniff mode allows a slave to go to sleep and only wake up periodically to check possible traffic. In the hold mode, a slave can temporarily suspend supporting data packets on the current channel; the capacity can be made free for other things, such as scanning, paging, inquiring, and even attending other piconets. Prior to entering the hold mode, an agreement should be reached between the master and slave on the hold duration. When a slave does not want to actively participate in the piconet, but still wants to remain synchronized, it can enter the park mode. The parked slave has to wake up regularly to listen to the beacon channel, for staying synchronized or checking broadcast packets.

On top of Baseband is the Link Manager (LM), which is responsible for link configuration and control, security functions, and power management. The corresponding protocol is called Link Manager Protocol (LMP). The Logical Link Control and Adaptation Protocol (L2CAP) provides connection-oriented and connectionless datagram services to upper-layer protocols. Two major functionalities of L2CAP are protocol multiplexing and segmentation and reassembly (SAR).

The Service Discovery Protocol (SDP) defines the means for users to discover which services are available in their neighborhood and the characteristics of these services. The RFCOMM protocol provides emulation of serial ports over L2CAP so as to support many legacy applications based on serial ports over Bluetooth without any modifications. Up to 60 serial ports can be emulated.

2.2 Operations of the Park Mode

This paper proposes a new topology called BlueRing for scatternet structure. Since a scatternet must involve multiple piconets, some devices must participate in more than one piconet. Such devices are called *bridges* in this paper, and a bridging policy is needed for them to efficiently relay packets from piconets to piconets. A bridge host has to frequently pause activities in one piconet and switch to another piconet. In this paper, we propose to adopt the park mode for this purpose.

Below we give the reason why we choose park mode. The Bluetooth specification provides three options for a device to temporarily pause its current activity: sniff, hold, and park modes. The sniff mode has a periodical, prearranged wakeup pattern, and thus is more suitable for a device to switch from piconets to piconets with a regular pattern. It is not selected here because with a regular pattern time slots may easily get wasted. Moreover, with our BlueRing, which chains a sequence of piconets, determining a good sniffing pattern is very difficult. The hold mode would be favorable if the amount of time that a bridge should stay in each piconet can be predetermined. Unfortunately, this assumption is unrealistic, especially in a dynamic environment. The park mode is more favorable in our case since it allows a device to temporarily give up its current activity in one piconet for an arbitrary period of time until an unpark request is issued. The unpark request can be master-activated or slave-activated, but should be approved by the master. Hence, we adopt the park mode in our bridging policy, considering its simplicity and flexibility.

In the following, we review the park mode operations in more details. On entering the park mode, a slave gives up its active member address AM_ADDR, but receives two new addresses:

- PM_ADDR: 8-bit Parked Member Address
- AR_ADDR: 8-bit Access Request Address

The PM_ADDR distinguishes a parked slave from the other parked slaves. This address is used in the master-initiated unpark procedure. In addition to the PM_ADDR, a parked slave can also be unparked by its 48-bit BD_ADDR (Bluetooth Device ADDRess). The all-zero PM_ADDR is a reserved address. If a parked unit has the all-zero PM_ADDR, it can only be unparked by the BD_ADDR. In that case, the PM_ADDR has no meaning.

The AR_ADDR is used by the slave in the slave-initiated unpark procedure. All messages sent to the parked slaves have to be carried by broadcast packets (the all-zero AM_ADDR) because they have no AM_ADDR. To support parked slaves, the master establishes a beacon channel when one

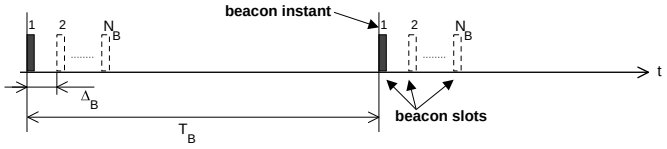


Figure 2: General beacon channel format.

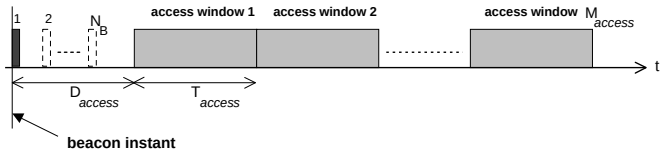


Figure 3: Definition of access window.

or more slaves are parked. The beacon channel consists of one beacon slot or a train of equidistant beacon slots which is transmitted periodically with a constant time interval. The beacon channel is illustrated in Fig. 2. In each period of T_B slots, a train of N_B ($N_B \geq 1$) beacon slots is transmitted. The start of the first beacon slot is referred to as the *beacon instant*, which serves as the timing reference. Parameters N_B and T_B are chosen such that there are sufficient beacon slots for a parked slave to synchronize during a certain time window in an error-prone environment.

In addition to the beacon slots, an access window is defined where parked slaves can send unpark requests. To increase reliability, the access window can be repeated M_{access} times ($M_{access} \geq 1$), as shown in Fig. 3. The first access window starts a fixed delay D_{access} after the beacon instant. The width of each access window is T_{access}. The same TDD structure is adopted by alternative transmission between the master and slaves, as shown in Fig. 4. However, the slave-to-master slot is divided into two half slots of 312.5μs each. A parked slave should count the half slots and only in the proper half slot corresponding to its AR_ADDR may it respond. The parked slave is only permitted to send an unpark request if in the preceding master-to-slave slot a broadcast packet has been received. If the unpark request is approved, the master polls the parked slave.

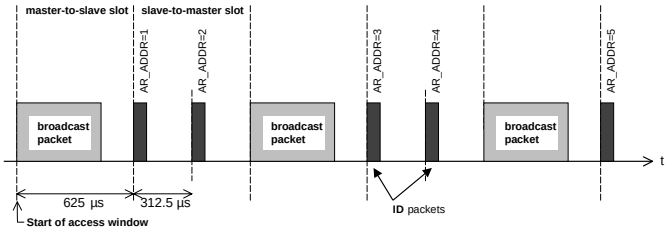


Figure 4: Unpark request procedure in access window.

2.3 Bluetooth Device Address and Access Codes

Following the IEEE 802 standard, each Bluetooth unit is assigned a unique 48-bit Bluetooth device address (BD_ADDR), which consists of three fields as follows:

- LAP: 24-bit lower address part,
- UAP: 8-bit upper address part, and
- NAP: 16-bit non-significant address part.

The clock and BD_ADDR of a master defines the frequency hopping sequence of its piconet. In addition, the LAP of a master's BD_ADDR determines the access code to be used within its piconet. In Bluetooth, each packet is preambled by a 68/72-bit access code. The access code is used mainly for synchronization and identification. Three types of access codes are available:

- Channel Access Code (CAC)
- Device Access Code (DAC)
- Inquiry Access Code (IAC)

The CAC is determined by the LAP of master's BD_ADDR and is used in connected mode, when piconets are already established. CAC, which precedes every packet, identifies all packets exchanged on the same piconet channel. Hence a Bluetooth unit can easily tell packets of its own piconet from others. The DAC is used for paging and responding to paging. DAC is derived from the LAP of the paged device's BD_ADDR. The IAC is used in inquiry process. Two variants of IAC are supported:

- General Inquiry Access Code (GIAC) and
- Dedicated Inquiry Access Code (DIAC).

There is only one GIAC, which is for discovering all Bluetooth devices in range. However, there are 63 DIACs, each for discovering a specific class of Bluetooth units. Since IACs are derived from LAPs, a block of 64 contiguous LAPs is reserved for this purpose. One LAP is used for general inquiry, and the remaining 63 LAPs are reserved for dedicated inquiries. None of these LAPs can be part of a Bluetooth unit's BD_ADDR.

We will take advantage of the reserved DIACs in our BlueRing recovery procedures for fault tolerance.

2.4 Review of Scatternet Formation Algorithms

Below, we review some existing scatternet formation schemes. The work in [6] has studied one piconet with both active and park slaves. Slaves are switched between active and park modes based on the timestamps when they entered a state. A parked slave with the oldest timestamp is periodically unparked by parking the active slave with the oldest timestamp. This model suffers from low system throughput and long packet delays, and incurs many mode-switching overheads when the number of slaves is large. Also, with one piconet, the multi-channel benefit of using scatternet is not exploited. According to [9], by grouping nodes into different piconets, significant performance improvement may be obtained. The reason is that simultaneous communications can occur among different piconets. However, since different frequency-hopping channels of different piconets may still present radio interference, the Bluetooth Whitepaper [2] has suggested that up to 8 piconets may coexist in the same physical environment.

Scatternet formation is explored in [7, 11, 13, 14]. In [11], a 2-stage distributed randomized algorithm is proposed to form a network of star-shaped clusters, where each cluster is a piconet with at most 7 active slaves. The goal is to maximize the number of nodes into each piconet so that the number of clusters is minimized. However, how these piconets are interconnected is not addressed. A similar work is in [14], where a fast scatternet formation algorithm is proposed to connect piconets as a tree. How to form a tree-like scatternet with bounded time and message complexities is presented in [7]; there is no limitation on the number of participant nodes. Clearly, the center/root host in a star/tree scatternet can become a communication bottleneck of the network. Furthermore, designing fault-tolerant routing protocols on a star/tree-like network is a difficult job since any single fault will partition the network. In [13], assuming that all devices are within each other's radio coverage, a fully-connected scatternet is constructed such that connectivity exists between each pair of piconets. At most 36 Bluetooth devices can participate in the scatternet.

We note that all the above works [7, 11, 13, 14] do not clearly address the corresponding routing and bridging protocols to be run over the proposed scatternets. While there is no standard criteria for good scatternet topologies, we conclude some guidelines for scatternet construction:

- The number of piconets should be kept as small as possible, so as to reduce inter-piconet interference and communication complexity.
- A node should participate in at most two piconets, so as to reduce switching overheads.

- To reduce redundant inter-piconet links, two piconets should not be connected by more than one bridge.
- Simple and efficient routing.
- Good mobility- or fault-tolerant capability.

3 The BlueRing Formation Protocol

3.1 Network Architecture

In this subsection, we propose the BlueRing structure. A BlueRing is a scatternet consisting of a cycle of piconets which form a ring. Although physically the ring is undirected, logically we impose a direction on it (say, clockwise). So each piconet has a *downstream piconet* in the forward direction, and an *upstream piconet* in the backward direction. Packets will flow following the direction of the ring, until destinations are reached. In each piconet, two of the slaves are designated as *bridges*, one for connecting to the upstream piconet, called the *upstream bridge*, and one for connecting to the downstream piconet, called the *downstream bridge*. For instance, as shown in Fig. 5, nodes 4 and 6 are upstream and downstream bridges of master M2, respectively. Similarly, each bridge also has a upstream and a downstream masters. Therefore, each bridge host serves as an upstream bridge in one piconet and a downstream bridge in another piconet, and each piconet should have at least two slaves. Fig. 5 illustrates the BlueRing architecture and a real example.

3.2 Initial Formation

In order to construct a BlueRing, we adopt a centralized formation mechanism similar to [13]. Assume that all Bluetooth devices are within the radio coverage of each other. We define a parameter RING_MEM for each Bluetooth to indicate whether it has become a member of the BlueRing or not (1 for "yes", 0 for "not yet"). Initially, RING_MEM equals 0. The construction has two stages.

- **Stage I:** Each Bluetooth chooses to inquiry (I) with probability p and to inquiry scan (IS) with probability $1 - p$. When some I matches with some IS, the two Bluetooths establish a temporary piconet. Three parameters are exchanged between them: RING_MEM, number of acquired Bluetooths, and BD_ADDR. First, their RING_MEMs are compared. The one with RING_MEM=1 wins if the other's RING_MEM=0. In case of a tie, the one that has gathered more Bluetooths' information wins. If the above cannot determine a winner, tie is

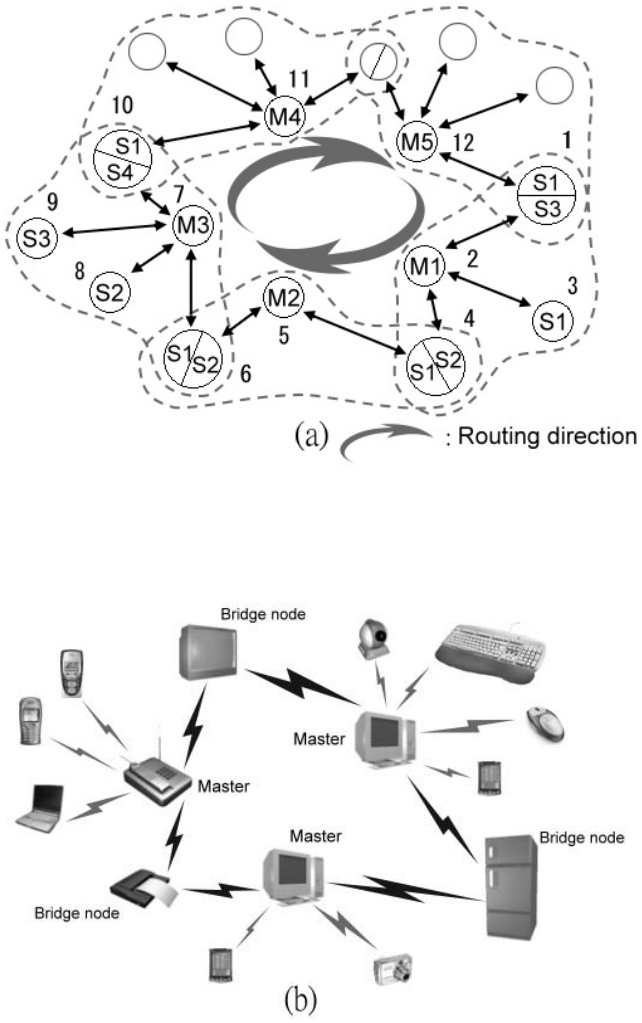


Figure 5: (a) the BlueRing architecture with nodes 2, 5, 7, 11, and 12 serving as masters, and (b) a BlueRing example.

broken by their unique BD_ADDRs. The loser should provide the winner with all Bluetooths' information it has gathered. After the information exchange, the temporary piconet is torn down. The potential winner can claim itself as a leader if no further I/IS message is received within an *inquiry timeout (IT)*. Then the (potentially only) leader enters the page state, trying to collect other non-leaders, which must enter the page scan state, waiting to be paged. The details are in Stage II.

- **Stage II:** Based on the desired ring topology, the leader designates several Bluetooths as masters by paging them and setting up a temporary piconet. For each designated master, the leader provides it with the information of its slaves, including assigned downstream and upstream bridges. Upon receiving such information, each master pages its slaves and establishes its piconet. A unit serving as a bridge should make sure that both its downstream and

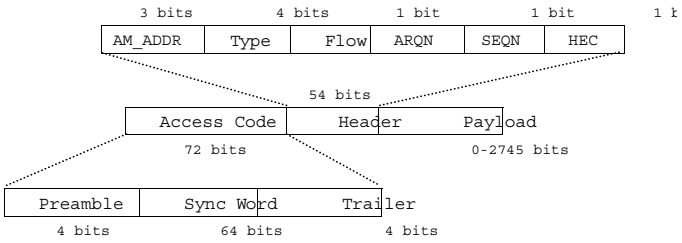


Figure 6: General baseband packet format.

upstream masters have connected to it properly. Once becoming part of the ring, a Bluetooth sets its RING_MEM to 1.

The resultant BlueRing is quite fault-tolerable and scalable. We will discuss the maintenance protocol to handle Bluetooths leaving and joining in Section 5.

4 The BlueRing Routing Protocol

4.1 Unicast and Broadcast

In this subsection, we propose a routing protocol, which supports both unicasting and broadcasting on BlueRing. As mentioned earlier, data packets will be routed following the direction of the BlueRing. Since a packet flowing around the ring will eventually reach its destination piconet, no route discovery process is required. So routing on BlueRing is stateless since no routing table needs to be maintained (on the contrary, most routing protocols for ad hoc networks need to keep routing tables [10]).

To understand how packets are routed, we need to discuss the packet formats in more details. The general Bluetooth baseband data packet format is shown in Fig. 6. Each packet has a 72-bit access code, which can uniquely identify a piconet, followed by a header and a payload. The header carries 18 bits of information, and is encoded by the 1/3 FEC (Forward Error Correction) code, resulting in a 54-bit header. The payload can range from 0 to 2745 bits.

Data packets supported by the ACL (Asynchronous ConnectionLess) link can occupy 1, 3, or 5 time slots. Type DM1/DH1 packets cover a single time slot, type DM3/DH3 packets 3 slots, and type DM5/DH5 packets 5 slots. On the payload field, there is also a payload header. Bluetooth adopts different payload headers for single-slot and multi-slot packets. Fig. 7 details the formats of payload headers.

To route packets on our BlueRing, several control bits should be appended after the payload header. There are three formats for the payload field in BlueRing, depending on their communica-

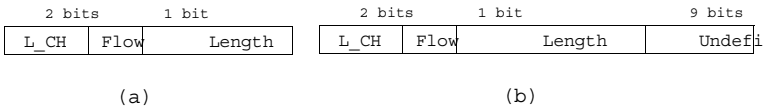


Figure 7: Payload header formats: (a) single-slot packets and (b) multi-slot packets.

tion types, as shown in Fig. 8. These fields (in gray) are explained below.

- **Broadcast bit**: This bit distinguishes broadcast packets from unicast ones. This bit is set to TRUE if the packet needs to be disseminated throughout the whole BlueRing, and set to FALSE otherwise.

- **Relay bit**: For single-hop unicasting, the relay bit is set to FALSE, indicating no relaying is needed to reach the destination. By observing a packet with relay bit set to FALSE, a Bluetooth unit accepts the packet. For multi-hop unicasting packet, its relay bit is set to TRUE, indicating that the packet needs to be relayed to reach the destination. The packet continues to be relayed along the ring, until some master discovers that either the destination is itself or the destination belongs to its piconet. In the former case, the master accepts the packet. In the latter case, the master sets the relay bit to FALSE and forwards the packet to the destination slave. On seeing a packet with relay bit = FALSE, the slave realizes that this is a packet destined for itself, and thus accepts the packet. For broadcasting, if the source is a master, the relay bit is initialized to FALSE and the packet is broadcast to its piconet. All slaves within the piconet will accept the packet. On examining the packet content, the downstream bridge of the sending master will determine that it is a broadcast packet and needs to be further relayed. The downstream bridge will set the relay bit to TRUE and forwards the packet to its downstream master to continue broadcasting. The relay bit (= TRUE) is to inform the downstream master that the received packet should be relayed further. The downstream master should set the relay bit back to FALSE and broadcasts the packet to its piconet. This procedure is repeated to disseminate the packet over the BlueRing. On the other hand, if the source is a slave, the relay bit is initialized to TRUE and the broadcast packet is sent to its master for broadcasting.

- **Dirty bit**: For single-hop unicasting, the dirty bit is set to FALSE. Along with the relay bit (= FALSE), the receiving side can easily tell that the packet is directly from the sending host. For multi-hop unicasting and broadcasting, the dirty bit is to detect the presence

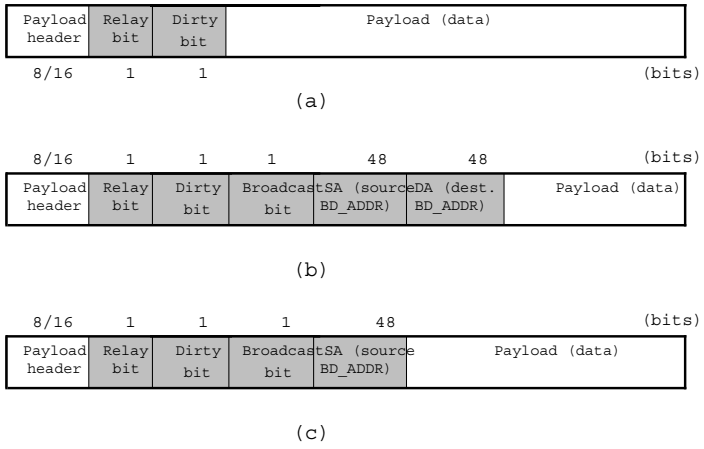


Figure 8: Payload formats in BlueRing: (a) single-hop unicast communication, (b) multi-hop unicast communication, and (c) scatternet broadcast communication. The fields in gray are what added by BlueRing.

of orphan packets (with missing receiver) or duplicate broadcast to prevent packets from endlessly circulating on the BlueRing. Whenever a master touches a packet, the dirty bit is set to TRUE before relaying it to the next hop. This dirty bit, together with the source BD_ADDR, is to identify if a packet has traveled throughout the whole ring or not. If the packet has already traveled around the whole ring once, it is removed from the network to avoid unnecessary circulation. When first initiated, a packet sent by a slave has dirty bit = FALSE, and a packet sent by a master has dirty bit = TRUE. However, the former's dirty bit will be changed to TRUE once being relayed by the first master.

- **SA**: This field contains a 48-bit source Bluetooth Device ADDRess (BD_ADDR).
- **DA**: This field contains a 48-bit destination Bluetooth Device ADDRess (BD_ADDR).

The formal BlueRing routing protocol is provided in Fig. 9 and Fig. 10. Fig. 9 illustrates the operations to be taken by a slave upon receiving a packet. If the relay bit is 1, the packet is directly forwarded to the downstream master without further examining its content. If the relay bit is 0, the packet should be accepted and forwarded to the upper layer. The dirty bit can be used to determine the origin of the packet. If the dirty bit is 0, this packet is directly from the master; otherwise, it has been relayed and its origin can be found from the field SA. In addition, for those slaves acting as downstream bridges, they should check the broadcast bit. If the broadcast bit is 1, the bridge makes a copy of the packet (forwarding it to the upper layer), and sets relay bit to 1. Then the bridge forwards the packet to its downstream master.

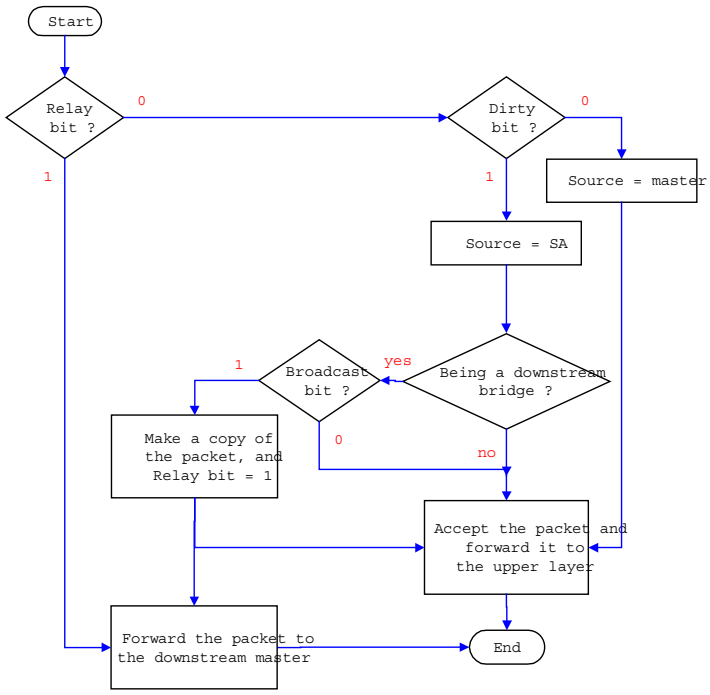


Figure 9: The BlueRing routing protocol for slaves.

Fig. 10 shows the operations to be taken by a master upon receiving a packet. If the relay bit is 0, the packet is accepted. Otherwise, we check the dirty bit. For a packet with dirty bit = 1, we need to check if the SA (source host address) is in this piconet (including the master itself). If so, this is an orphan packet or duplicate broadcast and should be deleted from the network. Also, whenever the first master touches a packet with dirty bit = 0, the dirty bit is set to 1 (so as to detect future orphan packets/duplicate broadcasts). Then depending on the broadcast bit, the master proceeds as follows. For a broadcast packet, the master sets the relay bit to 0, and broadcasts it to its piconet. For a unicast packet, the master needs to decide whether the packet should be forwarded or not. If the DA field is equal to the master itself, the packet is accepted. Otherwise, the DA field is compared to the list of BD_ADDRs belonging to this piconet. If so, the packet is forwarded to host DA in the local piconet; otherwise, the packet is forwarded to the downstream bridge.

Below, we demonstrate several routing examples based on the network in Fig. 5(a). Fig. 11(a) illustrates the packet contents for a single-hop unicast from node 8 to node 7 in piconet M3. Fig. 11(b) and (c) are multi-hop unicasts from node 8 to node 10 and from node 3 to node 9, respectively; the former is an intra-piconet communication, and the latter an inter-piconet commu-

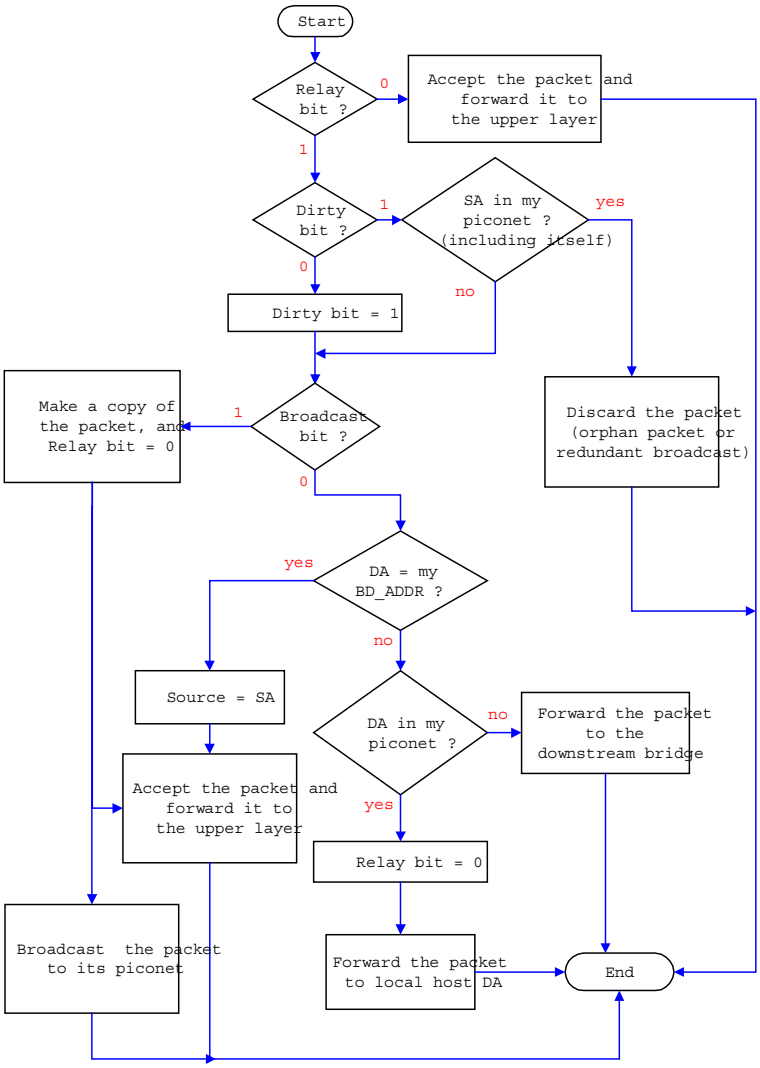


Figure 10: The BlueRing routing protocol for masters.

nication. Finally, Fig. 11 (d) illustrates the packet contents for a scatternet broadcast originated at node 5.

4.2 Bridging Policy

In BlueRing, a bridge host should participate in two piconets. Since Bluetooth is a TDD, FH radio system, a unit can only stay in one piconet at one time. In BlueRing, we propose to use parking and unparking for bridges to switch between piconets. We give the reason why we choose park mode as follows. The Bluetooth specification provides three options for a device to temporarily pause its current activity: sniff, hold, and park modes. The sniff mode has a periodical, prearranged wakeup pattern, and thus is more suitable for a device to switch from piconets to piconets with a regular

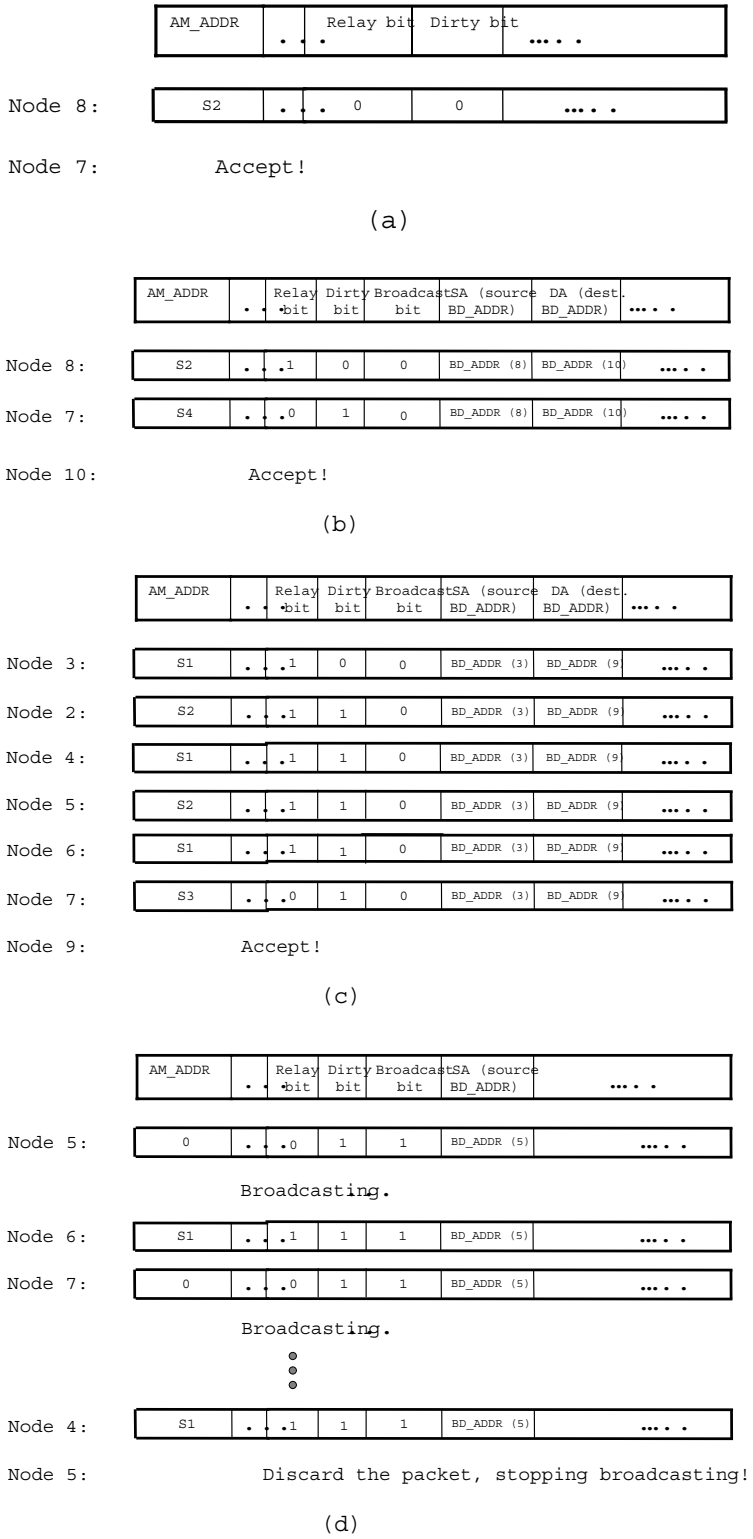


Figure 11: BlueRing routing examples: (a) intra-piconet one-hop unicast (node 8 to node 7), (b) intra-piconet two-hop unicast (node 8 to node 10), (c) inter-piconet multi-hop unicast (node 3 to node 9), and (d) scatternet broadcast originated at node 5.

pattern. It is not selected here because with a regular pattern time slots may easily get wasted. Moreover, with our BlueRing, which chains a sequence of piconets, determining a good sniffing pattern is very difficult. The hold mode would be favorable if the amount of time that a bridge should stay in each piconet can be predetermined. Unfortunately, this assumption is unrealistic, especially in a dynamic environment. The park mode is more favorable in our case since it allows a device to temporarily give up its current activity in one piconet for an arbitrary period of time until an unpark request is issued. The unpark request can be master-activated or slave-activated, but should be approved by the master. Hence, we adopt the park mode in our bridging policy, considering its simplicity and flexibility.

Below, we propose the bridging policy used in our BlueRing. A threshold-based strategy is adopted to initiate park/unpark requests. Three parameters are used in the bridging policy: (1) T_b: a threshold value to evaluate the queued packets in a bridge, (2) T_m: a threshold value to evaluate the queued packets in a master, and (3) T_{out}: a timeout value to evaluate when a bridge should switch piconets. Intuitively, under normal situations, a bridge will connect to its upstream piconet for most of the time. When some threshold conditions become true, it will switch to its downstream piconet. Once connected to its downstream piconet, the bridge will be treated with higher priority by its downstream master so as to drain the packets in its buffer. The detailed switching strategy is described below.

- **From upstream to downstream:** A bridge connecting to its upstream piconet should switch to its downstream piconet when: (i) the number of queued packets to be relayed exceeds T_b, or (ii) the clock T_{out} expires. In this case, a park request should be sent to its upstream master and an unpark request should be sent in the next available access window in the downstream piconet. The downstream master should treat this bridge with higher priority to drain its buffered packets.

- **From downstream to upstream:** A bridge connecting to its downstream piconet should switch to its upstream piconet when: (i) all its buffered packets have been drained by its downstream master, or (ii) its upstream master has queued a number of packets exceeding the threshold T_m. In the former case, the unparking request is initiated by the bridge itself, while in the latter case, the unparking request is initiated by the upstream master to call the slave back. A bridge called by its upstream master should park its current piconet immediately, and switch to the calling piconet channel.

5 The BlueRing Maintenance Protocol

Fault tolerance is an essential issue in packet routing, especially under a mobile environment. In BlueRing, when any master or bridge leaves the network, the ring will become broken and reduce to a linear path. New Bluetooth units may join the network. In this section, we show how to maintain a BlueRing. To tolerate single-point failure, Section 5.1 proposes a protocol-level remedy mechanism. To tolerate multi-point failure, Section 5.2 proposes a recovery mechanism to reconnect the BlueRing. Note that the former one does not try to reestablish the BlueRing so the network may become a linear path. The later one will conduct local reconnection. Graceful failure is tolerable as long as no two or more critical points fail at the same time.

5.1 Single-Point Failure

Suppose that one host serving as a master or a bridge fails. Since there is a default routing direction on BlueRing, a host is unable to reach another host in the backward direction if packets are always sent in the forward direction. The basic idea here is to add a new control bit called *Direction* after the payload header. This bit assists hosts to determine which routing direction (forward/backward) to be followed. Below, we summarize the necessary enhancements on our BlueRing protocol for the fault-tolerant routing.

- The default value for *Direction* is 0, which indicates the forward direction. When a master/bridge on the BlueRing detects that the next hop on the ring is non-existing any more, it simply sets $Direction = 1$ and relays the packet backwards.
- Any master/bridge on receiving a packet with $Direction = 1$ should relay the packet in the backward direction.
- The condition for discarding orphan packets should be revised as follows. Observe that a packet with $Direction = 1$ may reach the source piconet more than once. It is erroneous to discard such a packet by the source piconet master on observing $Dirty = 1$. In this case, the packet should be allowed to continue traveling on the ring until the destination is reached or the other end of the BlueRing is reached. Therefore, the condition for determining a packet to be an orphan should be done by a master/bridge with no upstream node when observing a packet to be undeliverable with $Direction = 1$.

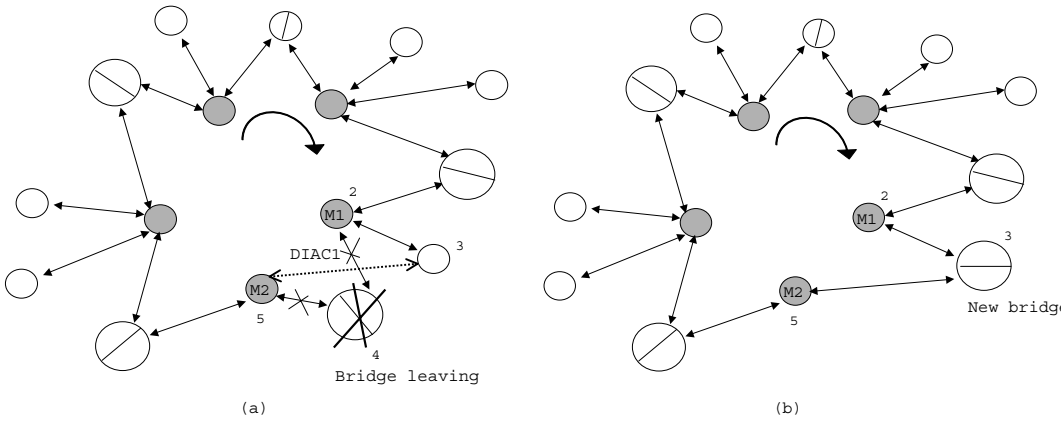


Figure 12: An example to illustrate bridge leaving recovery procedures: (a) DIAC1 discovering and (b) the reconnected BlueRing.

- (Optional) One optimization which can be done here is to have each master keep a list of destination addresses that are unreachable on the forward direction of the BlueRing. On seeing a packet destined to any host in the list, the packet can be sent directly on the backward direction to save communication bandwidth.

Note that if the failure point is a bridge, the whole network remains connected. If a master fails, the non-bridge slaves of the master will become orphans. The other masters should execute the inquiry process from time to time to collect such orphan slaves. The details are in the next subsection.

5.2 Multi-Point Failure

The fault-tolerant routing protocol proposed above ensures routing unaffected, but leaving the broken point unfixed. In this subsection, we propose a recovery mechanism that can reconnect the network as a BlueRing. New hosts can join an existing BlueRing too. Only local reconnection is required. As long as no two critical points fail simultaneously, the protocol can work correctly.

Recall that Bluetooth provides 63 reserved DIACs for discovering certain dedicated units in range. Here, we propose to use 2 reserved DIACs, say DIAC1 and DIAC2, to facilitate BlueRing recovery. Also, the GIAC will be used to invite new hosts to join an existing BlueRing.

Below, we show how to manage cases of bridge leaving and master leaving. In some cases we will have to extend a BlueRing by creating more piconets.

- **Bridge Missing:** When a bridge leaves, its downstream master performs DIAC1 inquiry, hoping to connect with another upstream bridge. On the other hand, the leaving bridge's

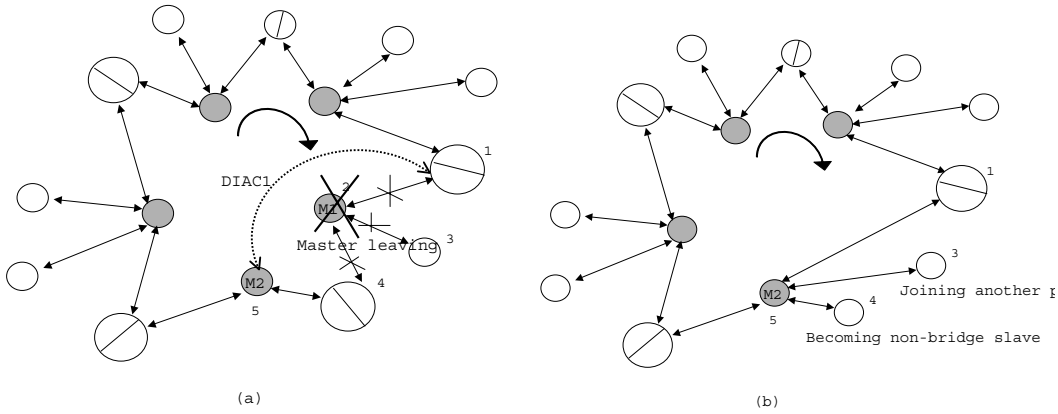


Figure 13: An example to illustrate master leaving recovery procedures: (a) DIAC1 discovering and (b) the reconnected BlueRing.

upstream master checks if it has any other non-bridge slaves that can serve as its new downstream bridge. If this is the case, the master notifies this bridge such and commands it to enter DIAC1 inquiry scan. Otherwise, the upstream master should tear down its current piconet and wait to be discovered by other masters. This case will induce a missing master, which can be cured by the "**Master Missing**" procedure in the subsequent paragraph. Fig. 12 illustrates the recovery procedures when the bridge node 4 leaves. Upstream master node 2 reassigns node 3 as its new downstream bridge and then node 3 enters DIAC1 inquiry scan. Meanwhile, the downstream master node 5 performs DIAC1 inquiry, and discovers node 3. A new connection is formed between nodes 5 and 3, healing the BlueRing.

- **Master Missing:** When a master leaves, all of its slaves, except the downstream and upstream bridges, become orphans. The downstream bridge of the leaving master should change its state to a non-bridge and inform its downstream master to perform DIAC1 inquiry, in hope of finding a new upstream bridge. On the other hand, the upstream bridge of the leaving master enters DIAC1 inquiry scan, hoping to be discovered. Fig. 13 shows the scenarios when master node 2 leaves, leaving node 3 isolated from the ring. The downstream bridge node 4 reduces a non-bridge slave and informs its downstream master node 5 to execute DIAC1 inquiry. Upstream bridge node 1 starts DIAC1 inquiry scan, and is discovered by node 5. A new connection is established between nodes 5 and 1. Moreover, by GIAC inquiry/inquiry scan, node 3 is discovered and invited to join node 5's piconet, thus becoming part of the ring again. So the BlueRing is reconnected.

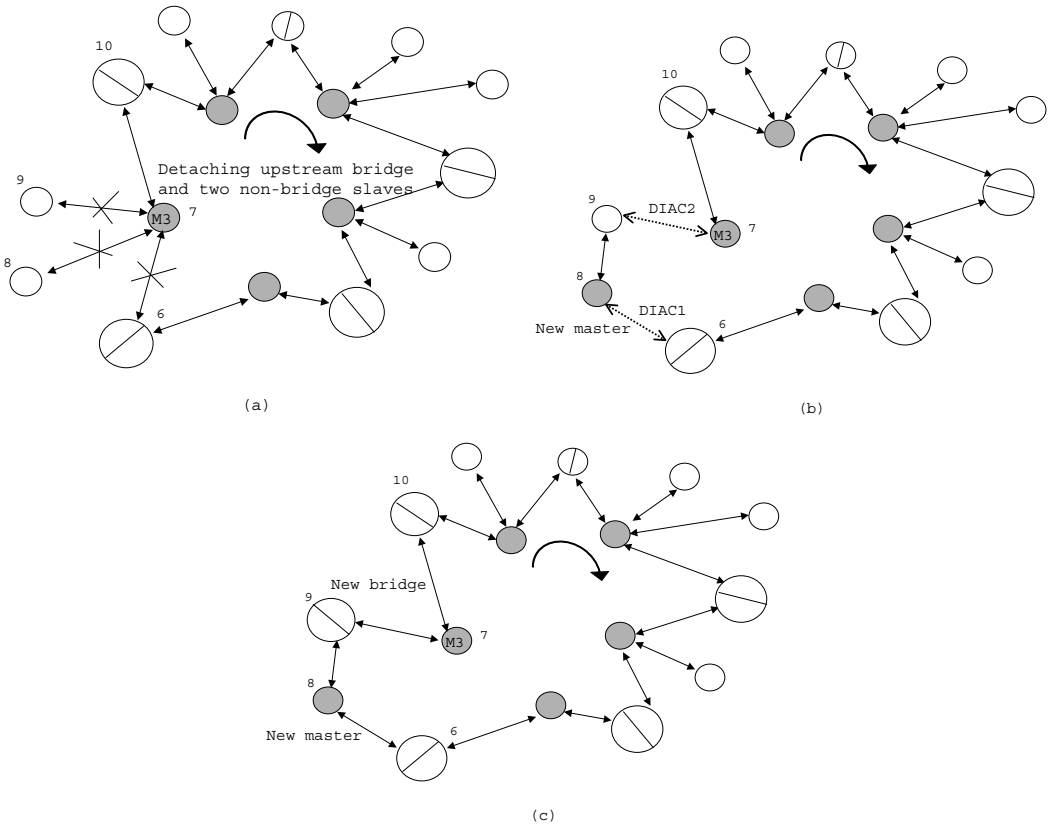


Figure 14: A BlueRing extension example with $\alpha = 4$.

- **Piconet Splitting:** Recall that in order to include more Bluetooths into the BlueRing, each master should execute GIAC inquiry from time to time. This happens when new Bluetooth units join or orphan Bluetooth units appear. When the number of slaves belonging to a master exceeds a certain limit, we will split it into two piconets. The procedure is as follows. Assume that the desirable maximum number of slaves per piconet is α ($\alpha \geq 4$). Whenever the number of slaves a master possesses reaches α, the master sends out a *split_request* token to obtain split permission from all other masters. The *split_request* packet traverses the ring to ensure that concurrent splitting operations on the BlueRing do not exist. Once the split request is approved by all piconets on the ring, the master detaches its upstream bridge and two non-bridge slaves (this must succeed since $\alpha \geq 4$). Then the master starts DIAC2 inquiry. Upon discovering master missing, the upstream bridge enters DIAC1 inquiry scan in search of new downstream master. On the other hand, one of the two detached non-bridge slaves is designated as new master and provided with the information of the other detached slave, which is informed to enter page scan. The former will then page the later, thus setting up a new piconet consisting of two members (one master and one slave). The new master designates

Table 1: Routing distances for packets originated at (a) a non-bridge slave, (b) a master, and (c) a bridge.

	No. of destinations	Dista
Intra-piconet-slave	$c_{s1} = S-3$	$d_{s1} = 2$
Ring-body	$c_{s2} = R-2$	$d_{s2} = R+1$
Inter-piconet-slave	$c_{s3} = S(R-1)$	$d_{s3} = R+2$

(a) Source is a non-bridge slave

	No. of destinations	Dista
Intra-piconet-slave	$c_{m1} = S-2$	$d_{m1} = 1$
Ring-body	$c_{m2} = R-1$	$d_{m2} = R$
Inter-piconet-slave	$c_{m3} = S(R-1)$	$d_{m3} = R+1$

(b) Source is a master

	No. of destinations	Dista
Intra-piconet-slave	$c_{b1} = S-2$	$d_{b1} = 2$
Ring-body	$c_{b2} = R-1$	$d_{b2} = R$
Inter-piconet-slave	$c_{b3} = S(R-1)$	$d_{b3} = R+1$

(c) Source is a bridge

its only slave as its downstream bridge, and starts DIAC1 inquiry to discover an upstream bridge. Meanwhile, the new master orders its downstream bridge to enter DIAC2 inquiry scan. Fig. 14 shows a splitting example, assuming $\alpha = 4$. After obtaining the permission to split, master node 7 disconnects from nodes 6, 8, and 9, and designates node 8 as the new master. Immediately, node 8 sets up a new piconet with node 9, which serves as its downstream bridge. Then node 8 discovers node 6 by DIAC1 inquiry, while node 7 discovers node 9 via DIAC2 inquiry. The ring is now connected again with one more piconet. By creating more piconets, more new Bluetooths can join the ring, making BlueRing extensible.

We remark on two points. First, every split operation needs to detach 2 non-bridge slaves. Plus downstream and upstream bridges, only piconets with no less than 4 slaves can request for splitting. That explains why we enforce $\alpha \geq 4$. Second, the *split_request* token should compete with other potential *split_request* tokens while traveling on the ring. The can be easily done by allowing a requesting master with a higher BD_ADDR to inhibit other requesting masters' tokens with lower BD_ADDRs.

6 Analysis and Simulation Results

In this section, we first evaluate the maximum achievable throughput of the BlueRing. Let the ring contain R piconets ($R \geq 2$) and each piconet contain S slaves (including both non-bridge slaves and bridges). So the total number of Bluetooths on the ring is $S \cdot R$. We shall derive the average number

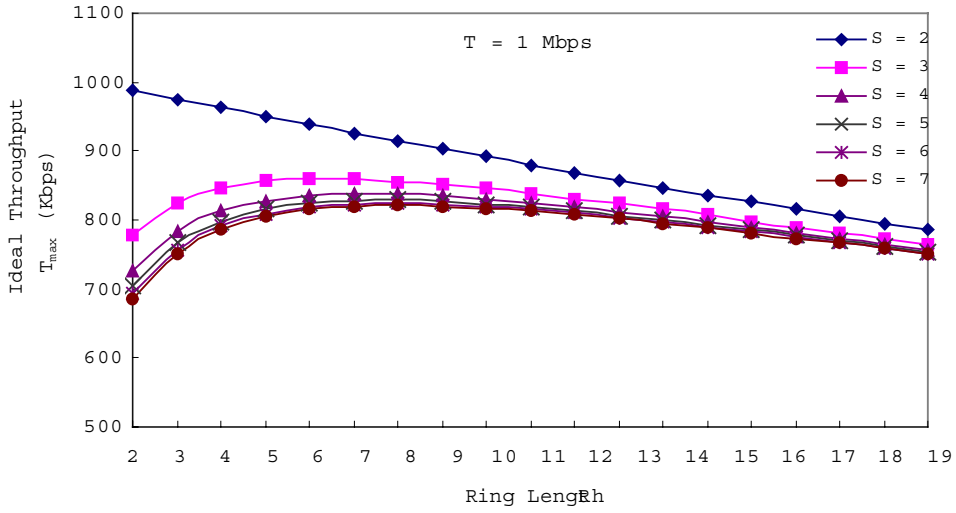


Figure 15: Ideal BlueRing throughput for different R's and S's.

of hops that a data packet needs to travel before reaching its destination. This will depend on the roll of the source, which can be master, bridge slave, and non-bridge slave. Table 1 summarizes these three cases. In the tables, destinations are classified into ring-body, intra-piconet-slave, and inter-piconet-slave. Note that the ring-body contains all masters and bridges, and the rest of the slaves are classified into intra- and inter-piconet cases.

Based on Table 1, we derive the average distances d_s, d_m, and d_b for packets originated at a non-bridge slave, master, and bridge, respectively, as follows:

$$d_s = \frac{c_{s1} \cdot d_{s1} + c_{s2} \cdot d_{s2} + c_{s3} \cdot d_{s3}}{c_{s1} + c_{s2} + c_{s3}} = \frac{SR^2 + SR - 2}{SR - 1} \tag{1}$$

$$d_m = \frac{c_{m1} \cdot d_{m1} + c_{m2} \cdot d_{m2} + c_{m3} \cdot d_{m3}}{c_{m1} + c_{m2} + c_{m3}} = \frac{SR^2 - R}{SR - 1} = R \tag{2}$$

$$d_b = \frac{c_{b1} \cdot d_{b1} + c_{b2} \cdot d_{b2} + c_{b3} \cdot d_{b3}}{c_{b1} + c_{b2} + c_{b3}} = \frac{SR^2 + (S-3)R}{SR - 1} \tag{3}$$

Since there are $(S-2)R$ non-bridge slaves, R masters, and R bridges, the average traveling distance can be derived as:

$$D_{avg} = \frac{(S-2) \cdot R \cdot d_s + R \cdot d_m + R \cdot d_b}{SR} = \frac{S^2R^3 + (S^2 - S - 4)R^2 + (4 - 2S)R}{S^2R^2 - SR}. \tag{4}$$

Taking $S = 7$ and $R = 3$ for example, the average traveling distance D_{avg} will be 3.89.

The following analysis further considers interference between piconets. Assume that the maximum throughput of a single piconet under an interference-free environment is T. By extending to a scatternet, different piconets which choose the same FH channel in the same time slot result in a

collision. Given that 79 frequencies are available in Bluetooth, the probability that a time slot of a piconet suffers no interference, denoted by P_S, can be approximated by $(78/79)^{R-1}$, where R is the number of piconets in the transmission range of each other.

The available network bandwidth is $T \cdot R \cdot P_S$. Dividing this by the average traveling distance D_{avg}, we obtain the maximum achievable throughput T_{max} of BlueRing:

$$T_{max} = \frac{T \cdot R \cdot P_S}{D_{ave}}. \tag{5}$$

Using the Bluetooth nominal bandwidth $T = 1$ Mbps, $S = 7$, and $R = 3$, we can compute $T_{max} = 751$ Kbps. Fig. 15 shows the ideal throughput T_{max} by varying R and S. From the curves, it can be seen that a BlueRing length of $R = 5 \sim 8$ would be quite cost-effective.

In the following, we present our simulation results to verify the above theoretical analysis and to compare our BlueRing with other scatternet topologies. We simulate only DH1 data packets. For a single-slot DH1 packet, 336 bits (including the access code, header, payload header, payload, and CRC) are transmitted over a time slot with 625μs duration, which contributes to a reduced $T = 538$ Kbps throughput/piconet. For simplicity, we assume that collisions due to frequency overlapping do not happen, and thus $P_S = 1$. Taking a 21-node BlueRing ($S = 7$, $R = 3$) for instance, we obtain $T_{max} = 415$ Kbps, which predicts the saturation point in throughput. This can be used to verify the correctness of our analysis.

In our simulation, the number of Bluetooth devices that may participate in the BlueRing could be $N = 14$, 21, 28, 35 or 42. For each simulation instance, we initiate $N/3$ data-link connections, each with a randomly chosen source-destination pair. Each connection is an ACL link and can be an intra- or inter-piconet communication. We also vary the ratio of the numbers of intra- to inter-piconet connections. The numbers of intra- to inter-piconet connections could be equal, more intra-connections, or more inter-connections. We will evaluate the influence of this factor. For each connection, we assign it one of three data arrival rates, 256Kbps, 128Kbps, and 16Kbps, with equal possibility. A master keeps a separate buffer queue for each of its slaves. No mobility is modeled. Besides, physical properties such as fading and interference are not considered. Each simulation run lasts for 75 seconds. Only DH1 data packets are simulated.

The bridging policy proposed in Section 4.2 is followed. Switching between piconets is realized using park/unpark procedures by following the proposed control message exchanges. We set the buffering threshold to 60% for bridges to switch between piconets. The maximum number of slaves per piconet is set to S. This factor will affect the ring length as well as packet delay. Two

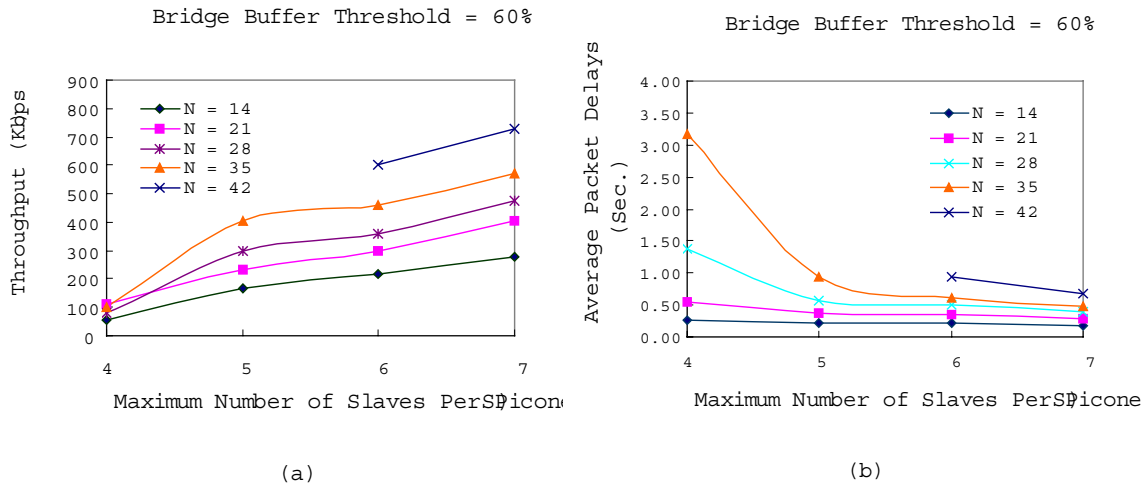


Figure 16: Effect of ring length on BlueRing.

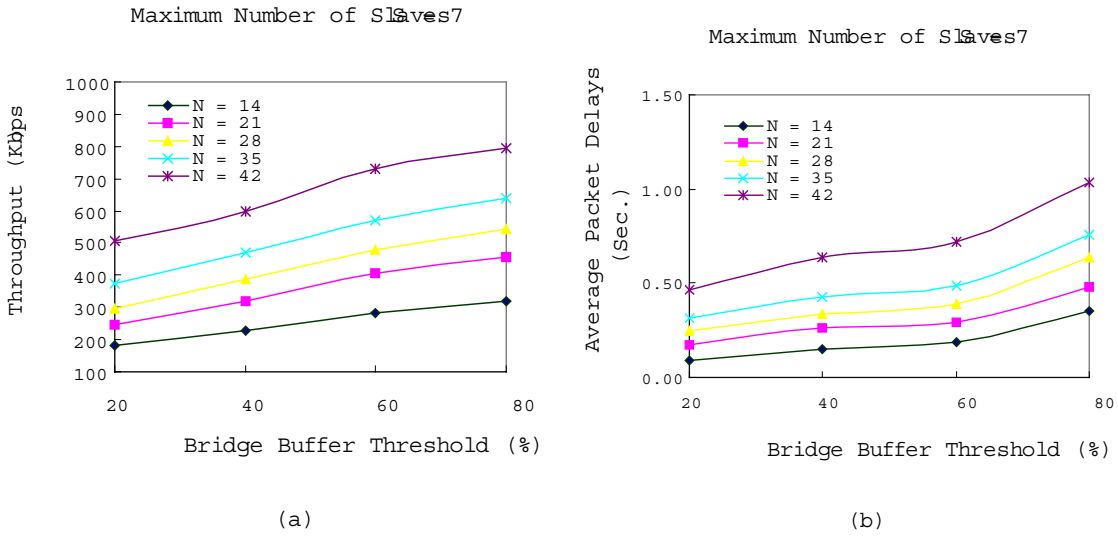


Figure 17: Effect of bridge buffer threshold on BlueRing.

performance metrics are observed: *throughput* and *average packet delays*. In Section 6.1, we first study the impact of several BlueRing-related parameters. Then in Section 6.2 we present some performance comparison results with other scatternet structures.

6.1 Tuning BlueRing-Related Parameters

In this subsection, we investigate three factors that may affect the performance of BlueRing: N (network size), S (maximum number of slaves per piconet), and bridge buffer threshold.

Fig. 16 illustrates the throughput and average packet delays against S when $N = 14$, 21, 28, 35, and 42. In all values of N, we observe that the throughput increases and the packet delay decreases as S grows. When N is fixed, a larger S implies a shorter ring. So this indicates that a shorter

ring length can result in higher network throughput and lower packet delay.

The bridge buffer threshold affects when a bridge node should switch to its downstream piconet. By setting the threshold to 20%, 40%, 60%, and 80% of the total buffer size, Fig. 17(a) shows that the network throughput will increase slightly as the threshold goes up. This is because a smaller threshold will incur more switches (and thus more switching overheads). On the other hand, Fig. 17(b) also shows that the average packet delay will increase as the threshold grows, due to longer queuing delays on bridges. Hence, the threshold value should be properly set to balance both network throughput and packet delay.

6.2 Performance Comparison with Other Scatternet Structures

We compare BlueRing with two other scatternet structures. The first one is a simple *single-piconet* structure. According to the Bluetooth specification, at most 7 active slaves can be supported in a single piconet. To support more than 7 slaves, the extra slaves must enter the *park* mode. In our implementation, we let the channel be shared by slaves in a round-robin manner. In other words, communicating entities are parked/unparked periodically, taking turns to access the channel. So many extra control packets exchanges will take place. The second structure is the *star-shaped* scatternet proposed in [8]. One piconet is placed in the center. Each slave of the central piconet may be connected to another master, and if so, will act as a bridge of the two piconets. Non-central piconets do not extend to more piconet. So this can be regarded as a two-level hierarchy. All inter-piconet traffic must go through the central piconet, and thus this may present a traffic bottleneck, but the benefit is a less average number of hops that packets have to go through for inter-piconet communications. Although the bridging policy is not specified in [8], here we adopt our threshold-based policy in Section 4.2 by regarding the central piconet as upstream, and non-central piconets as downstreams. When $N = 20$, Fig. 18 compares our BlueRing with the star-shaped scatternet.

Fig. 19 demonstrates the network throughput and packet delay under different traffic loads. Here, load is reflected by the number of connections initiated. Each connection has a data rate of 256 Kbps, and could be an intra- or inter-piconet commuincation. Fig. 19(a) shows that BlueRing saturates at the highest point compared to the other two structures. The saturated throughput is about 415 Kbps, which is consistent with the prediction of our analysis. For the star-shaped structure, its throughput outperforms that of the single-piconet model when the number of simultaneous connections exceeds 6. This is because multi-piconet has the advantage of using multiple frequency-hopping channels at the same time. Fig. 19(b) demonstrates that, when the number

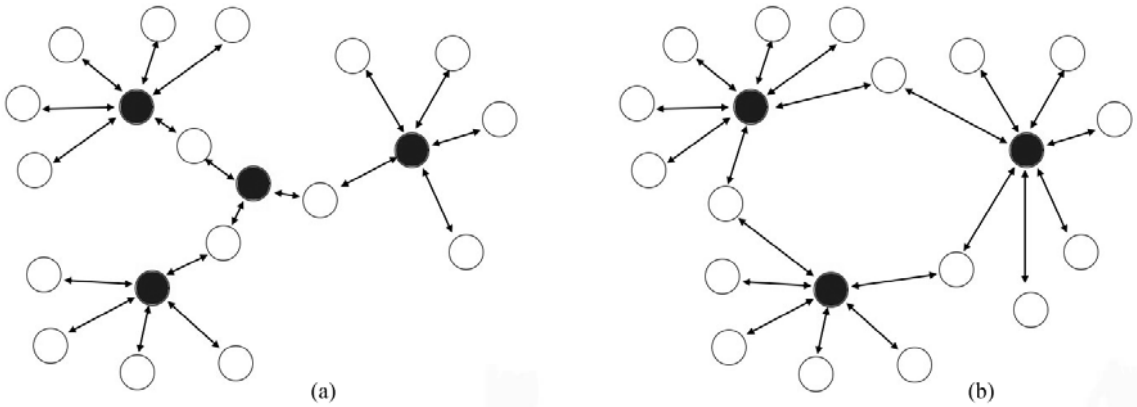


Figure 18: Two scatternet topologies with $N = 20$ hosts: (a) star-shaped structure and (b) BlueRing (black nodes are masters).

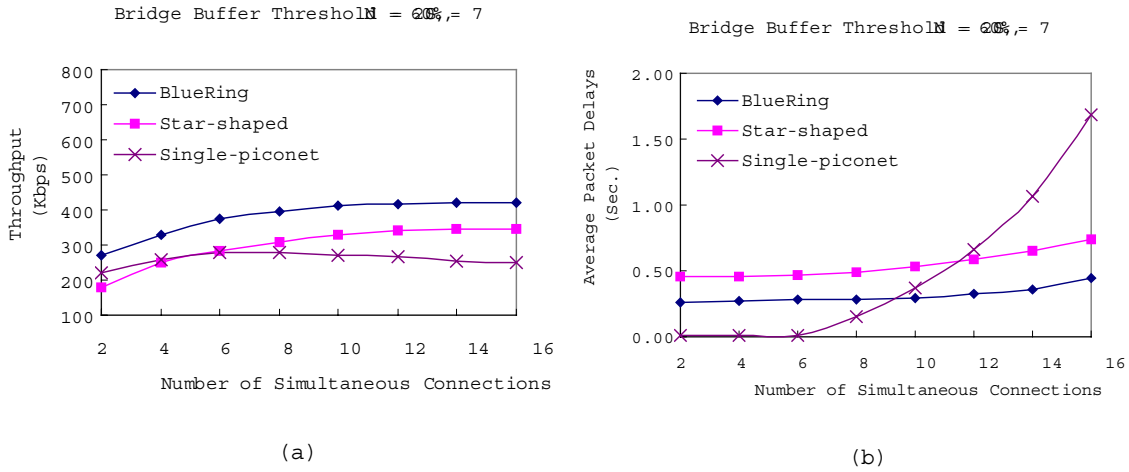


Figure 19: Performance comparison by varying the number of connections: (a) throughput and (b) packet delay.

of simultaneous connections is below 10, both BlueRing and star scatternet suffer higher delays compared to the single-piconet case due to bridging costs and larger network diameters. However, when traffic load becomes higher, the packet delay of single-piconet structure raises dramatically. The reason is that the throughput of single-piconet structure has reached a saturated point, which leads to significant increase of packet delays.

We also investigate the impact of different intra- to inter-piconet connection ratios on the network performance. Note that with more inter-piconet connections, the traffic load is higher. Fig. 20 compares the throughput and average packet delay of the three scatternet structures under different traffic loads. Here the ratio of intra-picont to inter-piconet traffic is 3:1. The figure shows that BlueRing yields the highest throughput with moderate packet delay. For the single-

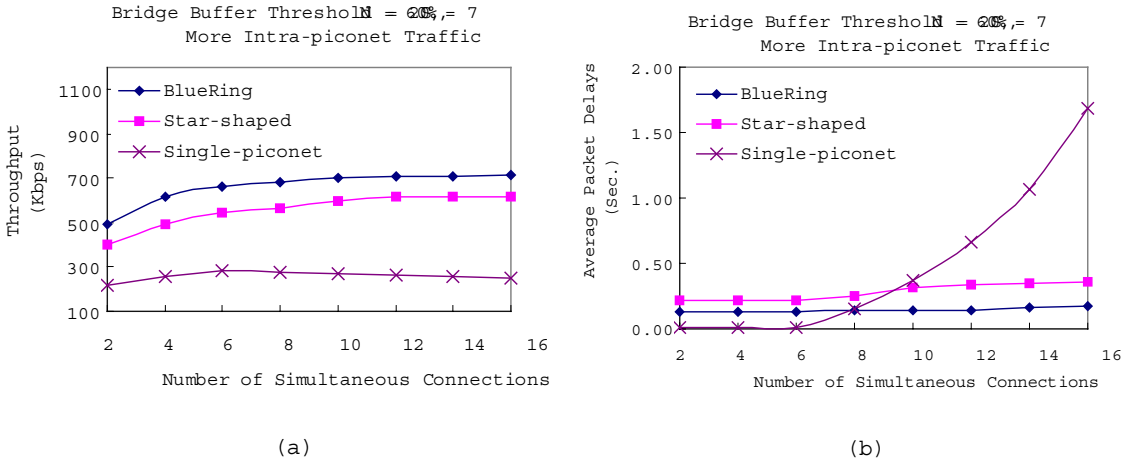


Figure 20: Performance comparison by varying the number of connections (with more intra-piconet traffic): (a) throughput and (b) packet delay.

piconet structure, the throughput saturates when the number of simultaneous connections reaches 6. This is because many time slots are wasted on exchanging control packets for park/unpark procedures. Furthermore, a single piconet can only utilize one frequency-hopping sequence, and thus the maximum frequency utilization is only 1/79. On the other hand, BlueRing may utilize multiple FH sequences over the same space. This is what limits the capability of the single-piconet structure. With more intra-piconet connections, the BlueRing structure allows more simultaneous transmissions and shorter routing distances. For the star-shaped structure, the throughput is between those of BlueRing and single-piconet structures. With more intra-piconet connections, the bottleneck effect, incurred by the central piconet, of the star-shaped scatternet becomes less significant.

Fig. 21 illustrates the case when there are more inter-piconet connections (with intra- to inter-piconet connection ratio equal to 1:3). The performance of the star-shaped structure drops significantly due to more serious bottleneck effect. For BlueRing, the throughput also declines, but is still superior to the star-shaped structure. The single-piconet structure remains unaffected since all traffic is intra-piconet.

7 Conclusions

In this paper, we have designed the corresponding formation, routing, and topology maintenance protocols for BlueRing. Due to BlueRing's simplicity and regularity, routing on it is *stateless*, in the sense that no routing table needs to be kept by any host (and thus no route discovery procedure

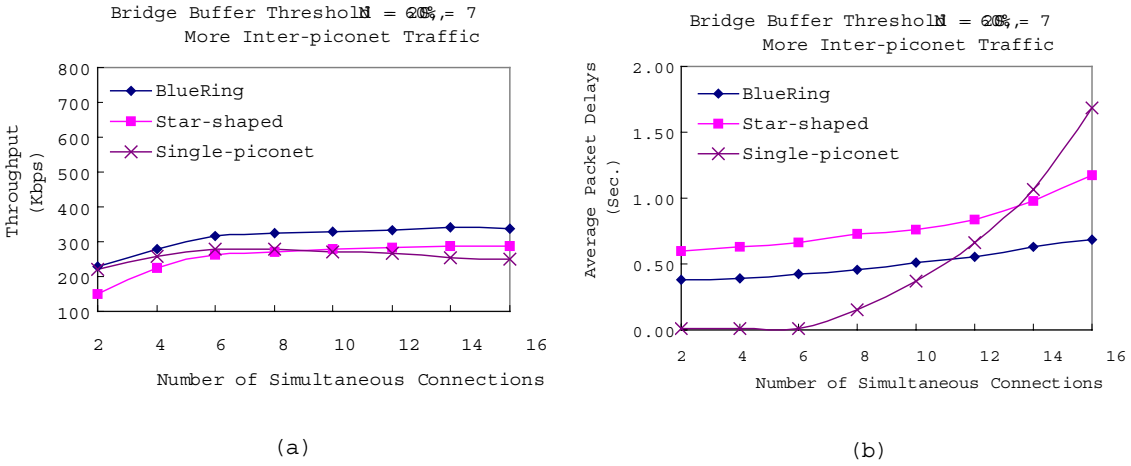


Figure 21: Performance comparison by varying the number of connections (with more inter-piconet traffic): (a) throughput and (b) packet delay.

needs to be conducted prior to sending any packet). A protocol-level remedy is developed to keep the network alive when there is a single-point failure on the ring. To tolerate multi-point failure, a recovery protocol is devised to reconnect the BlueRing. We believe that the important fault-tolerant issue has not been properly addressed by existing proposed scatternet protocols. To demonstrate the scalability of BlueRing with respect to network size, analyses and simulation experiments have been conducted. The results do indicate that BlueRing outperforms other network structures, such as single-piconet and star-shaped scatternet, with higher throughput and moderate packet delay. To conclude, we believe that the BlueRing is an efficient topology in terms of both network performance and fault-tolerant capability.

Our future works include analyzing the fault tolerance capability of BlueRing and devising mechanisms to deal with more than one simultaneous failure on the ring. Moreover, a real implementation of BlueRing is also being planned in the National Chiao-Tung University and will be reported in our future work.

References

[1] Bluetooth SIG Bluetooth Specification v1.1, http://www.bluetooth.com. February, 2001.

[2] AU-System Bluetooth Whitepaper 1.1, http://www.ausystem.com. January, 2000.

[3] P. Bhagwat and A. Segall. A Routing Vector Method (RVM) for Routing in Bluetooth Scatternets. *IEEE Int'l Workshop on Mobile Multimedia Communications (MoMuC)*, 1999.

[4] D. Groten and J. Schmidt. Bluetooth-based Mobile Ad Hoc Networks: Opportunities and Challenges for a Telecommunications Operator. *IEEE Vehicular Technology Conference (VTC)*, 2001.

[5] P. Johansson, M. Kazantzidis, R. Kapoor, and M. Gerla. Bluetooth: An Enabler for Personal Area Networking. *IEEE Network*, pages 28–37, September/October 2001.

[6] M. Kalia, S. Garg, and R. Shorey. Scatternet Structure and Inter-Piconet Communication in the Bluetooth System. *IEEE National Conference on Communications*, New Delhi, India, 2000.

[7] C. Law, A. K. Mehta, and K.-Y. Siu. Performance of a New Bluetooth Scatternet Formation Protocol . *ACM Int'l Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2001.

[8] W. Lilakiatsakun and A. Seneviratne. Wireless Home Networks based on a Hierarchical Bluetooth Scatternet Architecture. *IEEE Int'l Conference on Networks (ICON)*, 2001.

[9] G. Miklos, A. Racz, Z. Turanyi, A. Valko, and P. Johansson. Performance Aspects of Bluetooth Scatternet Formation. *ACM Int'l Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2000.

[10] C. E. Perkins. Ad Hoc Networking. *Addison-Wesley*, 2001.

[11] L. Ramachandran, M. Kapoor, A. Sarkar, and A. Aggarwal. Clustering Algorithms for Wireless Ad Hoc Networks. *ACM DIAL M Workshop*, pages 54–63, 2000.

[12] T. Salonidis, P. Bhagwat, and L. Tassiulas. Proximity Awareness and Fast Connection Establishment in Bluetooth. *ACM Int'l Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2000.

[13] T. Salonidis, P. Bhagwat, L. Tassiulas, and R. LaMaire. Distributed Topology Construction of Bluetooth Personal Area Networks. *IEEE INFOCOM*, 2001.

[14] G. V. Zaruba, S. Basagni, and I. Chlamtac. Bluetrees - Scatternet Formation to Enable Bluetooth-Based Ad Hoc Networks. *IEEE Int'l Conference on Communications (ICC)*, 2001.

Analysis of Bluetooth Device Discovery and Some Speedup Mechanisms*

Jehn-Ruey Jiang§, Bing-Rong Lin†, and Yu-Chee Tseng†

§Department of Computer Science and Information Engineering
National Central University, Taiwan

†Department of Computer Science and Information Engineering
National Chiao-Tung University, Taiwan

jrjiang@hcu.edu.tw, linnbiro@csie.nctu.edu.tw, yctseng@csie.nctu.edu.tw

Abstract

The device discovery time of Bluetooth is prohibitively long. This may significantly impact many mobile applications. In this work, we start by analyzing the frequency-matching delay of Bluetooth for both versions V1.1 and V1.2. We then propose three schemes to speed up the device discovery procedure of Bluetooth. The result is a significant reduction of average frequency-matching time from 23.55 seconds to 11.38 seconds.

Keywords: Bluetooth, device discovery, frequency-hopping spread spectrum (FHSS), inquiry and scan, wireless network.

1 Introduction

Bluetooth [2] is a promising technology for short-range, low-power wireless communications. Operating in the 2.4GHz license-free ISM (Industrial, Scientific-Medical) band, Bluetooth adopts a 79-channel *Frequency Hopping Spread Spectrum (FHSS)*[1] technology with a hopping rate of 1600 hops per second. In Bluetooth, before any two devices can communicate with each other, they must go through a device discovery procedure which consists of two steps, *inquiry* and *paging*. The former is for devices to find each other, while the latter is to establish actual connections. According to the specification [2], the inquiring procedure may take 10.24 seconds or longer, and the paging, 7.68 seconds or longer. This long connection setup time is fine for static applications, but is intolerable for mobile applications demanding quick and short connections, such as multi-media name card exchange [4] and pedestrian surroundings information retrieval [9]. Consequently, many approaches [1, 4, 5, 6, 7, 8, 9] have been proposed to speed up the Bluetooth device discovery procedure.

One major component in the discovery delay is the long *frequency-matching* time. Bluetooth adopts a master-slave architecture. To establish a connection between two devices, a potential master should be in the *inquiry* state

*Y. C. Tseng's research is co-sponsored by the MOE Program for Promoting Academic Excellence of Universities under grant number 89-E-FA04-1-4, by NSC of Taiwan under grant numbers NSC92-2213-E009-076 and NSC92-2219-E009-013, by the Institute for Information Industry and MOEA, R.O.C, under the Handheld Device Embedded System Software Technology Development Project, and by the Lee and MTI Center of NCTU.

[1]The number of channels may be reduced to 23 in certain countries.

to periodically send consecutive *ID* packets on some predefined 32 channels (or frequencies[2]), and a potential slave should be in the *inquiry scan* state trying to catch an *ID* packet from the right channel at the right time. Only when a frequency-matching occurs, i.e., the slave correctly receives an ID packet, can the inquiry-paging procedure be started.

A lot of works [3, 4, 6, 7, 8, 9, 10] have addressed the Bluetooth device discovery speedup problem. Some [4, 6, 7, 9] suggest to modify the device discovery parameters, some [3, 10] suggest to use auxiliary devices, while some [8] relies on device cooperation to assist device discovery. The recent Bluetooth specification V1.2 also proposes a "faster connection" based on the concept of interlaced inquiry scan frequencies.

In this work, we start by analyzing the frequency-matching time of Bluetooth, the major component of delay in its device discovery, for both versions V1.1 and V1.2. We show through analysis that the average delay is about 23.55 seconds. This motivates us to search for schemes to shorten the frequency-matching time. In this paper, three schemes are proposed. The reduction is shown to be significant.

The rest of this paper is organized as follows. Section 2 presents some backgrounds. In Section 3, we analyze the frequency-matching delay of Bluetooth's device discovery. Section 4 presents our schemes. Concluding remarks are drawn in Section 5.

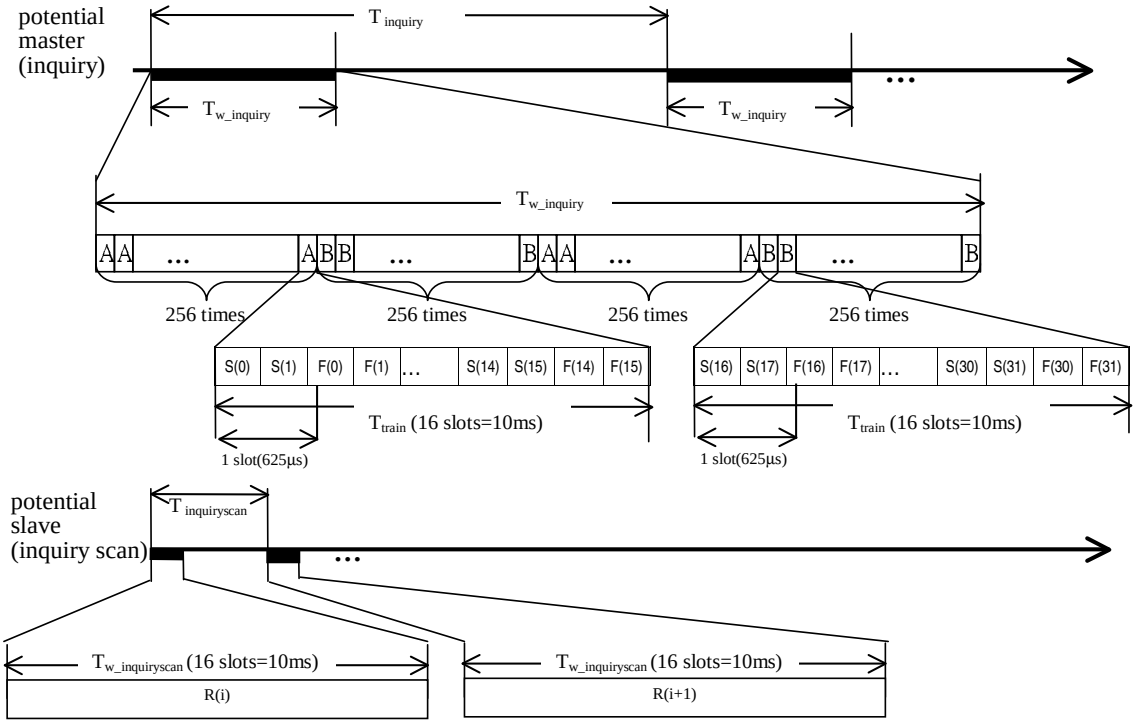
2 Backgrounds

2.1 Inquiry and Paging Procedures of Bluetooth

The device discovery in Bluetooth involves two steps: *inquiry* and *paging*. The inquiry procedure is asymmetric. A potential master must enter the INQUIRY state first, and a potential slave must enter the INQUIRY SCAN state. The master will periodically broadcast ID packets in every $T_{inquiry}$ interval (refer to Fig. 1). These ID packets are hopping on 32 common channels. These 32 channels are divided into two sets, each with 16 channels. ID packets are grouped into *A trains* and *B trains*, each using one of the two sets of 16 channels exclusively. In a $T_{w\_inquiry}$ interval, $N_{inquiry}$ A trains, followed by $N_{inquiry}$ B trains, $N_{inquiry}$ A trains, and $N_{inquiry}$ B trains of ID packets are sequentially transmitted, where $N_{inquiry} = 256$. Each train consists of 16 slots (of length $T_{train} = 10$ ms). Two ID packets on two different channels are placed in one 625-μs slot. So there are 8 slots of ID packets interleaved by 8 response slots reserved for slaves to reply. Consequently, $T_{w\_inquiry}$ takes up to 10.24 seconds to complete (4×256 of A/B trains, each of 10 ms), unless the master has collected enough ($\geq N_{inquiry\_responses}$) responses and determines to abort the INQUIRY procedure earlier. For example, one commonly selected setting is that masters enter the INQUIRY state every one minute, i.e., $T_{inquiry}$=60 sec.

A potential slave should enter the INQUIRY SCAN state to listen to ID packets (refer to Fig. 1). It sequentially hops on the aforementioned 32 channels, but at a much slower speed. It takes $T_{inquiryscan}$ seconds to hop from one channel to another. In each hop, it only enters the listening status for $T_{w\_inquiryscan}$=10 ms. Note that it is necessary that $T_{w\_inquiryscan} \geq T_{train}$ so as to guarantee that the slave can catch an ID packet from the master. The Bluetooth specification suggests that $T_{inquiryscan}$ be no longer than 2.56 seconds, which equals the length of $N_{inquiry}$ A/B trains. Note that many vendors set $T_{inquiryscan} = 1.28$ seconds, which will also be adopted in this paper. Table 1 summarizes all the above timing parameters.

[2]In this paper, the word "channel" and the word "frequency" are used interchangeably.



S(*i*) stands for sending ID packet in inquiry hopping frequency channel *i*, *i*=0..31.
R(*i*) stands for listening to ID packet in inquiry hopping frequency channel *i*, *i*=0..31.
F(*i*) stands for listening to FHS packet in inquiry hopping frequency channel *i*, *i*=0..31.

Figure 1. Bluetooth inquiry procedure.

Table 1. Timing parameters of inquiry and inquiry scan.

Parameter	Description	Recommended value
$T_{inquiry}$	inquiry interval	60s
$T_{w\_inquiry}$	inquiry window length	10.24s
$T_{inquiryscan}$	inquiry scan interval	1.28s
$T_{w\_inquiryscan}$	inquiry scan window length	10ms
T_{train}	length of a train	10ms
$N_{inquiry}$	train repetition number	≥ 256

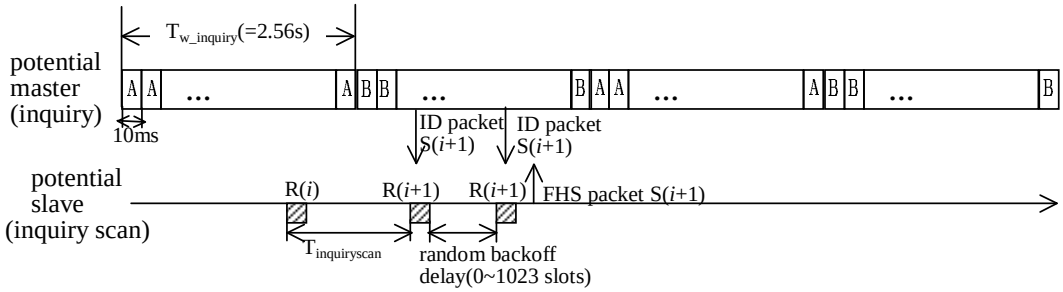


Figure 2. The backoff procedure for a slave to reply a FHS packet.

Upon receiving an ID packet from some channel, say i, a slave should take a random backoff and then reply a *Frequency Hopping Synchronization (FHS)* packet via the same channel. The backoff value is between 0 to 1023 slots to avoid possible collisions with other slaves. After the backoff, the slave should continuously listen to channel i and reply a FHS immediately after the first ID packet (also on channel i) is heard. Fig. 2 illustrates this procedure. Note that the average backoff value is 512 slots, which equals 32 trains. This explains why A/B trains need to be repeated so many times.

2.2 Related Work

Several methods have been proposed to improve the Bluetooth device discovery procedure [3, 4, 6, 7, 8, 9, 10]. Some schemes try to modify the device discovery parameters [4, 6, 7, 9]. Some schemes propose to use auxiliary devices [3, 10], while some relies on device cooperation to assist the discovery [8].

In [9], three methods are proposed. The first method tries to decrease or even eliminate the random backoff in INQUIRY SCAN, the second method uses one single 32-frequency train to replace the two 16-frequency trains in INQUIRY, and the last method is a hybrid one to combine the first two methods. According to [9], these methods can improve the connection setup time up to 75% without deteriorating the overall system performance. A hardware empirical testbed is developed to verify these methods in [6]; the result suggests that a single train with no backoff has the best performance. In [4, 7], each device is assumed to alternate between "potential master" and "potential slave" modes in a random fashion. Analysis and simulation results show that the connection establishment latency can be reduced to be 80 ms with a probability of 0.95. In [3, 10], it is suggested to use auxiliary devices, such as IrDA interfaces or RFID transponders, to facilitate connection setup. In [8], a cooperative device discovery scheme is proposed to allow devices to exchange their knowledge of nearby devices, such as BD addresses and clocks, to speed up device discovery. The recent Bluetooth specification V1.2 also proposes a mechanism which requires a device to perform inquiry scan with interlaced hopping frequency in A and B trains.

3 Analyses for Bluetooth Device Discovery

In this section, we analyze the frequency-matching time of Bluetooth V1.1 and V1.2, which is the major component of delay in its device discovery. We start with the analysis for Bluetooth V1.1. Suppose that there is already a master device performing the scan procedure. According to whether or not the master is sending ID packets, we divide the time axis into *inquiry windows* and *non-inquiry windows*. Now suppose that there is a slave device tuning to the inquiry scan procedure and starting with an inquiry scan window. We are interested in the frequency-matching delay, denoted by D, measured by the elapsed time from the time when the slave starts inquiry scan to the time when it successfully receives an ID packet from the master.

By investigating the timing diagram of Fig. 1, the slave may start its inquiry scan in an inquiry window with probability $\frac{T_{w\_inquiry}}{T_{inquiry}}$, and in a non-inquiry window with probability $\frac{T_{inquiry}-T_{w\_inquiry}}{T_{inquiry}}$. So we have

$$D = \frac{T_{w\_inquiry}}{T_{inquiry}} \times X + \frac{T_{inquiry} - T_{w\_inquiry}}{T_{inquiry}} \times (\frac{T_{inquiry} - T_{w\_inquiry}}{2} + \frac{T_{inquiryscan}}{2} + Y), \quad (1)$$

where X is the expected delay after the slave starts its inquiry scan and Y is the expected delay after the slave's first inquiry scan encounters the master's first inquiry window. Note that in the second case, the slave has to wait

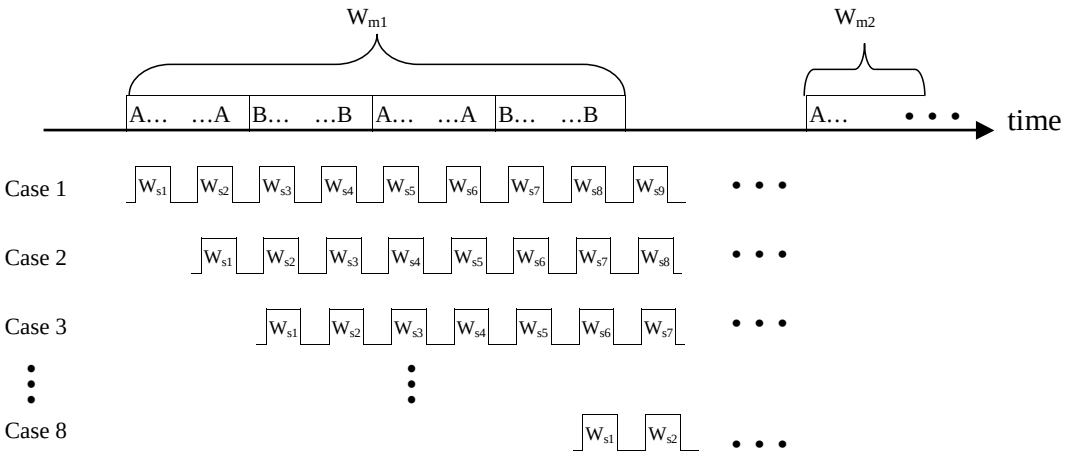


Figure 3. Eight possible cases for slave to start its inquiry scan. W_{mi} is the i-th inquiry window of the master, and W_{si} is the i-th inquiry scan window of the slave.

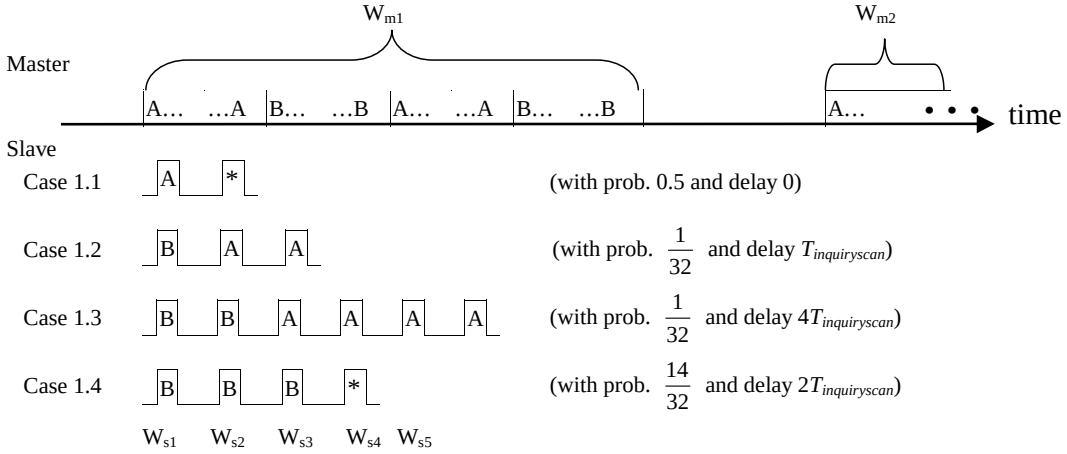


Figure 4. Illustration of Eq. (2), which contains four subcases of case 1 for frequency-matching between a master and a slave. A "*" means a "Don't Care" frequency, because a matching has already appeared in the previous inquiry scan window.

$\frac{T_{inquiry}-T_{w\_inquiry}}{2} + \frac{T_{inquiryscan}}{2}$ time in average before its first inquiry scan window encounters the master's first inquiry window.

In the following analysis, we follow the recommended values of Bluetooth that the length of one inquiry scan interval is one half of a sequence of 256 A/B trains. Therefore, the slave has two chances to match with the frequencies on which the master sends ID packets. Now, to calculate the expected value of X, we have to consider all possible locations where the first inquiry scan window of the slave (denoted by W_{s1}) appears in the first inquiry window of the master (denoted by W_{m1}). Basically, we evenly divide the window W_{m1} into 8 partitions, as illustrated in Fig. 3. There are 8 cases to consider, which are discussed in the following.

Case 1: (W_{s1} in the first $\frac{1}{8}$ window of W_{m1}) In this case, the delay will depend on the frequencies on which the slave is waiting for the master's ID packets. Recall that the slave will repeatedly scan all frequencies of train A in 16 consecutive inquiry scan windows, followed by all frequencies of train B in 16 consecutive inuiry scan windows. So there are 32 possibilities where the slave can catch an ID packet on the right frequency from the master. These

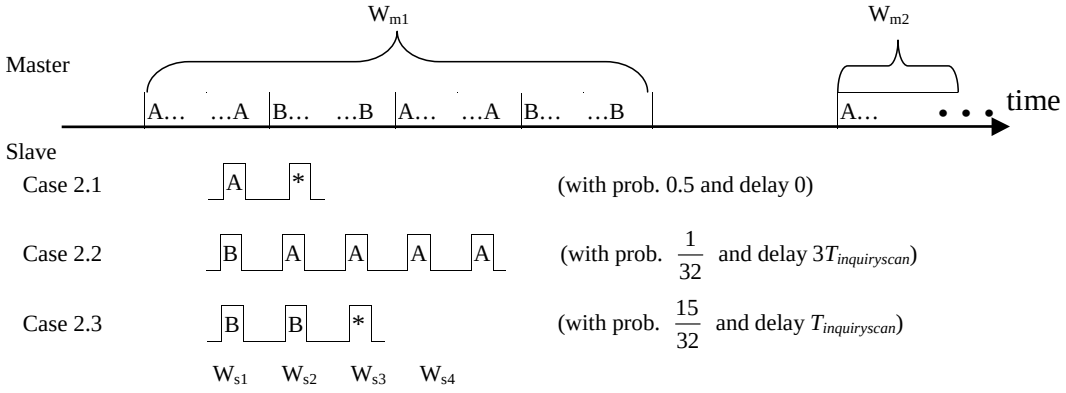


Figure 5. Illustration of Eq. (3), which contains three subcases of case 2 for frequency-matching between a master and a slave.

possibilities can be classified into 4 subcases, as illustrated in Fig. 4. So the expected value of X in this case can be approximated by

$$X_1 = \frac{1}{32} \times T_{inquiryscan} + \frac{1}{32} \times (4 \times T_{inquiryscan}) + \frac{14}{32} \times (2 \times T_{inquiryscan}). \quad (2)$$

Note that there is no delay for case 1.1 in Fig. 4. For case 1.2, the delay is one inquiry scan interval, as reflected in the first term of Eq. (2). Similarly, there are four and two inquiry scan intervals of delays for cases 1.3 and 1.4, respectively.

Case 2: (W_{s1} in the second $\frac{1}{8}$ window of W_{m1}) As described in case 1, the slave hops on 32 frequencies repeatedly. Similarly, there are also 32 possibilities where the slave can catch an ID packet on the right frequency from the master. These possibilities can be classified into 3 subcases, as illustrated in Fig. 5 So the expected value of X in this case can be approximated by

$$X_2 = \frac{1}{32} \times (3 \times T_{inquiryscan}) + \frac{15}{32} \times T_{inquiryscan}. \quad (3)$$

Note that there is no delay for case 2.1 in Fig. 5. For cases 2.2 and 2.3, the delays are three and one inquiry scan interval, respectively.

The next two cases are similar to the above two cases. So we omit the explanations.

Case 3: (W_{s1} in the third $\frac{1}{8}$ window of W_{m1})

$$X_3 = \frac{1}{32} \times T_{inquiryscan} + \frac{1}{32} \times (4 \times T_{inquiryscan}) + \frac{14}{32} \times (2 \times T_{inquiryscan}). \quad (4)$$

Case 4: (W_{s1} in the fourth $\frac{1}{8}$ window of W_{m1})

$$X_4 = \frac{1}{32} \times (3 \times T_{inquiryscan}) + \frac{15}{32} \times T_{inquiryscan}. \quad (5)$$

Case 5: (W_{s1} in the fifth $\frac{1}{8}$ window of W_{m1}) The 32 frequency-matching possibilities of case 5 can be classified into four subcases, as shown in Fig. 6. All subcases are similar to earlier discussions, except subcase 5.3, where the frequency-matching will occur in next inquiry window W_{m2}. The slave thus has to wait $\lceil \frac{T_{inquiry} - T_{inquiryscan} \times 4}{T_{inquiryscan}} \rceil \times T_{inquiryscan}$ for window W_{m2} to appear. In the following analysis, we assume that $T_{inquiry}$ is a multiple of

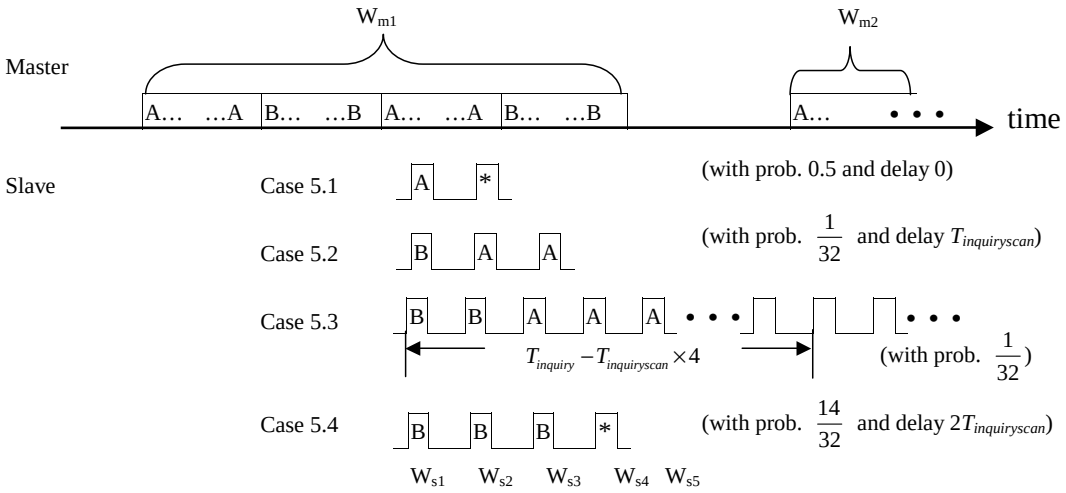


Figure 6. Illustration of Eq. (6), which contains four subcases of case 5. The frequency-matching of case 5.3 will occur in window W_{m2}

$T_{inquiryscan}$ for simplicity. So the waiting time is simplified to be $(T_{inquiry} - T_{inquiryscan} \times 4)$. After the waiting, it will take X_1 time more for frequency-matching. So the expected value of X in this case can be approximated by

$$X_5 = \frac{1}{32} \times T_{inquiryscan} + \frac{1}{32} \times (T_{inquiry} - 4 \times T_{inquiryscan} + X_1) + \frac{14}{32} \times (2 \times T_{inquiryscan}). \quad (6)$$

The next three cases are similar to case 5. So we omit the explanations.

Case 6: (W_{s1} in the sixth $\frac{1}{8}$ window of W_{m1})

$$X_6 = \frac{1}{32} \times (T_{inquiry} - 5 \times T_{inquiryscan} + X_1) + \frac{15}{32} \times T_{inquiryscan}. \quad (7)$$

Case 7: (W_{s1} in the seventh $\frac{1}{8}$ window of W_{m1})

$$X_7 = \frac{1}{32} \times T_{inquiryscan} + \frac{15}{32} \times (T_{inquiry} - 6 \times T_{inquiryscan} + X_1). \quad (8)$$

Case 8: (W_{s1} in the eighth $\frac{1}{8}$ window of W_{m1})

$$X_8 = \frac{1}{2}(T_{inquiry} - 7 \times T_{inquiryscan} + X_1). \quad (9)$$

We can now get the expected value of X as follows:

$$X = \frac{1}{8} \sum_{i=1}^{8} X_i. \quad (10)$$

Next, we derive the value of Y. It is not hard to see that the calculation is similar to the case 1 of X. Therefore, the expected value of Y is

$$Y = \frac{1}{32} \times T_{inquiryscan} + \frac{1}{32} \times (4 \times T_{inquiryscan}) + \frac{14}{32} \times (2 \times T_{inquiryscan}). \quad (11)$$

Below, we analyze the frequency-matching delay for the interlaced inquiry scan which is proposed in Bluetooth V1.2. Bluetooth V1.2 tries to interlace the inquiry scan hopping sequence of V1.1. Specifically, let $f_0, f_1, ..., f_{31}$

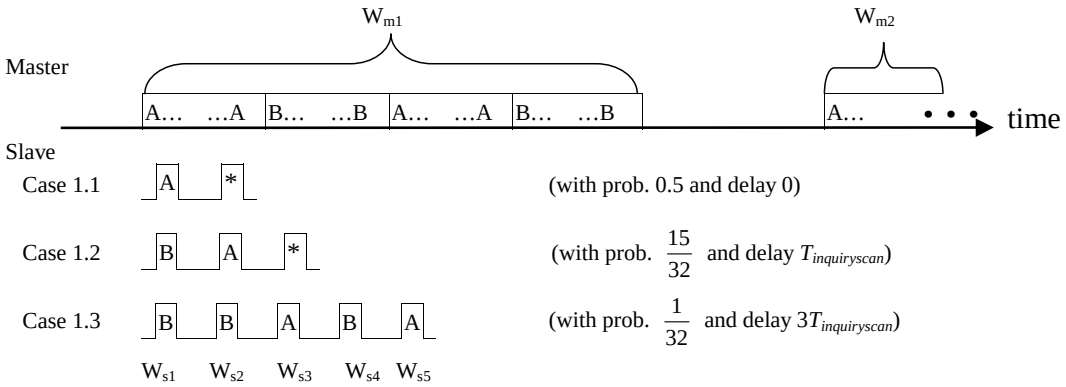


Figure 7. Three subcases of case 1 for Bluetooth V1.2.

be the hopping sequence in V1.1. Then the hopping sequence in V1.2 will replace f_i for each odd *i* by $f'_i = f_{i+16(mod\ 32)}$. Therefore, the Eq. (1) can still be applied to Bluetooth V1.2. We only need to recalculate the values of X and Y.

There are also 8 cases for analyzing X, as discussed below.

Case 1: (W_{s1} in the first $\frac{1}{8}$ window of W_{m1}) The analysis is similar to the case 1 of X in Bluetooth V1.1. There are also 32 possibilities where the slave can catch an ID packet on the right frequency from the master. These possibilities can be classified into 4 subcases, as illustrated in Fig. 7. Note that in Fig. 7, a frequency in $f_0, f_1, ..., f_{15}$ is denoted by an "A", and a frequency in $f_{16}, f_{17}, ..., f_{31}$ is denoted by a "B". Also note that case 3.1 happens when frequencies $f_{31}, f_{16}, f_1, f_{18}$ and f_3 appear in windows $W_{s1}, W_{s2}, W_{s3}, W_{s4}$ and W_{s5}, respectively. So the expected value of X in this case can be approximated by

$$X_1 = \frac{15}{32} \times T_{inquiryscan} + \frac{1}{32} \times (3 \times T_{inquiryscan}). \tag{12}$$

Case 2: (W_{s1} in the second $\frac{1}{8}$ window of W_{m1}) This case is shown in Fig. 8.

$$X_2 = \frac{1}{32} \times T_{inquiryscan} + \frac{14}{32} \times (2 \times T_{inquiryscan}) + \frac{1}{32} \times (4 \times T_{inquiryscan}). \tag{13}$$

Case 3: (W_{s1} in the third $\frac{1}{8}$ window of W_{m1}) This case is similar to case 1.

$$X_3 = \frac{15}{32} \times T_{inquiryscan} + \frac{1}{32} \times (3 \times T_{inquiryscan}). \tag{14}$$

Case 4: (W_{s1} in the fourth $\frac{1}{8}$ window of W_{m1}) This case is similar to case 2.

$$X_4 = \frac{1}{32} \times T_{inquiryscan} + \frac{14}{32} \times (2 \times T_{inquiryscan}) + \frac{1}{32} \times (4 \times T_{inquiryscan}). \tag{15}$$

Case 5: (W_{s1} in the fifth $\frac{1}{8}$ window of W_{m1}) This case is similar to case 1.

$$X_5 = \frac{15}{32} \times T_{inquiryscan} + \frac{1}{32} \times (3 \times T_{inquiryscan}). \tag{16}$$

Case 6: (W_{s1} in the sixth $\frac{1}{8}$ window of W_{m1}) This case is similar to case 5 in V1.1. The slave may need to wait ($T_{inquiry} - T_{inquiryscan} \times 5$) for window W_{m2} to appear.

$$X_6 = \frac{1}{32} \times T_{inquiryscan} + \frac{14}{32} \times (2 \times T_{inquiryscan}) + \frac{1}{32} \times (T_{inquiry} - 5 \times T_{inquiryscan} + X_1). \tag{17}$$

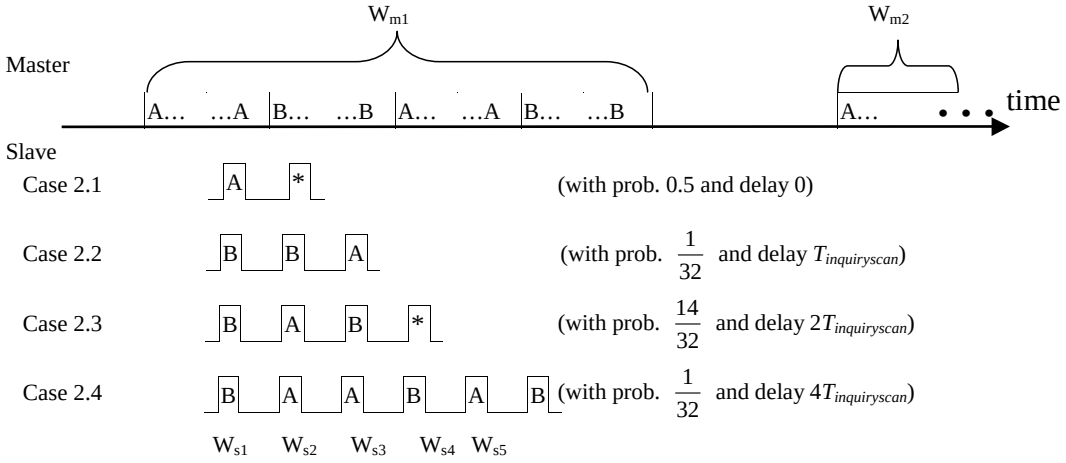


Figure 8. Four subcases of case 2 for Bluetooth V1.2.

The next two cases are similar to the above cases. So we omit the details.

Case 7: (W_{s1} in the seventh $\frac{1}{8}$ window of W_{m1})

$$X_7 = \frac{15}{32} \times T_{inquiryscan} + \frac{1}{32} \times (T_{inquiry} - 6 \times T_{inquiryscan} + X_1). \tag{18}$$

Case 8: (W_{s1} in the eighth $\frac{1}{8}$ window of W_{m1})

$$X_8 = \frac{1}{2} \times (T_{inquiry} - 7 \times T_{inquiryscan} + X_1). \tag{19}$$

We can now get the expected value of X as follows:

$$X = \frac{1}{8} \sum_{i=1}^{8} X_i. \tag{20}$$

Next, we want to calculate Y. It is not hard to see that the calculation is similar to the case 1 of X. Therefore, the expected value of Y is

$$Y = \frac{15}{32} \times T_{inquiryscan} + \frac{1}{32} \times (3 \times T_{inquiryscan}). \tag{21}$$

If we set $T_{inquiry} = 60$ and $T_{w\_inquiry} = 10.24$ seconds according to Table 1, we get the frequency-matching time $D = 23.55$ and 22.53 for Bluetooth V1.1 and V1.2, respectively. If we look in further details, we find that $X = 7.58$ and 4.47 and $Y = 1.32$ and 0.72 for V1.1 and V1.2, respectively. The interlaced inquiry scan indeed speeds up the frequency-matching but overall the improvement does not seem to be significant. The reason is because the value of $T_{inquiry}$ is too large. Thus, in Section 4 we propose some methods to speed up the bluetooth device discovery.

4 Speedup Schemes for Bluetooth Device Discovery

In this section, we propose three methods for speeding up the Bluetooth device discovery.

4.1 Half Inquiry Interval (HII)

From the analysis in Section 3, especially in Eq. (1), we note that the frequency-matching time is dominated by $T_{inquiry}$. Thus, we recommend that $T_{inquiry}$ be halved. In order to keep the same ratio of inquiry time, we

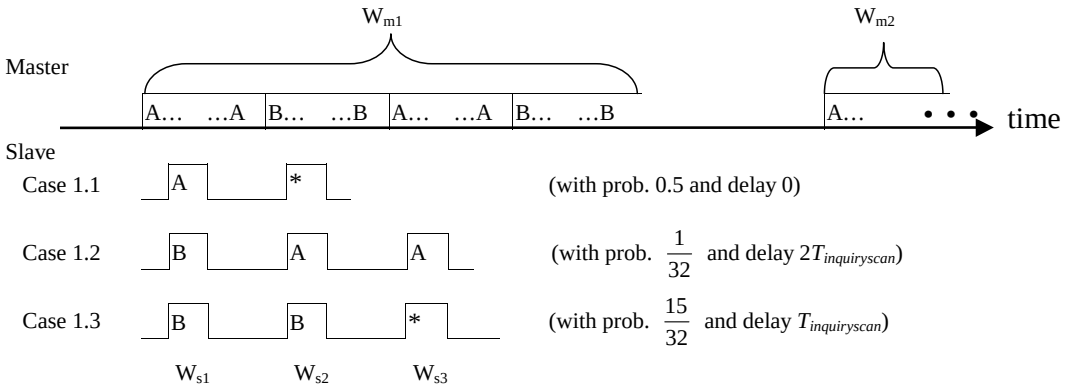


Figure 9. The case 1 of the HII method.

also recommend that $T_{w\_inquiry}$ be halved. As a result of this, the slave has only one chance to match with the frequencies on which the master sends ID packets during a sequence of 256 A/B trains. Note that here we do not use the interlacing technique in V1.2.

Below, we analyze the new frequency-matching time due to these changes. Eq. (1) is still applicable. However, there are only four cases of X, as discussed below.

Case 1: (W_{s1} in the first $\frac{1}{4}$ window of W_{m1}) There are 32 possibilities, which can be classified into 3 subcases as illustrated in Fig. 9. Note that there is only one chance for frequency matching during a sequence of 256 A/B trains. The delay is:

$$X_1 = \frac{1}{32} \times (2 \times T_{inquiryscan}) + \frac{15}{32} \times T_{inquiryscan}. \tag{22}$$

Case 2: (W_{s1} in the second $\frac{1}{4}$ window of W_{m1}) This case is similar to case 1.

$$X_2 = \frac{1}{32} \times (2 \times T_{inquiryscan}) + \frac{15}{32} \times T_{inquiryscan}. \tag{23}$$

Case 3: (W_{s1} in the third $\frac{1}{4}$ window of W_{m1}) The 32 frequency-matching possibilities can be classified into three subcases, as shown in Fig. 10. All subcases are similar to earlier discussions, except subcase 3.2, where the frequency-matching will occur in next inquiry window W_{m2}. Recall that we assume that $T_{inquiry}$ is a multiple of $T_{inquiryscan}$, so the waiting time is ($T_{inquiry} - T_{inquiryscan} \times 2$). The expected value of X in this case can be approximated by

$$X_3 = \frac{1}{32} \times (T_{inquiry} - 2 \times T_{inquiryscan} + X_1) + \frac{15}{32} \times T_{inquiryscan}. \tag{24}$$

Case 4: (W_{s1} in the fourth $\frac{1}{4}$ window of W_{m1}) This case is similar to case 3.

$$X_4 = \frac{1}{2} \times (T_{inquiry} - 3 \times T_{inquiryscan} + X_1). \tag{25}$$

We can now get the expected value of X as follows:

$$X = \frac{1}{4} \sum_{i=1}^{4} X_i. \tag{26}$$

The calculation of Y is similar to the case 1 of X. The expected value of Y is

$$Y = \frac{1}{32} \times (2 \times T_{inquiryscan}) + \frac{15}{32} \times T_{inquiryscan}. \tag{27}$$

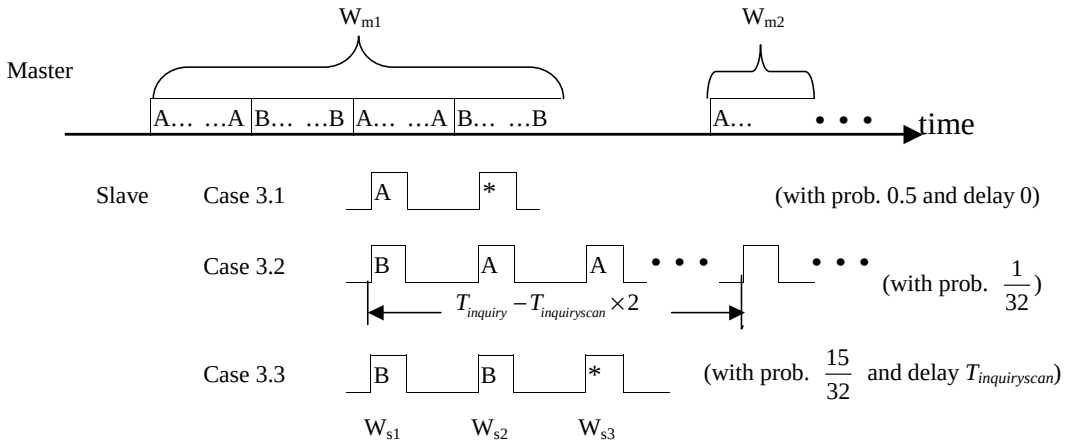
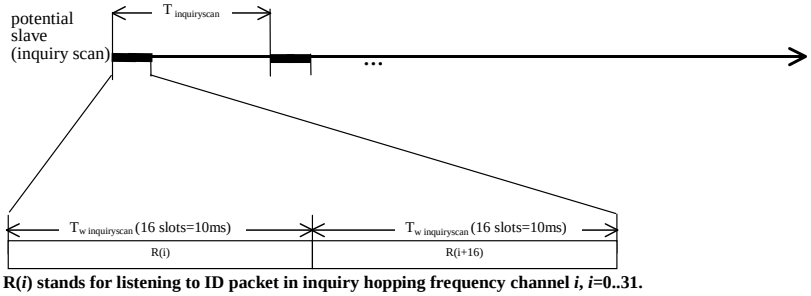


Figure 10. The case 3 of the HII method.



R(*i*) stands for listening to ID packet in inquiry hopping frequency channel *i*, *i*=0..31.

Figure 11. The proposed DIS scheme.

For example, if we set $T_{inquiry} = 30$ and $T_{w\_inquiry} = 5.12$ seconds, then we get the frequency-matching time D = 12.11 seconds. The reduction is significant. In this case, X is 4.06 seconds and Y is 0.68 seconds. So the reduction is mainly contributed by the reduction of $T_{inquiry}$.

4.2 Dual Inquiry Scan (DIS)

In this scheme, we hope that once an inquiry scan window of a slave encounters an inquiry window of a master, a frequency matching will occur as long as there is sufficient overlapping between these two windows. Toward this goal, the *Dual Inquiry Scan* (*DIS*) scheme requires the slave to perform inquiry scan on dual frequencies, one in A train and the other in B train. To be more precise, for every $T_{inquiryscan}$ period, the slave should perform inquiry scan on two frequencies, f_i and f_{i+16}, each for a duration of $T_{w\_inquiryscan}$ (refer to Fig. 11). Note that the value of i is increased by 1 (with modulo 32) after each inquiry scan window. As a result, frequency-matching will occur on either f_i or f_{i+16} with a high probability. In order to keep the same ratio of inquiry scan time, we recommend that $T_{inquiryscan}$ be doubled.

Below, we analyze the frequency-matching delay for the *DIS* scheme. Eq. (1) can also be applied to the analysis except that Y is replaced by X. That is, we have

$$D = \frac{T_{w\_inquiry}}{T_{inquiry}} \times X + \frac{T_{inquiry} - T_{w\_inquiry}}{T_{inquiry}} \times (\frac{T_{inquiry} - T_{w\_inquiry}}{2} + \frac{T_{inquiryscan}}{2} + X), \qquad (28)$$

where X is the expected delay after the slave starts an inquiry scan window during an inquiry window. When

the master is sending an A/B trains which is sufficiently covered by the slave's inquiry scan window, frequency-matching will occur with no delay with a probability of about $\frac{1}{2}$ and with $T_{train}(= 0.01)$ delay with a probability of about $\frac{1}{2}$. Thus, we have $X \approx 0.005$, which gives $D \approx 21.71$ seconds.

4.3 Combination of HII and DIS

If we combine the above two strategies by adopting *HII* for the master and adopting *DIS* for the slave, then further reduction of D can be obtained. The analysis is similar and can be obtained from Eq. (28). By setting $T_{inquiry} = 30$ and $T_{w\_inquiry} = 5.12$ seconds, D can be reduced to be 11.38 seconds.

5 Conclusions

In this paper, we have analyzed the frequency-matching time of Bluetooth V1.1 and V1.2. The main component of delay in its long device discovery is the long waiting time for the appearance of inquiry windows from the master. The proposed *HII* scheme can reduce the aforementioned waiting time. The *DIS* scheme can further reduce the frequency-matching delay by scanning two frequencies back to back. If we combine these two schemes, the expected frequency-matching delay can be reduced from 23.55 seconds to 11.38 seconds. The ratio of time for performing inquiry and inquiry scan does not increased.

6 Acknowledgement

Y. C. Tseng's research is co-sponsored by the NSC Program for Promoting Academic Excellence of Universities under grant number 93-2752-E-007-001-PAE, by Computer and Communications Research Labs., ITRI, Taiwan, by Intel Inc., by the Institute for Information Industry and MOEA, R.O.C, under the Handheld Device Embedded System Software Technology Development Project and the Communications Software Technology Project, and by Chung-Shan Institute of Science and Technology under contract number BC93B12P.

References

[1] S. Basagni, R. Bruno, and C. Petrioli. "Device Discovery in Bluetooth Networks: A Scatternet Perspective," *Proc. of the Second IFIP-TC6 Networking Conference,Networking 2002*, Pisa, Italy, May 2002, pp. 1087-1092.

[2] Bluetooth Special Interest Group. Bluetooth specification version 1.1 and 1.2. *http://www.bluetooth.com*, 2001.

[3] A. Busboom, I. Herwono, M. Schuba, and G. Zavagli. "Unambiguous Device Identification and Fast Connection Setup in Bluetooth," *Proc. of the European Wireless 2002*, vol. 0, Florence, Italy, Feb 2002.

[4] K. Cheolgi, M. Joongsoo, L. Joonwon. "A Random Inquiry Procedure using Bluetooth," *Proc. of International Conference on Communication in Computing (CIC)*, Las Vegas, USA, Jun 2001.

[5] I. Maric. "Connection Establishment in the Bluetooth System," Masters Thesis, the State University of New Jersey, 2000.

[6] P. Murphy, E. Welsh, and J. P. Frantz. "Using Bluetooth for Short-Term Ad-Hoc Connections Between Moving Vehicles: A Fesability Study," *IEEE Vehicular Technology Conference*, vol. 1, no. 55, Birmingham, AL, May 2002, pp. 414-418.

[7] T. Salonidis, P. Bhagwat, and L. Tassiulas. "Proximity awareness and fast connection establishment in Bluetooth," *Proc. of Mobile and Ad Hoc Networking and Computing, 2000 (MobiHOC'00)*, Boston, Massachusetts, Aug. 2000, pp. 141-142.

[8] F. Siegemund and M. Rohs. "Rendezvous Layer Protocols for Bluetooth-Enabled Smart Devices," *Proc. 1st International Conference on Architecture of Computing Systems*, vol. 2299, Karlsruhe, Germany, pp. 256-273, Apr. 2002.

[9] E. Welsh, P. Murphy, and J. P. Frantz. Improving Connection Times for Bluetooth Devices in Mobile Environments. *Proc. of International Conference on Fundamentals of Electronics, Communications and Computer Sciences (ICFS)*, Mar. 2002.

[10] R. Woodings, D. Joos, T. Clifton, and C. D. Knutson. "Rapid Heterogeneous Connection Establishment: Accelerating Bluetooth Inquiry Using IrDA," *Proc. of the Third Annual IEEE Wireless Communications and Networking Conference (WCNC) 2002*, vol. 1, Orlando, Florida, Mar. 2002, pp. 342-349.

Biographies

Jehn-Ruey Jiang received his Ph. D. degree in Computer Science in 1995 from National Tsing-Hua University, Taiwan. He joined Chung-Yuan Christian University and Hsuan-Chuang University as an Associate Professor in 1995 and 1998, respectively. He is currently with the Department of Computer Science and Information Engineering, National Central University. He is a recipient of the Best Paper Award in Int'l Conf. on Parallel Processing, 2003. His research interests include distributed computing, mobile computing, peer-to-peer computing, distributed fault-tolerance, protocols for mobile ad hoc networks and wireless sensor networks.

Bing-Rong Lin received his B.S. degree in Computer Science from the National Chiao-Tung University, Taiwan, in 2002. His research interests include wireless network, sensor network and Bluetooth.

Yu-Chee Tseng received his B.S. and M.S. degrees in Computer Science from the National Taiwan University and the National Tsing-Hua University in 1985 and 1987, respectively. He worked for the D-LINK Inc. as an engineer in 1990. He obtained his Ph.D. in Computer and Information Science from the Ohio State University in January of 1994. He was an Associate Professor at the Chung-Hua University (1994 1996) and at the National Central University (1996 1999), and a Full Professor at the National Central University (1999 2000). Since 2000, he has been a Full Professor at the Department of Computer Science and Information Engineering, National Chiao-Tung University, Taiwan.

He is a two-time recipient of the Outstanding Research Award, National Science Council, ROC, in 2001-2002 and 2003-2005, and a recipient of the Best Paper Award in Int'l Conf. on Parallel Processing, 2003. Several of his papers have been chosen as Selected/Distinguished Papers in international conferences. He has guided students to participate in several national programming contests and received several awards. His research interests include mobile computing, wireless communication, network security, and parallel and distributed computing. Dr. Tseng is a member of ACM and a Senior Member of IEEE.