AN ADAPTIVE SCHEDULING ALGORITHM FOR BLUETOOTH AD-HOC NETWORKS

By

Raymond Yan Lam Lee

B.A.Sc. University of British Columbia, 2001

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

In

THE FACULTY OF GRADUATE STUDIES

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

September 2004

Library Authorization

In presenting this thesis in partial fulfillment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

RAYMOND YAN LAM LEE
Name of Author *(please print)*

30/09/2004
Date (dd/mm/yyyy)

Title of Thesis: An Adaptive Scheduling Algorithm for Bluetooth Ad-Hoc Networks

Degree: Master of Applied Science Year: 2004

Department of Electrical and Computer Engineering
The University of British Columbia
Vancouver, BC Canada

Abstract

Bluetooth devices are required to form a *piconet* before exchanging data. Each piconet has a *master unit* that controls the channel access and frequency hopping sequence. Other nodes in the piconet are referred to as the *slave units*. In a piconet, the master controls the access of all devices to the channel through a time division duplex master-slave polling scheme. Several piconets can be interconnected via bridge nodes to create a *scatternet*. *Bridge nodes* are capable of time-sharing between multiple piconets, receiving packets from one piconet and forwarding them to another. A distributed scatternet scheduling algorithm is necessary: (i) to facilitate the polling operation from a master to its slaves; and (ii) to coordinate the switching of the bridge nodes between different piconets.

In this thesis, we propose an Adaptive Scheduling Algorithm (ASA) for Bluetooth scatternets. ASA is adaptive in which the bandwidth allocated on each link or session is dynamically adjusted based on the estimated traffic. Moreover, ASA integrates both intra-piconet and inter-piconet scheduling to improve the aggregate throughput and delay. In addition, ASA prevents the bridge node conflict and satisfies the max-min fairness criterion.

We compare our proposed ASA with two other scheduling algorithms via simulations. Results show that ASA can achieve the max-min fairness under different traffic conditions. Simulation results also show that under Constant Bit Rate (CBR) or bursty on-off User Datagram Protocol (UDP) traffic, ASA can maintain a high aggregate throughput and low delay. In addition, under Transmission Control Protocol (TCP) traffic, results show that ASA can achieve a small average transfer delay for different file sizes.

Table of Contents

Abstract ii

List of Tables vii

List of Figures viii

List of Figures viii

Acknowledgements xi

Chapter 1 - Introduction 1

1.1 Bluetooth Background 1

1.1.1 Bluetooth Links 3

1.1.2 Bluetooth Modes 4

1.2 Bluetooth Scatternet Scheduling 5

1.2.1 Intra-piconet Scheduling Issues 5

1.2.2 Inter-piconet Scheduling Issues 6

1.3 Max-Min Fairness Criterion 7

1.4 Motivations and Objectives 9

1.5 Contributions of the Thesis 11

1.6 Organization of the Thesis 11

Chapter 2 - Related Work 13

2.1 Piconet Scheduling 13

2.1.1 Pure Round Robin (PRR) 13

2.1.2 Exhaustive Round Robin (ERR) 13

2.1.3 K-limited Round Robin (K-limited RR) 14

2.1.4 Limited and Weighted Round Robin (LWRR) 14
2.1.5 Deficit Round Robin (DRR) 15
2.1.6 Look Ahead Round Robin (LARR) 16
2.1.7 K-Look-Ahead Round Robin (K-Look-Ahead RR) 17
2.1.8 Adaptive Flow-based Polling (AFP) 17
2.1.9 Sticky Adaptive Flow-based Polling (Sticky AFP) 18
2.1.10 HOL Priority Policy (HOL-PP) 19
2.1.11 HOL K-Fairness Policy (HOL-KFP) 20
2.2 Scatternet Scheduling 21
2.2.1 Deterministic-based Scatternet Scheduling Algorithms 21
2.2.1.1 Rendezvous Scheduling 21
2.2.1.2 Flexible Scatternet-wide Scheduling Algorithm (FSS) 23
2.2.1.3 Jump Mode 26
2.2.1.4 Load Adaptive Algorithm (LAA) 29
2.2.1.5 A Fair and Traffic Dependent Scheduling Algorithm 30
2.2.1.6 A Locally Coordinated Scatternet Scheduling Algorithm (LCS) 34
2.2.2 Individual Node Based Scatternet Scheduling Algorithms 37
2.2.2.1 Credit Based Scheduling (CBS) 37
2.2.2.2 Pseudo-Random Coordinated Scatternet Scheduling (PCSS) 39
2.3 Summary 41
Chapter 3 – Adaptive Scheduling Algorithm 43
3.1 Assumptions 43
3.2 Adaptive Scheduling Algorithm 44

3.2.1 Traffic Estimation 46
3.2.2 Intra-piconet Scheduling 50
3.2.3 Inter-piconet Scheduling 53
3.2.4 Resume Send 56
3.3 Fairness Discussion 58
3.3.1 Fairness around the Master Node 58
3.4 Example 60
3.5 Summary 63
Chapter 4 – Performance Evaluation 65
4.1 Fairness Comparison 66
4.1.1 Balance Traffic Load 67
4.1.2 Unbalance Traffic Load around a Master 70
4.1.3 Unbalance Traffic Load around a Bridge 74
4.2 Comparison with UDP Traffic 78
4.2.1 CBR Scenario 78
4.2.2 Bursty On-Off Traffic Scenario 84
4.3 Comparison with TCP Traffic 88
4.4 Repeated Experiments 92
4.4.1 Single Bridge Node in Two Piconets 92
4.4.2 Different Number of Slaves 93
4.4.3 Single Bridge Node Between Three Piconets 94
4.4.4 Piconet with Two Bridge Nodes 95
4.5 Summary 97

Chapter 5 – Conclusions 98

5.1 Summary 98

5.2 Future Work 99

Bibliography 101

Glossary of Acronyms 104

List of Tables

Table 2.1. Procedure adopted by the master if slots available for the RP is less than 10 33

Table 4.1. Traffic pattern with fixed rate.....68

Table 4.2. Traffic pattern for testing bandwidth allocation around master 71

Table 4.3. Traffic pattern for testing bandwidth allocation around bridge.....75

Table 4.4. Traffic pattern for M-S and M-B-M traffic generating 80

Table 4.5. Traffic pattern for S-M-S and S-M-B-M-S traffic 82

Table 4.6. Traffic Pattern for M-S and M-B-M.....88

Table 4.7. Traffic Pattern for S-M-S and S-M-B-M-S 90

List of Figures

Figure 1.1. Piconet 2

Figure 1.2. Scatternet 2

Figure 1.3 An eSCO link for single-slot packet 3

Figure 1.4. SCO and ACL connections 4

Figure 1.5. SNIFF mode 5

Figure 1.6. Switching of bridge node between two piconets 6

Figure 1.7. Topology for testing max-min fairness 7

Figure 2.1. Rendezvous scheduling 22

Figure 2.2. Switch-table for a bridge node connected with 3 masters 23

Figure 2.3. Pseudo random sequence of RWs 27

Figure 2.4. Pseudocode of LAA 29

Figure 2.5. Algorithm for updating the estimation of N and r on slaves 31

Figure 3.1. Structure of a switch schedule 46

Figure 3.2. Algorithm for traffic rate estimation 47

Figure 3.3. Relation between MUSS and Trigger Value 48

Figure 3.4. Algorithm for trigger point estimation 49

Figure 3.5. Algorithm for updating active and waiting list 52

Figure 3.6. The algorithm for obtaining master suggested meeting time 54

Figure 3.7. The algorithm for obtaining bridge confirmed meeting time 56

Figure 3.8. The algorithm for updating node status 58

Figure 3.9. Scatternet topology and switch schedule 61

Figure 4.1. Topology of a scatternet 66

Figure 4.2. Fairness comparison for ASA in general 69

Figure 4.3. Fairness comparison for FSS in general 69

Figure 4.4. Fairness comparison for CBS in general 70

Figure 4.5. Fairness comparison for ASA on master 72

Figure 4.6. Fairness comparison for FSS on master 72

Figure 4.7. Fairness comparison for CBS on master 73

Figure 4.8. Fairness comparison for ASA on bridge 76

Figure 4.9. Fairness comparison for FSS on bridge 76

Figure 4.10. Fairness comparison for CBS on bridge 77

Figure 4.11. Aggregate throughput for M-S and M-B-M traffic 81

Figure 4.12. Average delay for M-S and M-B-M traffic 81

Figure 4.13. Aggregate throughput for S-M-S and S-M-B-M-S traffic 83

Figure 4.14. Average Delay for S-M-S and S-M-B-M-S traffic 83

Figure 4.15. Aggregate throughput with ON:OFF ratio of 1:1 85

Figure 4.16. Average delay with ON:OFF ratio of 1:1 85

Figure 4.17. Aggregate throughput with ON:OFF ratio of 1:2 86

Figure 4.18. Average delay with ON:OFF ratio of 1:2 86

Figure 4.19. Aggregate throughput with ON:OFF ratio of 2:1 87

Figure 4.20. Average delay with ON:OFF ratio of 2:1 87

Figure 4.21. TCP Traffic between M-S 89

Figure 4.22. TCP Traffic between M-S-M 89

Figure 4.23. TCP Traffic between S-M-S 91

Figure 4.24. TCP Traffic between S-M-B-M-S 91

Figure 4.25. Sharing of bandwidth within a piconet 92

Figure 4.26. Sharing of bandwidth between bridge nodes and slave node 93

Figure 4.27. Sharing of bandwidth between bridge node and slave node in piconet II 94

Figure 4.28. Sharing of bandwidth for all M-B links 95

Figure 4.29. Sharing of bandwidth between B1 and B2 in piconet II 96

Acknowledgements

I would like to express my appreciation to my supervisor Dr. Vincent Wong, for his guidance and suggestions throughout the course of this thesis. This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) under grant number STPGP 257684-02.

Finally, my thanks go to my family and friends, especially Joey Mak, Manman and Paupau, for their encouragement and support throughout my graduate studies.

Chapter 1 - Introduction

Bluetooth [1] is a low power, low complexity, and low cost short-range wireless communications technology for electronic devices. Bluetooth operates in the 2.4 GHz unlicensed Industrial Scientific Medical (ISM) band. The ISM band is divided into 79 frequency sub-bands. A *channel* in Bluetooth is defined by a pseudo-random frequency hopping (FH) sequence hopping through the 79 frequency bands at a rate of 1600 hops per second. The result is a slotted channel with the slot duration equal to 625 μs.

The Bluetooth specification has defined various packet types for both data and control packets. Data packets with Forward Error Correction (FEC) capability and occupy either one, three, or five time-slots are called Data-Medium Rate 1 (*DM*1), Data-Medium Rate 3 (*DM*3), and Data-Medium Rate 5 (*DM*5) packets, respectively. These DM packets use the (15, 10) shortened Hamming code for error correction. On the other hand, data packets without FEC capability and occupy either one, three, or five time-slots are called Data-High Rate 1 (*DH*1), Data-High Rate 3 (*DH*3), and Data-High Rate 5 (*DH*5) packets, respectively.

1.1 Bluetooth Background

Each Bluetooth device has a unique 48-bit Bluetooth device address (BD_ADDR). Bluetooth devices are required to form a *piconet* before exchanging data. Each piconet has a *master unit* that controls the channel access and frequency hopping sequence. Other nodes in the piconet are referred to as the *slave units*. Within a piconet, there is one master and up to seven active slaves, and all devices share the same wireless channel. The master will assign 3-bit Logical Transport Address (LT_ADDR) to each of the active slave. In order to

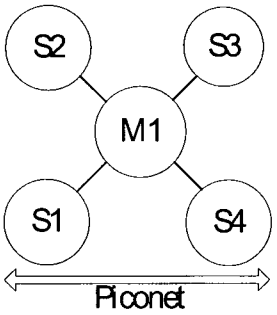

Figure 1.1. Piconet

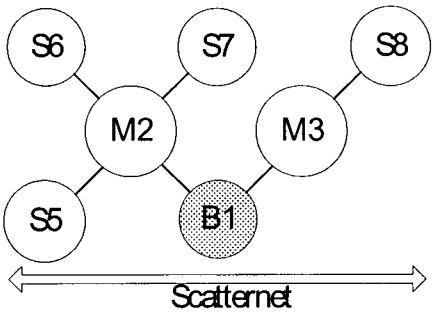

Figure 1.2. Scatternet

minimize the interference between different piconets, the frequency hopping sequence is unique for each piconet.

In a piconet, the master controls the access of all devices to the channel through a Time Division Duplex (TDD) master-slave polling scheme. For communication between a master and its slave, the master first sends a data packet to the slave in even-numbered slots as a polling message. If the master queue for the slave is empty, it can instead send a POLL packet as a polling message. When the slave receives the polling message, it immediately replies with a data packet to the master. If the slave queue is empty, it can instead send a NULL packet as a reply message.

Several piconets can be interconnected via bridge nodes to create a *scatternet*. *Bridge nodes* are capable of time-sharing between multiple piconets, receiving packets from one piconet and forwarding them to another. A bridge node can be a master in one piconet and act as slave in other piconets. This is called a *master-slave bridge*. Alternatively, a bridge node can act as a slave in all the piconets it is connected to. This is called a *slave-slave bridge*. Figure 1.1 shows a piconet in which master node M1 is connected to slave nodes S1, S2, S3 and S4. Figure 1.2 shows a scatternet in which both master nodes M2 and M3 are connected to bridge node B1. In addition, master node M2 is connected to slave nodes S5, S6

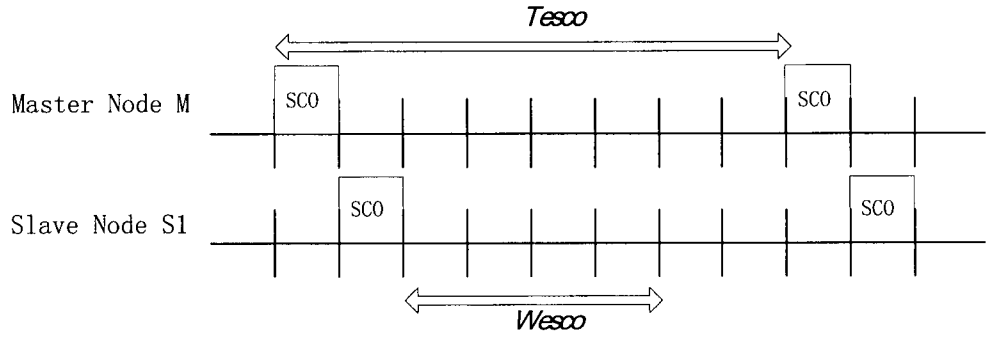

Figure 1.3 An eSCO link for single-slot packet

and S7. Master node M3 is connected to slave node S8.

1.1.1 Bluetooth Links

Bluetooth supports different traffic types between two Bluetooth devices. For voice traffic, the master maintains a Synchronous Connection-Oriented (SCO) link to a specific slave by reserving time-slots at regular T_{SCO} intervals. In the Bluetooth specification, the T_{SCO} interval can be equal to either two, four, or six time-slots. A SCO link does not support packet retransmission.

On the other hand, the Extended Synchronous Connection-Oriented (eSCO) link supports limited retransmission of packets. The master maintains an eSCO link to a specific slave by using reserved slots at regular T_{eSCO} intervals. The T_{eSCO} interval is negotiated between the master and the slave during link setup. The range of the T_{eSCO} interval is between 4 and 256 slots. Besides, the master also maintains an additional W_{eSCO} slots after the reserved slots as a retransmission window. The range for W_{eSCO} is from 0 to 256 slots. Figure 1.3 shows an eSCO link for single-slot packets between master M and slave node S1.

For data traffic, the master can establish an Asynchronous Connection-oriented Link (ACL) to any slaves in TDD slots not reserved for SCO links. Therefore, if a SCO link is

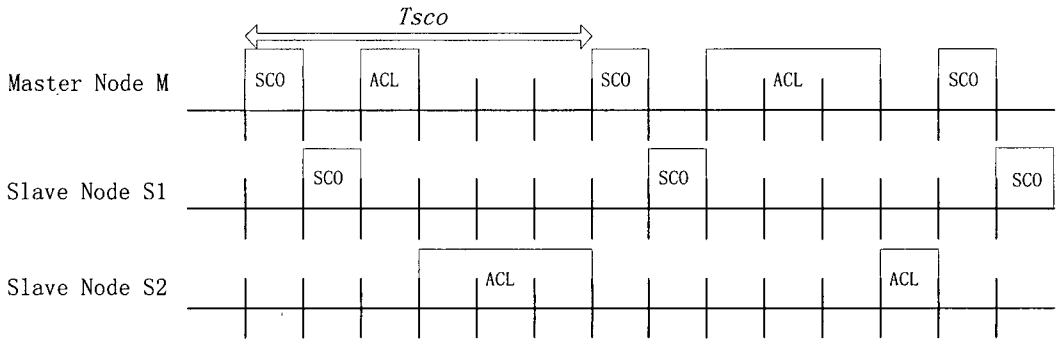

Figure 1.4. SCO and ACL connections

present, there will be either two or four time-slots available for all the ACL connections. As a result, data traffic using ACL is a best-effort service. Figure 1.4 shows the slots used by a SCO link and the slots available for an ACL connection.

1.1.2 Bluetooth Modes

There are four modes defined in the Bluetooth specification, namely: *active*, *sniff*, *hold*, and *park* modes. Nodes in *active mode* listen to their master all the time. On the other hand, nodes in *sniff mode* wake up at each sniff slot to start communicating with their master. The interval between two successive sniff slots is called T_{sniff}. Beginning from the sniff slot, the sniff node starts listening to the master for $N_{sniff\_attempt}$ slots until a packet with matching LT_ADDR with the sniff node is received. With each received matching LT_ADDR packet, the sniff node will continue to listen for $N_{sniff\_timeout}$ slots or the remaining of $N_{sniff\_attempt}$ slots whichever is larger. Once the sniff node has finished listening to the master, it will go back to the sleep state and wait for the next sniff slot. Figure 1.5 shows the sniff mode model.

For *hold mode*, the master first negotiates the holding period with the slave node. The slave node will then switch into the hold mode. During the holding period, the node is not required to listen to the master. When the timer of the holding period expires, the hold node

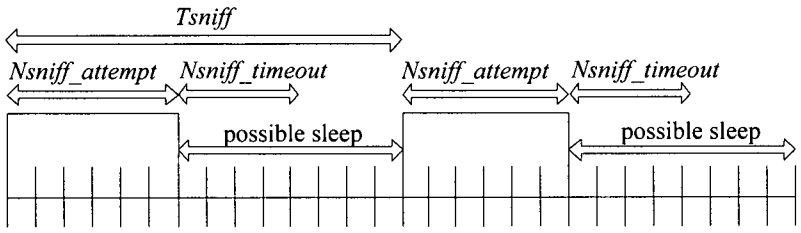

Figure 1.5. SNIFF mode

switches to the active node.

Nodes in *park mode* do not take part in the piconet and they have to give up their LT_ADDR before entering park mode. The parked node will then wake up at regular interval.

The Bluetooth specification [1] recommends that the masters use either the hold mode or sniff mode to allow a bridge node to switch between different piconets.

1.2 Bluetooth Scatternet Scheduling

Scheduling in Bluetooth scatternets can be divided into two tasks, namely: *intra-piconet scheduling* and *inter-piconet scheduling*. Intra-piconet scheduling focuses on the scheduling of packets transmission within a piconet. Since a slave node cannot transmit a packet without first being polled by its master, intra-piconet scheduling is controlled by the master. On the other hand, inter-piconet scheduling focuses on when a bridge node can switch between different piconets and how a bridge node communicates with the masters in different piconets. In both types of scheduling, there exist some constraints that must be considered when designing an efficient scheduling algorithm.

1.2.1 Intra-piconet Scheduling Issues

An efficient intra-piconet scheduling algorithm should *minimize the number of wasted slots*. Since each master-slave polling operation includes the transmission from the master to its

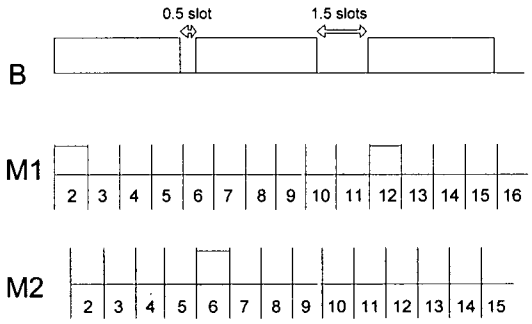

Figure 1.6. Switching of bridge node between two piconets

slave and a reply from the slave to its master, an empty queue in either side will result in a wasted slot. Moreover, since each node may generate traffic in different rate, the bandwidth requirement on each link may be different. Therefore, for efficient bandwidth utilization, the polling period for each slave may be different. In addition, it is also required to maintain bandwidth fairness among different slaves within the piconet.

1.2.2 Inter-piconet Scheduling Issues

The first issue on inter-piconet scheduling is the switching of bridge node between piconets. Because each Bluetooth device only has one transceiver, a bridge node can only participate in one piconet at a time. Therefore, a bridge node has to switch between connected piconets in order to transmit data from one piconet to another. As each master uses its own clock, a bridge node has to re-synchronize with the new master when it switches to a new piconet. The switch between two piconets may result as a maximum of two slots lost. As shown in Figure 1.6, a slave-slave bridge node first synchronizes with master M1. When the bridge node switches to master M2, it has to wait for the polling message from M2 at the even-numbered slot. Therefore, there will be a gap of half a slot. Later, when the bridge node switches back to M1, it has to wait for the polling message at even-numbered slot again. This

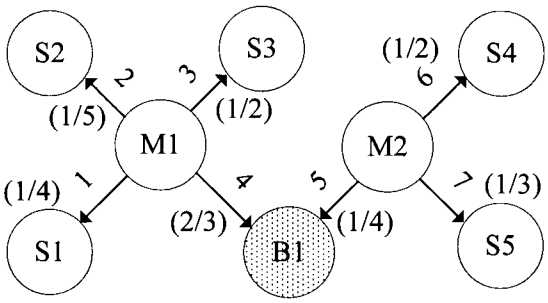

Figure 1.7. Topology for testing max-min fairness

time, there will be a gap of one and a half slots. As a result, there will be a maximum of two slots lost.

Besides the issue on slot wastage, another issue arises when two masters try to access the bridge node simultaneously. This is referred to as the *bridge node conflict*. Since a bridge node can only listen to one master at a time, the other master will not be able to communicate with the bridge node and will waste slots for the polling operation. Lastly, an efficient inter-piconet scheduling scheme also needs to dynamically allocate bandwidth on each link and maintain fairness among all nodes.

1.3 Max-Min Fairness Criterion

Definition 1: Let L be the set contains all links in a network, and let C be the total capacity of L. A rate allocation is *feasible* if the sum of rate on all links in L is smaller than or equal to C.

Definition 2: A rate allocation is *max-min fair* if it is impossible to increase the rate (r_i) on a link without decreasing the rate (r_j) on another link with $r_j \leq r_i$ ($i, j \in L$) and it is feasible.

According to the definition of max-min fairness [15], a bandwidth allocation on links is referred to as *max-min fair* if the bandwidth allocated to a link cannot be improved without

decreasing that of any other links which demand an equal or less bandwidth. The fundamental idea of achieving max-min fairness is to first allocated equal bandwidth to each link around a node. If the link cannot utilize the assigned bandwidth since it only demands for a lower bandwidth, the residual bandwidth will be equally distributed to other links.

We use the topology in Figure 1.7 to show how to allocate bandwidth for each link in order to achieve the max-min fairness. In the topology, the traffic rates for links 1 to 7 are (1/4, 1/5, 1/2, 2/3, 1/4, 1/2, 1/3), respectively. Since master node M1 is connected with four nodes, it will assign 1/4 of bandwidth to each of the connected links. On the other hand, since master node M2 is connected with three nodes, it will assign 1/3 of bandwidth to each of the connected links. Lastly, since bridge node B1 is connected with two nodes, it will assign 1/2 of bandwidth to each of the connected links.

We first consider the bandwidth allocation around M1. Since the traffic rate on link 1 is 1/4, which is equal to the assigned bandwidth, link 1 can fully utilize the assigned bandwidth. However, since the traffic rate on link 2 is 1/5, which is less than the assigned bandwidth, the bandwidth assignment on link 2 will become 1/5. Thus, the remaining bandwidth ($1/4 - 1/5 = 1/20$) should be reallocated to other links. As link 1 cannot utilize any extra bandwidth, the remaining bandwidth will be equally distributed to link 3 and link 4. Since link 3 generates traffic at a rate of 1/2, it can utilize the extra bandwidth. Therefore, the new bandwidth allocation on link 3 is $1/4 + (1/20 \times 1/2) = 11/40$. In order to determine whether link 4 can utilize the remaining extra bandwidth ($1/20 \times 1/2 = 1/40$), we have to determine what amount of bandwidth is assigned to link 4 from B1.

However, we need to check the bandwidth allocation around M2 in order to determine the bandwidth allocation around B1. Since the traffic rate on link 7 is 1/3, link 7

can fully utilize the assigned bandwidth. However, since the traffic rate on link 5 is 1/4, the bandwidth assignment on link 5 will become 1/4. Thus, the remaining bandwidth (1/3 – 1/4 = 1/12) should be reallocated to other links. As the traffic rate on link 6 is 1/2, it can utilize the extra bandwidth. Therefore, the new bandwidth allocation on link 6 is 1/3 + 1/12 = 5/12.

Lastly, we determine the bandwidth allocation around B1. Since link 5 can only utilize 1/4 of the bandwidth, the remaining bandwidth (1/2 – 1/4 = 1/4) can be reallocated to link 4. Therefore, according to B1, it is possible to assign 1/2 + 1/4 = 3/4 of bandwidth to link 4. On the other hand, according to M1, it is possible to assign 1/4 + 1/40 = 11/40 of bandwidth to link 4. Therefore, since M1 cannot allocate more than 11/40 of the bandwidth to link 4, the maximum possible bandwidth allocating to link 4 is 11/40. Lastly, as the traffic rate on link 4 is 2/3, the new bandwidth allocation on link 4 will become 11/40. As a result, the final max-min fairness bandwidth allocation for links 1 to 7 is (1/4, 1/5, 11/40, 11/40, 1/4, 5/12, 1/3).

1.4 Motivations and Objectives

The scheduling of a single piconet is well-studied. A number of algorithms has been reported in the literature. On the other hand, the research on scatternet scheduling is still an ongoing task. Some of the design criteria for scatternet scheduling are stated below:

1. Since a bridge node can only participate in one piconet at a time, it is necessary to organize the time for the bridge node to switch between different piconets.
2. When two masters in different piconets try to communicate with a bridge node simultaneously, only one master will succeed. As a result, it leads to a conflict at the

bridge node. Therefore, it is necessary to avoid the chance of bridge node conflict in order to reduce the number of packet loss.

3. For ACL connections with bursty traffic, it is necessary to allocate bandwidth dynamically to each ACL connection based on the traffic conditions. This will lead to better bandwidth utilization.
4. Within a scatternet, all nodes should receive a fair share amount of bandwidth for data transmission. Therefore, a master and a bridge node should fairly share the bandwidth for its connected links based on the traffic conditions. This prevents the possibility of starvation in some nodes.
5. Rather than proposing intra-piconet and inter-piconet scheduling algorithms independently, there may be benefits to design a scatternet scheduling algorithm which can handle both intra-piconet and inter-piconet scheduling.

The previous research work on scatternet scheduling, such as [9][12], focused on issues 1 and 2. In addition to issues 1 and 2, other work such as [20][18] considered issue 3 as well. Besides, the work in [10] considered issues 1, 2, 3 and 5. The work in [13] considered issues 1, 3, 4 and 5. The work in [16] considered issues 1, 3 and 5. Lastly, the recent work in [19] attempted to resolve all the above issues. A detailed discussion of these schemes will be given in Chapter 2.

The goal of our work is to develop a scatternet scheduling algorithm which can satisfy all the design issues described above. Our work aims to improve the throughput and delay performance of ACL connections, maintain the max-min fairness for all the nodes, prevent bridge node conflict, and allocate bandwidth dynamically based on traffic usage.

1.5 Contributions of the Thesis

In this thesis, we propose an Adaptive Scheduling Algorithm (ASA) for Bluetooth scatternets. Our proposed ASA has the following features:

1. Adapt to traffic change;
2. Maintain max-min fairness for all nodes;
3. Prevent the bridge node conflict;
4. Integrate both intra-piconet and inter-piconet scheduling in a single algorithm.

In order to monitor the traffic usage on each node, there is a traffic estimator on each node to estimate the packet arrival rate. By monitoring the size of the queue and some other values, the proposed ASA can use those information to adjust the bandwidth allocation on each link.

In ASA, each master node uses an active list and a waiting list to organize a fair serving order for all of its connected nodes. Moreover, in order to maintain fairness between different piconets, all masters and bridge nodes use a dynamic switch schedule to organize a fair serving order on their inter-piconet links. To avoid more than one master accessing a bridge node simultaneously, all masters and bridge nodes determine their next meeting time and duration dynamically. Lastly, the intra-piconet scheduling scheme in ASA uses the switch schedules information in order to improve the performance in throughput and delay.

1.6 Organization of the Thesis

The thesis is organized as follows: In Chapter 2, we describe the related work on scheduling schemes for Bluetooth piconets and scatternets. In Chapter 3, we propose an Adaptive

Scheduling Algorithm (ASA) for Bluetooth scatternets. We then present the performance comparison on fairness, throughput, and delay with two other scheduling schemes through simulations in Chapter 4. Finally, Chapter 5 concludes the thesis with a summary of work presented and some suggestions for future work.

Chapter 2 - Related Work

Several intra-piconet and inter-piconet scheduling algorithms for Bluetooth ad hoc networks have been proposed in the literature. In this chapter, we summarize these algorithms and point out the strengths and weaknesses of each method.

2.1 Piconet Scheduling

In this section, we summarize various intra-piconet scheduling algorithms proposed recently. Most of these algorithms aim to provide a high throughput, a low average delay, and to maintain fairness among all the slave nodes.

2.1.1 Pure Round Robin (PRR)

The original and default piconet scheduling scheme in Bluetooth is PRR [2]. In this scheme, the master polls each slave in a fixed cyclic order. All slaves take turn to transmit data. The advantage of this scheme is that it maintains fairness by allowing each slave has a chance to transmit in each polling period. Moreover, it is simple to implement. However, if both master and slave nodes have no data packets to send, the master will still need to send a POLL packet to the slave and the slave will reply with a NULL packet. This POLL-NULL event results in a decrease in average throughput and an increase in delay of data transmission.

2.1.2 Exhaustive Round Robin (ERR)

In ERR in [2], a master polls a slave exhaustively and does not switch to the next slave until both master queue and slave queue are empty. However, in order for the master to determine whether or not the slave queue is empty, the slave is required to piggyback the status of its

queue when it sends a packet to the master. The advantage of this scheme is that it can increase the throughput by reducing the number of wasted slots spending in POLL-NULL events. Since the master polls its slaves exhaustively, the average cycle time of ERR is longer than the average cycle time of PRR. Therefore, if there exists one master-slave pair with empty queues, both the master using PRR and the master using ERR will encounter a POLL-NULL case in a cycle. Since the average cycle time of PRR is shorter than that of ERR, the master in PRR will encounter for more POLL-NULL events than the master in ERR. However, as the master continues to poll the same slave until this master-slave pair has no data to send, it may cause starvation to other slaves in the piconet. Thus ERR does not provide fairness to all the nodes.

2.1.3 K-limited Round Robin (K-limited RR)

In the K-limited RR algorithm [2], the master polls a slave exhaustively but is limited to K times per cycle. Therefore, even though either the master or the slave may still have packets in the queue, once the master has finished K transmissions on the current master-slave pair, it will move on to serve the next slave. The advantage of K-limited RR is that it can retain the fairness generated by PRR. However, it has a lower throughput when compared with ERR.

2.1.4 Limited and Weighted Round Robin (LWRR)

The LWRR scheme proposed in [2] aims to reduce the number of POLL-NULL events. In this scheme, each slave has its own weight and is initially set to Maximum Priority (MP), which is a predefined value. In LWRR, if a POLL-NULL event happens on a master-slave pair, the master will decrement the weight of the slave by one. The lowest weight a slave can have is equal to one. As a result, the master will skip polling the slave for "MP minus

weight" cycles. Once a data transmission exists on the master-slave pair, the weight of the slave will be reset to MP. The advantage of this scheme is that it can predict the queue state on a master-slave pair and prevent the master to poll a slave with empty queue. However, this scheme may not react quickly to bursty data traffic when a slave is having its punishment during the skipping cycles.

2.1.5 Deficit Round Robin (DRR)

Another scheme called DRR [4] also aims to maintain fairness between all master-slave pairs. In this scheme, each master-slave pair maintains a state variable called *credit*. Each slot of data transmission consumes one credit. The scheme works as follows: each round before a master tries to poll a slave, a value of *quantum* is added to the master-slave pair credit account. Then the master checks whether the accumulated credits are large enough for the head-of-line (HOL) packets pair, which are the packets at the front of both master and slave queues. In the first case, if the accumulated credits are large enough for the size of the HOL packet pair, the master will poll the slave, deduct the total size of the transmitted packets from the accumulated credit, and check the size again. The process will repeat if the remaining credits are still large enough for the next HOL packet pair. If not, the master-slave pair will keep the credits and move on to the next slave. In the second case, if the accumulated credits are not enough for the size of the HOL packet pair after a quantum of credit is added to the pair, the current master-slave pair will keep the accumulated credits. The master will then move on to serve the next slave. Lastly, if both master and slave queues are empty, the master will reset the credit to zero and move on to the next slave. Since the master has to know the size of HOL packet in the slave's queue, a slave has to piggyback the

size of the next HOL packet in its queue when it sends a packet to the master.

The advantage of this scheme is that it can balance throughput and fairness among all master-slave pairs by assigning equal number of credits to each pairs. However, the current DRR scheme used in bluehoc [4] is just a simple modification of the scheme in [5]. In bluehoc, the DRR scheme only modifies the credits on master but not on slave. Therefore, it does not consider master and slave as a whole. As the original DRR scheme in [5] is for high-speed routers, its use in Bluetooth may require some modifications. Moreover, further work is required to investigate the value of quantum size in order to maintain fairness among all the slaves.

2.1.6 Look Ahead Round Robin (LARR)

The algorithms discussed in the previous sub-sections only consider ACL links. The LARR algorithm in [3] considered both SCO and ACL links. In this scheme, it first serves all slaves in RR fixed order and tries to examine the HOL packets in both the master and slave queues. If the current HOL packet pair fits into the current ACL frame, the master will serve the master-slave pair. Otherwise, the master will look ahead for the next master-slave pair whose HOL packet pair can fit into the current ACL frame by following the RR fashion. If the master cannot find any HOL packet pair that can fit into the current ACL frame, it will wait until the start of next ACL frame and give the turn to the first examined master-slave pair. The advantage of this scheme is that it can maximize the ACL frame usage and reduce the chance of packet loss. If a pair sends packets that do not fit into the ACL frame, it will result in a packet loss. If only ACL link is present, then LARR is equivalent to the RR scheme.

2.1.7 K-Look-Ahead Round Robin (K-Look-Ahead RR)

In order to maintain a stable time for the look-ahead process, K-Look-Ahead RR [3] scheme is used to maintain a fixed number of examinations of the HOL packets. In this scheme, if the current HOL packet pair does not fit into the current time frame for ACL, which is referred as ACL frame, the master will only look ahead for the next K pairs. If the master cannot find a HOL packet pair that can fit into the current ACL frame within the next K pairs, it will wait until the start of next ACL frame and give the turn to the first examined master-slave pair. The advantage of this scheme is that since it only searches for the next K HOL packet pairs, it can ensure that the process is running in linear time and uses fixed space. If LARR is used instead, the number of search in each round will not be stable because it searches all the remaining pairs in a piconet. However, the K-Look-Ahead RR wastes more slots in an ACL frame than LARR because many times the HOL packet pair that can fit into the current ACL frame may exist after K pairs. Therefore, if K-Look-Ahead RR gives up the turn after examining K pairs, it will waste all the remaining slots in the ACL frame.

2.1.8 Adaptive Flow-based Polling (AFP)

The AFP [6] scheme uses the status of queues to modify the polling interval for each master-slave pairs. In this scheme, it makes use of the flow bit in the payload header field of the packet to indicate the status of the queue. The flow bit is set to 1 when the number of packets in the queue exceeds a threshold value, which is defined as *buf_thresh*. AFP also uses another parameter called *flow* to represent the traffic status. If the flow bit is set to 1 in the packet transmitting in either uplink or downlink, AFP will set *flow* to 1. In AFP, all slaves initially have the same polling interval, which is defined as *P*. Each time when a POLL-

NULL event occurs on a master-slave link, the master will double the current polling interval for the slave. The master stops doubling the polling interval when polling interval is greater than or equal to a threshold value. The master resets the polling interval back to *P* if the *flow* is set to 1 and the HOL packet in the master to slave queue is a data packet.

The advantage of this scheme is that it can reduce wasted slots spending on POLL-NULL events by increasing the polling interval. In addition, when the master detects a high flow rate for the slave, it serves the slave more frequently by setting the polling interval back to the original value. Although AFP looks similar to LWRR, their conditions on setting the polling interval (polling cycle for LWRR) back to the original value on a slave are different. For LWRR, it resets the polling cycle on the master-slave pair when a data transmission exists. Different from LWRR, AFP only resets the polling interval on master-slave pair when it detects a high flow rate on the pair. The disadvantage of this scheme is that the master-slave pairs with low data rate have to suffer for a long delay until they receive bursty traffic or accumulate enough packets in the queue.

2.1.9 Sticky Adaptive Flow-based Polling (Sticky AFP)

Sticky AFP [6] is similar to AFP. The modification on Sticky AFP is that if the *flow* is set to 1 and the HOL packet in the master to slave queue is a data packet, the master will not only set the current polling interval back to the original value, but will also allow a maximum of *num_sticky* packets data transmission between the master-slave pair. The advantage of this scheme is that it can reduce the queue size quickly by transmitting multiple packets consecutively; therefore, it can prevent overflow of queue. However, since some master-slave pairs may transmit up to *num_sticky* packets, the cycle time will be longer and the

average delay on individual slave will also be longer.

2.1.10 HOL Priority Policy (HOL-PP)

In HOL-PP [6][7], it distinguishes different master-slave pairs based on the state of HOL packets in both master and slave queues. Since HOL-PP scheme considers both ACL links and SCO links, it only allows packets occupying either 1 slot or 3 slots. If there is no packet in the queue, HOL-PP uses "0" to represent the status of the queue. If there is a HOL packet with size 1, then "1" is used to represent the status of the queue. If there is a HOL packet with size 3, then "3" is used to represent the status of the queue. Because a master needs to know the status of the slave queues, the free bits in the Bluetooth payload header are used to communicate the status of slave queues to the master. When a 4-slot ACL link is used for data transmission, then the master-slave pairs with either 1-1, 3-1, or 1-3 state, which utilize 100% of the 4 slots, have class 1 priority with priority value P1. Master-slave pairs with either 3-0 or 0-3 state, which utilize 75% of the 4 slots, have class 2 priority with priority value P2. Master-slave pairs with either 1-0 and 0-1 state, which utilize 50% of the 4 slots, have class 3 priority with priority value P3. Lastly, master-slave pairs with 0-0 will not be scheduled. In addition, the value of P1 has the highest priority while the value of P3 indicates the lowest priority.

The policy of HOL-PP is that a master first polls all slaves in RR fashion. If the master-slave pair has priority value P1, it will serve the master-slave pair P1 times; it applies the same policy on master-slave pairs with a lower priority class as well. However, if the master-slave pair changes priority class while in service, the master will stop serving the current slave and move on to the next slave.

The advantage of this scheme is that it reduces the wasted slots spending on node with empty queue by yielding data transmissions to master-slave pairs with higher priority class; therefore, it can increase the system throughput. However, in this scheme, it does not mention how the master-slave pair with 0-0 status returns to be scheduled by the master. If the pair has to wait until the master has data to send, then a bursty traffic on a slave may overflow the slave's buffer. Moreover, the HOL-PP algorithm did not mention how to deal with the master-slave pair with 3-3 status since there are only 4 slots available for ACL link.

2.1.11 HOL K-Fairness Policy (HOL-KFP)

The HOL-KFP [6][7] scheme uses the same method as HOL-PP to assign priority status to all master-slave pairs. In addition, HOL-KFP keeps a counter for each master-slave pair. The counter on each pair will only be changed in two cases. The counter will be decremented on the pair who sacrifices its service to another pair. The counter will be incremented on the pair who receives the sacrificed service from another pair. HOL-KFP also uses the variable Q_{max} to keep track of the total number of service on the pair who receives the maximum number of excess service from other pairs and uses the variable Q_{min} to keep track of the total number of service on the pair who sacrifices the maximum number of service to other pairs. In HOL-KFP, the master serves all slaves in RR fashion. However, class 2 pairs will sacrifice service to class 1 pairs. Class 3 pairs will sacrifice service to class 1 pairs first and then to class 2 pairs. In addition, the transfer of service is only allowed if "$Q_{max} - Q_{min}$" is smaller than K where K is defined as a fixed number. Therefore, if a master-slave pair is not allowed to transfer service to pairs with higher class, it will be serviced according to its priority class. The advantage of this scheme is that it can guarantee a strict fairness bound by using the

number K. The number K represents the maximum unfairness existing between any two backlogged master-slave pairs. However, the issues of 0-0 status and 3-3 status in HOL-PP remain un-resolved in HOL-KFP. Moreover, once the fairness bound exceeds K, no pair can sacrifice service to others and no pair can gain service from others. When that situation occurs, HOL-PP scheme is equivalent to the RR scheme.

2.2 Scatternet Scheduling

In this section, we summarize various scatternet scheduling algorithms proposed recently. These algorithms can be divided into two groups: deterministic and random based.

2.2.1 Deterministic-based Scatternet Scheduling Algorithms

The scheduling schemes to be described in this sub-section focus on when the bridge node should be present in each connected piconet and how the master should communicate with the bridge node. Both the bridge node and the master node know the scheduled time for them to communicate with each other.

2.2.1.1 Rendezvous Scheduling

The rendezvous scheduling algorithm [8][9] focuses on inter-piconet scheduling. Two terms are introduced in this algorithm. The first term is the *rendezvous point* (RP). RP is the time slot at which the master agrees to poll the bridge node and the bridge node agrees to listen to the master. The second term is the *rendezvous window* (RW). RW is the period of time that the master and the bridge node spend to communicate with each other. This algorithm assumes that the bridge node can only be a slave-slave node. Because the bridge node switches between different piconets at each RP, RW is actually the distance between two RPs.

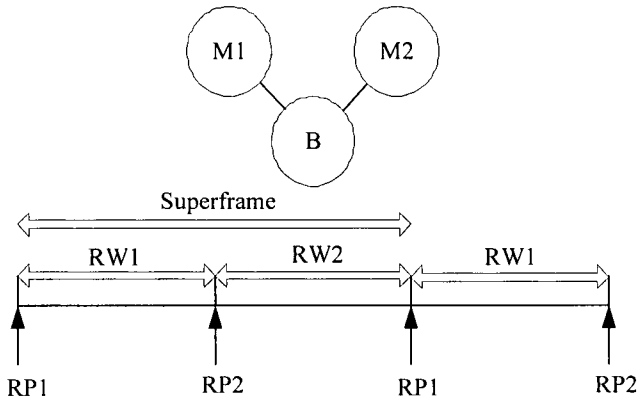

Figure 2.1. Rendezvous scheduling

Figure 2.1 shows the relationship between RP and RW. As each piconet has limited RW to communicate with a bridge node, the interval of RW should be maximized. Therefore, the *Maximum Distance Rendezvous Point* (MDRP) algorithm in [9] aims to maximize the distance between the RPs. In MDRP, it defines a periodic *superframe* that is a time period for all the nodes involving in the bridge node scheduling.

When a new piconet joins the bridge node, the RP between the new piconet and the targeted bridge node will be chosen from the middle slot of the largest interval between two different successive RPs. For instance, in Figure 2.1, if RW1 is larger than RW2, then RP3 will be added in the middle of RW1. Thus, the original interval of RW1 will be halved and the interval of RW3 will be equal to the new value of RW1.

In order to implement RP and RW, MDRP makes use of the Bluetooth sniff mode. Because a bridge node in sniff mode will only listen to the master for $N_{sniff\_attempt}$ slots, it can then stop listening to the master after $N_{sniff\_attempt}$ slots and join another piconet. At the beginning of next T_{sniff} period, the bridge node will listen to the master for $N_{sniff\_attempt}$ slots again. Since the RW in rendezvous scheduling is fixed, $T_{sniff\_timeout}$ is set to 0. Therefore, each RP will map to the beginning of a T_{sniff} period between the master and the bridge node, and

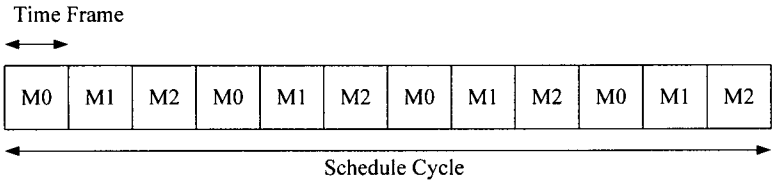

Figure 2.2. Switch-table for a bridge node connected with 3 masters

each RW will map to the $N_{sniff\_attempt}$ variable.

The advantage of Rendezvous Scheduling scheme is that it is simple to implement and can maintain a stable performance by having a fixed schedule. However, it does not consider short-term traffic changes. Since the size of each RW is the distance between two successive RPs, the number of master-bridge pairs limits the time slots allocated for each RW. When bursty traffic exists in one of the master-bridge pairs, the MDRP cannot flexibly change the RW for that pair. Moreover, the MDRP scheme does not maintain fairness for all the nodes within the scatternet. Therefore, MDRP may assign more bandwidth to a link than it should be received.

2.2.1.2 Flexible Scatternet-wide Scheduling Algorithm (FSS)

The FSS [10] scheme provides inter-piconet scheduling by placing a *switch table* in each bridge node. Each switch table specifies the starting time and ending time for the communication between the bridge node and master in each connected piconets. Initially, each connected piconet obtains the same number of *time frame* to communicate with the bridge node. The size of the time frame can be either 2, 6, or 10 Bluetooth time slots. However, all time frames in the switch table must have the same size. If there are three piconets connected with the bridge node and the switch table contains four consecutive scheduling sequences for all of them, then the sequence will repeat its cycle after three time frames and the switch table will repeat its cycle after 12 time frames. Figure 2.2 shows the

switch table for the bridge node described above.

The switch table is constructed at the time when the scatternet is formed. FSS assumes using centralized protocol such as the Bluetooth Topology Construction Protocol (BTCP) [11] to construct the topology so that the leader has the full knowledge of the scatternet. Once the switch table is formed, it will be saved at the bridge node and will be propagated to all masters in the connected piconets. As a result, the master knows when it should poll the bridge node and the bridge node knows when it should listen to the master.

In order to co-ordinate with the switch tables, FSS also maintains a flexible traffic scheduling algorithm for intra-piconet scheduling. In this algorithm, the master polls all the slaves following the weighted round robin fashion in each schedule cycle in the switch table. In FSS, the polling weight of each slave is represented by a tuple (P, R). The parameter P denotes the slave will be polled by the master every P schedule cycles, and the parameter R denotes the slave can receive maximum R times of polling in a schedule cycle. To provide flexibility for scheduling, FSS dynamically changes the weight on each slave depending on the traffic load observed from the master to slave queue. When there are traffic changes, FSS will first change the parameter P, and then it will change the parameter R. Each time when two consecutive polling slots are wasted, the parameter P for that slave will be increased by 1 until it reaches the maximum threshold value. On the other hand, if the slave does not waste the polling slots in a cycle, then P will be decreased by 1 until it reaches 1. However, if the current value of P is 1, FSS will start changing the parameter R. Each time a slave wastes polling slots, the parameter R for that slave will be decreased by 1 until it reaches 1. If the polling slots are not wasted, the parameter R of that slave will be increased by 1 until it reaches the maximum threshold value. Lastly, once the parameter of R reaches 1, FSS will

start modifying the parameter P again.

FSS gives higher priority to bridge node than pure slaves. At the beginning of each frame in a scheduling cycle, FSS checks to see if there is any SCO link reserving the time frame. If there is a SCO link, FSS will let the master poll the slave that is connected with the SCO link. Otherwise, FSS will start searching for a bridge node that has been polled the least within the scheduling cycle and can be polled by the master according to the switch table. If FSS is able to find a bridge node satisfying the requirements, it will allow the master to poll the bridge node. If FSS cannot find any bridge node satisfying the requirements, it will then search for each slave in a weighted round robin fashion. If FSS finds a slave that satisfies the (P, R) requirement on the schedule cycle, it will allow the master to poll the slave. Otherwise, the master will become idle for this time frame. At the end of the time frame, the master modifies the (P, R) parameters of a slave which has been polled in the current scheduling cycle according to the slave's status.

FSS modifies the bandwidth allocation for a master-bridge link by monitoring both the outgoing queue length and the incoming queue length of a bridge node. Each time when a master sends a packet to a bridge node, it updates its current queue length as well. When the total queue length in a master-bridge link exceeds a threshold value, the bridge node tries to find a *lender* to the link from all connected masters. The candidate lender will be the master-bridge link with the shortest queue length and has been assigned for more than one time frame in the switch table. If the bridge node is able to find a lender and the lender agrees to lend one of its time frames, then the switch table will be updated; a time frame from the lender will become the time frame for the *borrower*.

This scheme can guarantee that there is no conflict existing at the bridge node. Since

all masters and the bridge node know when they should communicate with each other, no two masters will poll the bridge node at the same time. Moreover, it also creates flexibility to bridge node traffic because a bridge node assigns different number of time frames to each master based on traffic change. However, the modification of a switch table involves many messages exchange between different masters and a bridge node. Therefore, a bridge node should not modify the switch table frequently. In addition, since this scheme gives higher priority to master-bridge links than master-slave links, it does not maintain fairness among the nodes in a scatternet.

2.2.1.3 Jump Mode

In order to improve the efficiency of inter-piconet scheduling, a new mode called the *jump mode* is proposed in [12] for bridge nodes in a scatternet. When a bridge node is in jump mode, it acts as a jumping node and is considered to be absent in the piconet. In this scheme, a bridge node can either be a master-slave node or a slave-slave node. The jumping node timeline is divided into time window called rendezvous window (RW) [8][9] with pseudo random length. The beginning of each RW is called the rendezvous point (RP) [8][9]. The master polls the jumping node at each of the RPs. If the jumping node wants to be present in a piconet, it has to signal the master in the piconets at one of the RPs. The jumping node will spend one or more RWs in a piconet before switching to another piconet. Once the jumping node is connected to the piconet, it follows the piconet scheduling scheme to communicate with the master.

In order for a jumping node and its respected master nodes to create the same pseudo random sequence of RWs, both the jumping node and all connected masters use the same size of time period, (which is defined as Nsf), and the same seed number, (which is defined

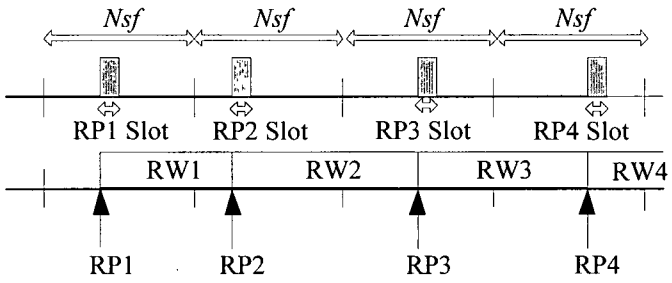

Figure 2.3. Pseudo random sequence of RWs

as the BD_ADDR of the jumping node), to implement the sequence. At the beginning, a timeline is divided into periods of *Nsf* frames, which contain a fixed number of Bluetooth time slots. Then in each *Nsf* period, the jumping node and all connected masters will pseudo randomly choose one slot to be the RP. Since all nodes use the same seed number to generate the pseudo random sequence for each period, the jumping node and all connected masters will generate the same pseudo random sequence. Lastly, the time window between the RP in current period to the RP in the next period becomes the RW. Figure 2.3 shows the process of generating pseudo random sequence of RWs.

The scheduling of the bridge node is described as follows. Since all connected masters know all the RPs at the bridge node, they will poll the jumping slave at each of the RPs. If the bridge node responds to the master, the master knows that the bridge node will be present in its piconet until the next RP. Therefore, the job of the jumping node is to choose the RP for each connected master and to respond to each of the master's polling messages respectively. Furthermore, in order to reduce the number of polling for each connected master, this scheme also provides a mixed long-term schedule option. In this option, the jumping node maintains a bitmap, which specifies if the jumping node will be present, absent, or undefined in the following certain number of RWs, for each of the connected master. Therefore, the master will only poll the jumping bridge node at the specified RPs in the

bitmap. On the other hand, if the jumping node is a master, then the jumping master node will set up a periodic schedule. The periodic schedule specifies the RWs allocated to all connected piconets. With the information in the periodic schedule, a slave, which connects to the jumping master node, can choose to adapt the schedule or not.

The advantage of this scheme is that the master can still communicate with other slaves while the bridge node is connected to the piconet. In other schemes, such as MDRP [9] and FSS [10], when the bridge node switches to a piconet, the master can only exchange data with the bridge node, but not with other slave nodes. However, in jump mode scheme, when the bridge node switches to a piconet, it just acts as a normal slave. It follows the piconet scheduling scheme to schedule the time for it to communicate with the master. Therefore, it can save time slots if there is no data transmission between the master and the bridge node. Moreover, it allows two bridge nodes to exist in a single piconet at the same time so that it can reduce the delay for data transmission passing through two or more piconets. In addition, by using the pseudo random sequence, it can reduce the chance for a master to poll two bridge nodes at the same time. Since a master uses different pseudo random sequence with different bridge nodes, it can guarantee that systematic conflicts do not exist. Moreover, by using the jump mode, it can eliminate the bridge node conflict issue as well. Since the bridge node is responsible for choosing the RPs for each of the connected masters, no two masters can access the bridge node at the same time.

The disadvantage of this scheme is that it does not specify how the bridge node schedules the order for serving each of the connected masters; therefore, a further analysis on the RP scheduling at the bridge node is necessary. Moreover, because the master has to poll the jumping node at each RP, it will interrupt the intra-piconet scheduling scheme for all the

```
if (time in piconet A > MTS) or (TC expired and (queue size to piconet B > MQS or IS))
  set TC = min (β * queue size to piconet B, MTS)
  switch to piconet B
```

Figure 2.4. Pseudocode of LAA

pure slaves. A further investigation on the coordination between the jump mode scheme and the intra-piconet scheduling scheme is necessary. Lastly, although a jumping node can maintain a bitmap for each of the connected masters, the message overhead is very high.

2.2.1.4 Load Adaptive Algorithm (LAA)

LAA [20] is an inter-piconet scheduling scheme targeted for small-scale scatternets. A scatternet is defined as a small-scale scatternet when there are only two piconets connected by a bridge node. LAA assumes the use of slave-slave bridge as a master-slave bridge may cause poor bandwidth utilization [17]. In LAA, a bridge node can use either hold mode or sniff mode to switch between piconets.

In LAA, the bridge node uses five parameters to determine the time for it to switch between two masters. The first parameter is *Idle state* (IS). If the queue of bridge node to a master is empty and the bridge node receives a POLL packet from the master, it will then enter IS. Once a bridge node enters IS, it will try to switch to another piconet. The second parameter is *Max queue size* (MQS), which has a predefined value. If the queue size of a bridge node to a master reaches MQS, the bridge node will try to switch to another piconet. The third parameter is *Time commitment* (TC), which defines the minimal communication time between a bridge node and a master. The value of TC depends on the queue size of bridge node to master. The fourth parameter is the *predictability factor* (β), which has a predefined value. It is used to estimate the future queue size of a bridge to a master in order to calculate the value of TC. The last parameter is *Max time-share* (MTS), which is used to

limit the communication time between a bridge node and a master node. By using the above parameters, the bridge node can decide the time to switch between two piconets. Figure 2.4 shows the pseudocode of LAA [20]. Once a master connects to a bridge node, it serves the bridge node by using an exhaustive rule in order to empty the queue of the bridge node.

The advantage of this scheme is that it can dynamically allocate bandwidth between two masters and a bridge node based on different values of load. It is also simple to implement. However, since the scheme is only applied in small-scale scatternet, it does not prevent bridge node conflict. Moreover, since a master exhaustively polls a bridge node until the bridge node's queue is empty, it does not maintain fairness among nodes in a scatternet.

2.2.1.5 A Fair and Traffic Dependent Scheduling Algorithm

The algorithm proposed in [19] aims to dynamically allocate bandwidth to each link within a scatternet and maintain fairness among nodes. This scheme assumes the use of slave-slave bridge nodes since a master-slave bridge may cause poor bandwidth utilization [17]. The scheme integrates both intra-piconet scheduling and inter-piconet scheduling.

In the intra-piconet scheduling module, the master uses the PRR scheme to poll each connected node. However, it also maintains an *active list*, in which a slave node will move in and out based on traffic estimation. If a slave node is not present in the active list, the master will skip polling the slave in the current round. As a result, this scheme can dynamically allocate bandwidth to all the connected slaves when there is traffic change. In order to estimate the traffic, the master node keeps track of two variables. The first variable is r, which is the estimated rate of traffic on the slave node. The second variable is N, which is the estimated queue length of the slave node. The scheme updates the estimated values of N and r by monitoring the data exchange between the master and the slave following the algorithm

For the slave just polled:

$N = N + r\tau - x$

$r = \alpha r + (1 - \alpha)(x/T)$ when $(x < M)$

$r = \alpha r + (1 - \alpha)(x/T) + \delta$ when $(x = M)$

For other slaves:

$N = N + r\tau$

Figure 2.5. Algorithm for updating the estimation of N and r on slaves

showed in Figure 2.5. In the algorithm, M denotes the maximum amount of data allowed to be exchanged; x denotes the amount of data exchanged; τ denotes the time difference between the current time and last update time; T denotes the time difference between the current time and the time for the last poll of the slave; α, which has a predefined value, is used for maintaining a stable rate estimation; δ, which also has a predefined value, is used for assigning more bandwidth to a slave when it can fully utilize the bandwidth.

Therefore, if the updated N value of a slave is less than a predefined "threshold" value, it will be removed from the active list. Otherwise, the master will add the slave back to the active list. The scheme chooses the size of three DH5 packets as the threshold value, and limits the maximum skipping cycles on a slave to 5 cycles. In addition, when considering a bi-directional traffic flow, x denotes the average of data exchanged between two nodes, r denotes the average of the estimated traffic rate between the two nodes, and N denotes the average of the estimated queue length for the two nodes.

In the inter-piconet scheduling module, it re-uses the term Rendezvous Point (RP) [8] to describe the Bluetooth slot at which a master meets with a bridge node. This scheme uses the hold mode to implement the RPs between a master and a bridge node. Before a bridge node switches to another piconet, the master negotiates the next RP with the bridge node, and then puts the bridge node into hold mode until the time for the next RP. Moreover, after

finishing the communication with the bridge node, the master node continues polling the slave nodes until the arrival of next RP. In order to determine the next RP, both the bridge node and the master node perform local estimation for the next possible RP between them.

In order to determine a possible RP for each round, a bridge node keeps track of the estimated value of N and r for each connected piconets, which is similar to the traffic estimation at a master node. However, instead of updating the value regarding to a slave, it updates the value regarding to a connected master. The bridge node monitors the data exchange between itself and the master and uses the algorithm showed in Figure 2.5 to update the values of N and r. Therefore, at each RP, the bridge node first determines how many Bluetooth slots, N_{thresh}, are available until the value of N for the current connected master node to reach the threshold value. In order to prevent a long waiting time for the next RP, the value of N_{thresh} is less than 400 Bluetooth slots. After calculating the value of N_{thresh}, the bridge node then sends N_{thresh} together with all its RPs with other masters to the current connected master node.

On the other hand, the master tries to find a possible RP for each round with which it can also maintain fairness for all connected slave nodes. In order to find the next RP, the master uses a counter called *num_slots*. At each RP, the master checks the connected slaves in a cyclic order. It begins from the slave after the current connected bridge node to the slave before the current connected bridge node. The master first sets *num_slots* to 0. It then checks whether the value of N for the slave exceeds the threshold after *num_slots* Bluetooth slots. If it is true, the master will increase *num_slots* by the value of threshold times two. Moreover, if any RP has the same value as *num_slots*, the master will again increase *num_slots* by the value of threshold times two. The process continues until it reaches the last slave node in the

Slots left for RP	Maximum length of packet sent by master	Maximum length of packet sent by slave
2	1	1
4	1	1
6	3	3
8	3	3

Table 2.1. Procedure adopted by the master if slots available for the RP is less than 10

cycle. By checking and reserving all possible polling time for each connected slave nodes, it can maintain fairness for all connected slave nodes. Again, in order to prevent a long waiting time for the next RP, the value of *num_slots* is bounded to be less than 400 Bluetooth slots. After calculating the value of *num_slots*, the master then sends *num_slots* together with all its RPs with other bridge nodes to the current connected bridge node.

After exchanging the information, both bridge node and master node can now finalize their next RP. Both of them first pick the largest of N_{thresh} and *num_slots* as the initial negotiated RP. If the value of negotiated RP is the same as an RP in either master node or bridge node, the value of the negotiated RP will be increased by the value of threshold times two. The process continues until the value of negotiated RP is not the same as any RP in either the master node or bridge node. Both master and bridge node then set the final negotiated RP as their next RP.

Sometimes the Bluetooth slots available before the beginning of an RP may not be enough for the size of packets going to exchange between the master node and the slave node. An additional rule is implemented to deal with this situation. As the maximum size of Bluetooth packet is DH5, which uses five Bluetooth time slots, the rule will only be triggered when the Bluetooth slots available for an RP is less than 10. Table 2.1 shows the procedure [19] adopted by the master to handle the situation. For example, if 6 Bluetooth slots are

available before an RP, the master will be allowed to send a maximum size of 3-slot packet and the polled slave will also be allowed to reply a maximum size of 3-slot packet. The reason for setting the same maximum length packet size on both sides is to maintain fairness between master node and slave node.

The advantage of this scheme is that it can maintain bandwidth fairness among nodes within the scatternet. At the same time, it can also adapt to traffic changes.

However, the traffic estimation algorithm for this scheme is suitable for stable traffic rate, but may not be suitable for bursty real-time traffic. In a dynamic environment, it is difficult to predict accurately the future traffic. Therefore, a traffic estimation error will affect the performance. Moreover, when considering bi-directional traffic, this scheme may lead to a long waiting time if one side has a high traffic rate and the other side has a low traffic rate. As a result, it increases the packet delay on the node with a higher traffic rate. Lastly, this scheme sets a limit to the maximum packet size sending between two nodes when the Bluetooth slots available before an RP is less than 10. However, scheduling occurs after the L2CAP layer has segmented the packet. Therefore, forcing the node once again to reduce the packet size does not strictly adhere to the current Bluetooth specification. In addition, the scheme did not discuss how to deal with the second half of the packet when the original packet has segmented to fit the size specified in Table 2.1.

2.2.1.6 A Locally Coordinated Scatternet Scheduling Algorithm (LCS)

The LCS algorithm [18] aims to optimize the throughput, delay, and energy. In addition, it also aims to adapt to the dynamic traffic conditions. In this scheme, each master node first meets with a child node (which can either be a slave node or a bridge node) and exchanges packets with the child node during the meeting time. It then negotiates the start time and the

minimum duration of the next meeting time with the child node. Afterwards, the master polls the child node again at their confirmed next meeting time. In order to negotiate the start time and the minimum duration of the next meeting time between the master and the child node, both nodes maintain local traffic information to make the decision.

In order to update the traffic change, both master node and child node monitor four variables during the meeting time. The first variable is *tx*, which stores the number of slots used for exchanging packets during the meeting time. The second variable is *qsz*, which stores the combined queue size for both nodes. The third variable is *r*, which stores the value of recess interval. The recess interval is the time difference between the start time of the current meeting and the finish time of the previous meeting. Lastly, the fourth variable is $u = tx/d$, which stores the link utilization. The value of d is equal to the duration of the current meeting time. By monitoring *tx* and *qsz*, it can update the average value of *avg_tx* and *avg_qsz* in an exponentially weighted moving fashion. Moreover, by monitoring *r* and *u*, it can update the average value of *avg_r* and *avg_u* based on the past *Nr* values of *r* and *u*, where *Nr* has a predefined value.

Based on the values of *avg_tx* and *avg_qsz*, both master and child nodes determine the minimum duration for the next meeting, d^{t+1}. Since the values of *avg_tx* and *avg_qsz* re influenced by traffic changes, LCS can vary the value of d^{t+1} to adapt the change. In order to maintain a stable condition, d^{t+1} is upper-bounded by D_{max} and lower-bounded by D_{min}. Both D_{max} and D_{min} have predefined values. At the beginning, d^{t+1} is set to the larger value of *avg_tx* and *avg_qsz*. Then, if the queue is not empty at the time when the node updates the value of d^{t+1}, LCS will increase d^{t+1} by $K_{qsz} \times qsz$ slots. K_{qsz} has a predefined value and is set to 0.5. By increasing the value of d^{t+1}, it can reduce the negotiation overhead. Lastly, if the

current value of *qsz* is larger than the current d^{t+1}, d^{t+1} will then be set to *qsz*.

On the other hand, based on the values of *avg_tx*, *avg_r* and *avg_u*, both master and child nodes determine the recess interval before the next meeting begins, r^{t+1}, and the start time of next meeting, s^{t+1}. By evaluating the values of *avg_tx*, *avg_r* and *avg_u*, it can determine whether the data rate is increasing, decreasing or stable. LCS uses α, which is defined as a multiplying factor, to modify the value of r^{t+1}. If the data rate is increasing, LCS will decrease the value of r^{t+1} by α. The link will receive more bandwidth. However, if the data rate is decreasing, LCS will increase the value of r^{t+1} by α. The link will receive less bandwidth. At the special case, when LCS updates the value of r^{t+1} after a long period of idle time, it will immediately set r^{t+1} to R_{init}. Lastly, if the data rate is stable, LCS will adjust the value of r^{t+1} so that *avg_u* will be close to *tar_u*, which has a predefined value. In order to maintain a stable condition, r^{t+1} is upper-bounded by R_{max}. Finally, the value of s^{t+1} will then be equal to r^{t+1} + the current clock value.

After determining the values of s^{t+1} and d^{t+1}, the master and child nodes are ready to negotiate for the next meeting time. In order to minimize the gap between two meeting time from different links, LCS modifies s^{t+1} of a link to the time right after the finishing time of the closest meeting of another link with similar data rate. With the finalized value of s^{t+1}, LCS chooses N_{meet} number of vacant intervals after s^{t+1}. Each vacant interval (v_s^i, v_f^i) represents the free time interval between two meeting periods of the node. At the beginning of the negotiation process, the master node sends N_{meet} vacant periods, d^{t+1}, and *qsz* to the child node. With the *qsz* value from the master node, the child node calculates its own d^{t+1}. Then the child node just picks the larger of the two as the finalized d^{t+1}. Since the child node also has its own N_{meet} vacant periods, it picks the earliest time interval, which falls within the

vacant period for both the master node and the child node and also has value greater than d^{t+1}, as the next meeting time. Finally, the child node replies the next meeting time to the master.

The advantage of this scheme is that it can dynamically allocate bandwidth based on traffic change. By dynamically allocating bandwidth, it can also reduce wasted slots. As a result, it can improve the efficiency of throughput and latency. Furthermore, the scheme can reduce energy usage since the node could go to the sleep state when there is no scheduled meeting time. Moreover, since each master-child link has distinct meeting time, it can prevent bridge node conflict.

However this scheme does not maintain bandwidth fairness among nodes within the scatternet. When LCS decides the meeting time between the master and child nodes, it does not consider any method to maintain fairness for other connected nodes. Although the scheme suggested the future work of employing DRR [5] to maintain max-min fairness [15], simply applying DRR on Bluetooth scatternet may not guarantee the max-min fairness. Since scatternet scheduling has to consider the time for the bridge node to switch between piconets and the time division shared by two connected nodes, LCS needs to implement a modified version of DRR in order to achieve the max-min fairness.

2.2.2 Individual Node Based Scatternet Scheduling Algorithms

The individual node based scatternet scheduling algorithms do not have a deterministic switching piconet schedule for the bridge node. Instead, a flexible scatternet scheduling scheme is maintained by each node within a scatternet.

2.2.2.1 Credit Based Scheduling (CBS)

The CBS scheme [13][14] applies a priority scheme to master nodes and bridge nodes in

order to decide when the master should poll a connected node and when the bridge node should listen to a master node. At the beginning, CBS puts all nodes connected to the master into the sniff mode. A master node is required to manage a number of sniff slot events respected to each of the connected nodes. Since a bridge node is connected to more than one master, a bridge node is also required to manage a number of sniff slot events respected to each of the connected masters. Thus, based on the link's priority, a master will decide whether to poll a node at its sniff slot and a bridge node will decide to which master node it should listen. In order to derive a priority scheme, each node maintains a credit account for each of its link. A link with more credits in its account has a higher priority class among all other credit links. Initially, all links from a master node and a bridge node are assigned with zero credit. If a link uses a Bluetooth slot for transmitting or listening data, one credit will be deducted from the link's credit account. In order to maintain a fixed number of total credits for all credit links from a master node and a bridge node, when one credit is deducted from a credit link, one credit is added to a temporary account. If the credit in the temporary account reaches n, which is the number of credit links connected to the node, then the temporary account will be reset to zero, and the credit in each credit link is increased by one. With CBS, an ongoing sniff event will yield its turn to another link's upcoming sniff slot if the credit in the upcoming link is higher than the credit in the ongoing link. CBS also defines a parameter called T_{poll} to set an upper bound to the time between two consecutive sniff events. If a link does not get a chance to transmit or listen for T_{poll} Bluetooth slots, an ongoing sniff event will yield its turn to the link. In order to reduce the number of piconet switch, CBS also defines a parameter called N_{switch_th} to lengthen the ongoing sniff events. Therefore, an ongoing sniff event will only yield its turn to a link's upcoming sniff slot if the upcoming link's credits

exceed the ongoing link's credits by N_{switch_th}.

Besides, CBS also re-distributes the credits among all credit links from a node in order to satisfy the max-min fairness [15]. If a POLL-NULL event exists on a credit link and the link has a positive credit number in its account, some of its credits will be equally distributed to other links. As a result, the re-distributed link has about the same number of credits as the link with the minimum credits.

Lastly, in order to reduce the wasted slots spending on unsuccessful sniff slots and POLL-NULL events, CBS uses the Adaptive Presence Point Density (APPD) scheme to change the T_{sniff} interval according to the traffic load. At each link's sniff slot that does not have data transmission, T_{sniff} of the link is doubled. When the value of T_{sniff} reaches the upper threshold, it will stop increasing.

The advantage of this scheme is that it can maintain a max-min link level fairness among all credit links by using the credit account, redistribution of credits, N_{switch_th}, T_{poll}, and APPD. Moreover, each node only has to manage its own traffic and does not have to exchange information with other nodes. Therefore, it can reduce the message overhead.

However, one of the problems of the CBS scheme is that it does not consider bridge node conflict. In CBS scheme, it is possible for two masters to poll the same bridge node at the same sniff slot; therefore, it wastes a number of time slots due to the conflict. In addition, the packet loss event also degrades the performance of TCP traffic. Moreover, the ADDP method of doubling the T_{sniff} interval may not be fast enough to react the traffic change. A further analysis on how to modify the T_{sniff} value is necessary.

2.2.2.2 Pseudo-Random Coordinated Scatternet Scheduling (PCSS)

PCSS [16] is another scheme that does not have a fixed switching piconet schedule for a

bridge node. Each node in the scatternet maintains its own schedule. In this scheme, it defines *checkpoint* as the meeting point for two connected nodes to exchange data, and it defines *checking interval*, T_{check}, as the period of time that contains exactly one checkpoint. It also defines *checking intensity* as the inverse of checking interval. In PCSS, a pair of nodes starts communicating with each other at a checkpoint; however, the communication ends when one of them leaves to attend a checkpoint for another node. In PCSS, each master-slave pair creates its own pseudo random sequence of checkpoints. As the pseudo random sequence is unique for each master-slave pair, it can prevent systematic conflict of checkpoints on different links to a bridge node. In PCSS, a master-slave pair uses the current master clock, the MAC address of slave, and the current T_{check} to generate pseudo random sequence of checkpoints.

In order to avoid slots wastage, PCSS changes the checkpoint intensity according to data traffic. Each node in PCSS maintains its own traffic measurement and independently changes the checkpoint intensity according to the traffic measurement. Therefore, the generated checking interval for each of the two connected nodes may be different. In order to synchronize the checkpoint positions between two connected nodes, the pseudo random checkpoint positions for the node with lower checking intensity must be a subset of the pseudo random checkpoint positions for the node with higher checking intensity. PCSS uses the utilization of checkpoints on each link and the total utilization on the node to determine whether to increase or decrease the checking intensity on a link. A checkpoint is considered to be *utilized* if both the master and the slave use the checkpoint and at least one data packet has been sending out from either the master or the slave. In PCSS, each node takes N_{sample_min} sample of measurements before deciding to change the checking intensity. If the utilization

of checkpoints on a link drops below the lower checkpoint utilization threshold, PCSS will decrease the checking intensity by doubling the checking interval. However, if the utilization of checkpoints on a link rises above the upper checkpoint utilization threshold and the total utilization on the node is still lower than the upper node utilization threshold, PCSS will increase the checkpoint intensity by halving the current checking interval. The checking of total utilization on a node is used to set a maximum limit for the checking intensity.

The advantage of this scheme is that the scheduling is localized on each node; therefore, it is easy to apply the PCSS scheme to the scatternet. Moreover, the uniqueness of pseudo random sequence of checkpoint positions on each link can reduce the chance of having bridge node conflict. Furthermore, the ability of changing checking intensity can also reduce wasted slots spending on low traffic link. Although there are many advantages from the PCSS scheme, there are also some weaknesses on this scheme. Sometimes if a checkpoint on a link is very close to a checkpoint on another link, the master may need to switch to the latter link while there is still a packet transmitting on the current link. Therefore, a maximum of six slots, including a 5-slot data packet and 1-slot polling packet, will be wasted. In addition, since the serving time for each node is bounded by the time between the current checkpoint for the serving node and the next checkpoint for another node, it does not provide a deterministic serving time for each node. PCSS does not maintain fairness for all nodes within the scatternet.

2.3 Summary

In this chapter, we summarized various intra-piconet and inter-piconet scheduling algorithms proposed in the literature. For each algorithm, we identified both the advantages and

limitations. For intra-piconet scheduling, most of the previous work focused on improving performance on throughput and delay, as well as maintaining the fairness for all the nodes. For inter-piconet scheduling (or scatternet scheduling), most of the previous work focused on one (or more) of the following issues: determine the meeting time between master and bridge node, prevent the bridge node conflict, maintain fairness for all nodes in a scatternet, and allocate bandwidth to each links based on traffic change. Our scatternet scheduling algorithm (to be described in Chapter 3) aims to satisfy all the above requirements.

Chapter 3 – Adaptive Scheduling Algorithm

In this chapter, we propose an adaptive scheduling algorithm (ASA) for Bluetooth scatternets. The ASA aims to achieve the max-min fairness by dynamically allocating bandwidth to each link based on real-time traffic. Moreover, it also aims to prevent the bridge node conflict in order to reduce wasted slots. Lastly, the ASA integrates both intra-piconet scheduling and inter-piconet scheduling.

The rest of this chapter is organized as follows: A list of assumptions are stated and explained in Section 3.1. The traffic estimator, intra-piconet and inter-piconet scheduling algorithms are described in Sections 3.2. A discussion of how ASA can achieve the max-min fairness is given is Section 3.3. An example of the operation of ASA in a sample scatternet topology is described in Section 3.4

3.1 Assumptions

In our work, we make the following assumptions:

- A bridge node is a slave-slave node;
- Only ACL connections are present in the scatternet;
- All nodes are time-synchronized with each other within the scatternet

The rationale for the above assumptions are as follows: Results in [17] show that a slave-slave bridge node can achieve a lower transfer delay. When a master is also acting as a bridge node, it decreases the bandwidth utilization within a piconet. For performance point of view, a lot of previous research work (including this one) assumes the use of a slave-slave bridge node.

Our scheme does not consider the support of SCO links within the scatternet. When there is a SCO link in a piconet, there will only be four slots available for ACL connections. Therefore, a node can no longer use a 5-slot packet for data transmission. In addition, if there are two SCO links in a piconet, a node will only be able to use 1-slot packet for data transmission. Thus, supporting SCO link will decrease the throughput and delay for other ACL links. Besides, if the concerned SCO link is a master-bridge link, then there will be a problem on the inter-piconet scheduling. Since the switching of the bridge node between piconets can result in a maximum of two slots lost, there will only be two slots available for the bridge node to reach another piconet. With only 2-slots available, a bridge node is not able to connect with more than two piconets. Therefore, there are proposals regarding the use of an ACL link to handle SCO like traffic. The work in [21] proposed a method of replacing SCO traffic with a QoS-constrainted ACL traffic. The work in [22] showed that ACL traffic is capable for supporting voice connections.

We assume that time slots for all the nodes are perfectly aligned. The reason for assuming all nodes are time-synchronized with each other is to simplify the discussion in this chapter. Our scheme can also handle the case where there is time-slot loss for a bridge node to switch between different piconets.

3.2 Adaptive Scheduling Algorithm

In order to maintain fairness for all nodes within the scatternet, ASA defines the *maximum usable serving slots* (*MUSS*) to limit the serving time between two nodes. The size of MUSS depends on the types of Bluetooth packet supported by the scatternet. Moreover, the size of MUSS must be large enough for both nodes to exchange packets. Therefore, if the scatternet

supports 1-slot, 3-slots and 5-slots packets, then the size of MUSS must be larger than or equal to ten Bluetooth time slots. If the scatternet only supports 1-slot and 3-slots packets, then the size of MUSS must be larger than or equal to six Bluetooth time slots. Lastly, if the scatternet only supports 1-slot packet, then the size of MUSS must be larger than or equal to two Bluetooth time slots.

In order to prevent bridge node conflict and at the same time to provide the max-min fairness for all the links within the scatternet, each master node or bridge node maintains a *dynamic switch schedule* to organize the time for it to communicate with each connected node. Figure 3.1 shows the structure of a switch schedule. A switch schedule divides the timeline into fixed size *switch schedule slots* (*SS_Slots*). A master reserves the serving time for a bridge node by assigning an SS_Slot to the bridge node. A master reserves the serving time for a slave node by using an empty SS_Slot. On the other hand, a bridge node reserves the serving time for a master node by assigning an SS_Slot to the master. Since ASA uses the switch schedule to reserve serving time between two nodes, the size of SS_Slot is equal to MUSS.

In ASA, when a master or a bridge node encounters an SS_Slot which has been assigned to a link, it spends a maximum of MUSS on the link. However, when a master or a bridge node encounters an SS_Slot which has not been assigned to any links, the master and the bridge node handle it differently. The master will use the time duration of the SS_Slot to communicate with its slaves following the intra-piconet scheduling. On the other hand, the bridge node will become idle during the entire SS_Slot. The switch schedule will be updated every time when the master node meets with the bridge node. Other than maintaining a switch schedule, a master node also maintains *active* and *waiting lists* to schedule the serving

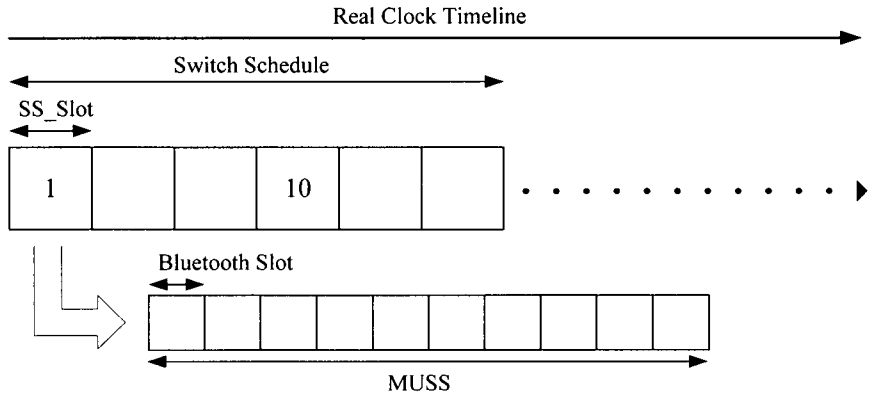

Figure 3.1. Structure of a switch schedule

time for each slave node. Furthermore, an estimator is placed on each node to monitor the traffic condition.

3.2.1 Traffic Estimation

In order to monitor the traffic change, each node is responsible to update the traffic rate either when it receives a packet from the application layer or when it bypasses a packet to another node. The process continues until the node leaves the scatternet. In ASA, it uses the time-sliding window (TSW) [23][24] for traffic estimation. We also modify the time to re-start the estimation in case of a long idle period. The reason for using TSW is that it maintains a time-based history for the estimation and decays the past history information over time, but not over packet arrivals. Therefore, a high traffic stream or a low traffic stream has the same weight on updating the estimated traffic rate. For example, if a node encounters bursty traffic, the estimated packet arrival rate will not be equal to the packet arrival rate for the bursty traffic. Since TSW monitors a window size of history, it considers all the packets arrived during the window length. As a result, it can smooth out the traffic estimation under bursty traffic. Our scheme does not focus on improving the traffic estimation scheme since

```
avg_interval: time window over which history is kept (constant)
avg_rate: measured arrival rate of traffic in unit of slot
restart_ratio: ratio difference which triggers the restart of estimation (constant)
stage: indicate the stage of traffic estimation and is initially set to 0
previous_t: store the arrival time of previous arriving packet
inter_arrival_t: current packet inter-arrival time
packet_slots: number of slots for current arriving packet
slots_in_win: number of slots in time window
new_slots: combined size of slots for current packet and slots_in_win

packet_arrived_from_upper_layer( ) {
        if (start equals 2)
                if ( inter_arrival_t × restart_ratio  < ( current time – previous_t ) )
                reset start to 0;

        if (start equals 0)
                previous_t = current time;
                set start to 1;
        else if (start equals 1)
                inter_arrival_t = current time – previous_t;
                avg_rate = packet_slots / inter_arrival_t;
                previous_t = current time;
                set start to 2;
        else if (start equals 2)
                inter_arrival_t = current time – previous_t;
                slots_in_win = avg_rate × avg_interval;
                new_slots = slots_in_win + packet_slots;
                avg_rate = new_slots / ( inter_arrival_t + avg_interval );
                previous_t = current time;
}
```

Figure 3.2. Algorithm for traffic rate estimation

our main concern is to obtain the estimated traffic information for creating the switch schedules and active lists.

Figure 3.2 shows the algorithm for the traffic estimation. In this algorithm, it takes the first two packets to calculate the initial packet arrival rate. It then updates the estimated packet arrival rate by using the TSW scheme. As the prediction of packet arrival rate is based on the reason that the future packet arrival time should be related to the current packet arrival time, a packet arrives after a long idle period will break this kind of relationship. Therefore, the traffic estimator restarts the estimation when the new packet inter-arrival time is larger

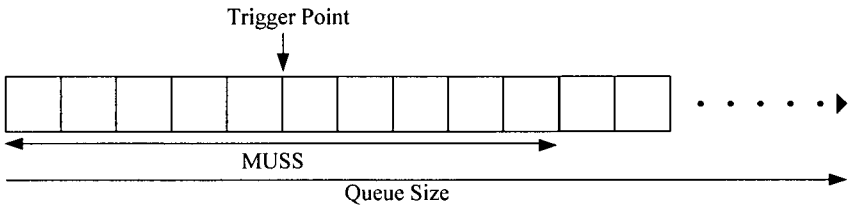

Figure 3.3. Relation between MUSS and Trigger Value

than the previous packet inter-arrival time by a multiple of *restart_ratio* which has a predefined value. The value of *restart_ratio* in ASA is set to 5.

Besides estimating the packet arrival rate, each node also determines the time for it to accumulate enough packets to request for data transmission. As mentioned in the previous section, a master only serves a connected node for no more than MUSS. Therefore, in order to allow both nodes to receive equal bandwidth for data transmission, ASA sets the *trigger point* for requesting data transmission to be half the size of MUSS. Figure 3.3 shows the relationship between MUSS and the trigger point. In ASA, the actual time for a node to reach the trigger point is defined as the *transmission request arrival time* (*TRAT*).

Figure 3.4 shows the algorithm for how each node updates the estimated TRAT before it transmits a packet. If a node already has enough packets in the queue, it will set TRAT to 0 in order to indicate that no waiting time is needed. Otherwise, it will use the estimated packet arrival rate to predict the value of TRAT. As a result, if a node already has enough packets in the queue, ASA does not require the traffic estimation information to determine the status of the node. Consequently, ASA does not always depend on traffic estimation.

If a node's queue is empty for a long period of time, which is referred as an idle period, the estimated packet arrival rate may not give enough information to predict the TRAT. As a result, the node may request for data transmission while it does not have any

```
avg_rate: measured arrival rate of traffic from the estimator in unit of Bluetooth slot
slots_in_queue: number of packet slots in the queue
slot_need: required accumulated number of slots to reach the trigger point (constant)
max_waiting_interval: the maximum waiting period in unit of second (constant)
pre_empty_q: indicate whether the queue is empty for previous round and is initially set to false
est_req_t: estimated duration time for the node to reach the trigger point in unit of second
est_trat: estimated TRAT in unit of second
skipping_time: keep track the multiplying factor for est_req_t and is initially set to 1

Note: If est_req_t and est_trat are set to 0, it indicates that the node had already
accumulated enough packets to reach the trigger point

require_to_send_data ( ) {
        if ( the queue is empty and pre_empty_q is true )
                if (current time ≥ est_trat)
                        skipping_time++;
                        est_req_t = slot_need / avg_rate;
                        if (est_req_t × skipping_time > max_waiting_interval)
                                est_req_t = max_waiting_interval;
                        else
                                est_req_t = est_req_t × skipping_time;
                        est_trat = est_trat + est_req_t;
        else if ( the queue is empty)
                set pre_empty_q to true;
                est_req_t = slot_need / avg_rate;
                est_trat = current time + est_req_t;
        else if (slots_in_queue < slot_need)
                est_req_t = (slot_need – slots_in_queue ) / avg_rate;
                est_trat = current time + est_req_t;
        else
                reset skipping_time to 1;
                est_req_t = 0;
                est_trat = 0;
        include the value of est_trat in the header of the packet
}
```

Figure 3.4. Algorithm for trigger point estimation

packet to send. Therefore, some time slots will be wasted during the idle period. In order to reduce the number of wasted slots in the above situation, ASA maintains a multiplying factor referred as the *skipping_time* to modify the waiting time. For each consecutive occurrence of an empty queue, if the current time has already passed the estimated TRAT of the node, the node will increase the value of the *skipping_time* by one. Besides, in order to avoid a master

skips polling a node for a long period of time, ASA defines the *maximum waiting interval* to limit the waiting time for the node to request for data transmission. Therefore, if the value of *skipping_time* multiplies the original estimated duration time required for the node to reach the trigger point is smaller than the *maximum waiting interval* for the node, it will be set as the next duration time for the node to reach TRAT. Otherwise, the *maximum waiting interval* will be set as the next duration time for the node to reach TRAT. Once the node accumulates enough packets to reach the trigger point, ASA re-sets the value of *skipping_time* to 1. In ASA, the value of the *maximum waiting interval* is set to be the time duration for 100 Bluetooth slots.

Since it is the responsibility for the master to update the nodes' status in the active and waiting lists, a slave node needs to include the actual TRAT in the header of the packet sending to the master. If we utilize a 64-slot wrap around switch schedule to indicate the TRAT, it will use 6 bits in the header for the information. Furthermore, since it is the responsibility for the bridge node to decide a confirmed meeting time on the inter-piconet link, a master needs to include the actual TRAT in the header of the packet sending to the bridge node. This information is important for maintaining fairness for nodes in a scatternet and allocating bandwidth to each link based on traffic change.

3.2.2 Intra-piconet Scheduling

In ASA, each master node maintains an *active list* and a *waiting list* to schedule the serving order for all connected slave nodes. An *active list* contains all the slave nodes that have accumulated enough packets to reach the trigger point. A *waiting list* contains all the slave nodes that do not have enough packets in the queue. The master follows the order of nodes in

its active list and serves the nodes in a round robin fashion. Furthermore, a master node starts to serve the next slave node either when it has finished serving the slave node for MUSS or when there is no data to send between the two nodes. As mentioned in Section 3.2, a master uses the dynamic switch schedule to reserve an SS_Slot, which is equal to the size of MUSS, for the communication between two nodes. However, sometimes two nodes may spend less than MUSS for data transmission. In that case, there will be some slots available before the master encounters an SS_Slot assigned to a bridge node. The master will continue to serve next node in the active list until the time for the master to communicate with a bridge node occurs. The master then switches to serve the bridge node. Besides, before the master serves another node, it will update the status of the slave node in the active list.

The algorithm for how each master node updates the active and waiting lists is shown in Figure 3.5. At the time when the master encounters an empty SS_Slot, it begins to serve the slave node in the active list. Each time when a slave node replies the packet to the master, it indicates its estimated TRAT in the packet's header. However, the master will only use the information to update the active list when it switches to serve another node. Therefore, at that time, if either a master or a slave node indicates that it has already accumulated enough packets to reach the trigger point, the master will keep the slave node in active list. However, if both nodes indicate that they do not have enough packets, the master will move the slave node to the waiting list. Simultaneously, the master also stores the estimated time for it to poll the slave node again. The estimated polling time is chosen from the smaller time between the estimated TRAT for the master and the estimated TRAT for the slave node. If the master reaches the end of the active list, it will select a slave node which has the smallest estimated polling time and has not been polled for this round in the waiting list for a test.

```
active_list:  a list contains all the nodes which are ready for polling
waiting_list:  a list contains all the nodes which are not ready for polling
est_s_trat:  estimated TRAT for the current serving  slave
est_m_trat:  estimated TRAT for the master on the current serving slave
est_poll_t[7]:  estimated time for the master to poll each slave node
pointer:  it indicates the current serving node in the active list

Note:   when est_s_trat equals 0, it indicates that the current slave node has already
        accumulated enough packets to reach the trigger point
        when est_m_trat equals 0, it indicates that the master has already accumulated
        enough packets to reach the trigger point for the current slave node

update_ list( ) {
        if ( either est_s_trat or est_m_trat is 0 )
                keep the current node on the active list;
                move the pointer to the next node in the active list;
        else
                remove the node from active list;
                add the node to waiting list;
                if ( est_m_trat < est_s_trat)
                        est_poll_t[current node] = est_m_tratt;
                        move the pointer to the next node in the active list;
                else
                        est_poll_t[current node] = est_s_trat;
                        move the pointer to the next node in the active list;

        if ( it is the end of active list )
                 select the node ⇒ with smallest est_poll_t[current node] in waiting list and
                                   has not been polled for this round;
                if ( est_poll_t[current node] for the node ≤ current time )
                        remove the node from waiting list;
                        add the node to the end of active list;
                else
                        move the pointer to the front of active list;
}
```

Figure 3.5. Algorithm for updating active and waiting list

If the current time has already passed the estimated polling time for the chosen slave node, the master will remove the node from the waiting list and add it back to the end of the active list. The master then serves the new slave node in the active list. The process continues until the master cannot find a node in the waiting list that is capable of moving to the active list. The master will then move to the next round and serve the first slave node in the updated

active list.

3.2.3 Inter-piconet Scheduling

A *switch schedule* is maintained between a master and a bridge node to organize the time for them to communicate with each other. Each time when a master meets with a bridge node, they negotiate their *next meeting time* and update their switch schedules. In ASA, a master uses the hold mode to allow the bridge node to switch between piconets. In hold mode, a master puts a connected node into sleep state in which the node does not require to listen to the master for a period of time. When the time expires, the connected node turns back into an active mode and actively listens to the master. Therefore, after determining the new meeting time, a master node knows how long it should put a bridge node into the hold mode. In ASA, the master initializes the negotiation process.

Figure 3.6 shows the algorithm for how the master node determines the *suggested next meeting time*. The master node first finds an empty SS_Slot as the initial estimated meeting time. Then for each slave node in the active list, the master reserves an empty SS_Slot for it to indicate that the SS_Slot is not eligible for scheduling with a master-bridge meeting time. Moreover, the master will skip the SS_Slot that has already been assigned to a link. The process continues until the master reaches the end of the active list.

After reserving SS_Slots for the active nodes, the master also checks the nodes in the waiting list. Thus, starting from the node with the earliest estimated polling time in the waiting list, if the start time for current estimated meeting SS_Slots is larger than the node's estimated polling time, the master will reserve an empty SS_Slot for the node. The process will not end until the master finishes checking all the nodes in the waiting list. As a result, the

```
current_m_slot: the current master SS_Slot in unit of Bluetooth slot
current_m_slot_t: the start time of current master SS_Slot in unit of second
suggested_m_slot: the suggested master meeting SS_Slot in unit of SS_Slot
suggested_m_slot_t: the start time of suggested master meeting SS_Slot in unit of second
SS_Slot_time: the time duration of one SS_Slot in unit of second (constant)
est_poll_t[7]: the estimated time for the master node to poll the slave node in unit of second

obtain_next_meeting_slot( ) {
        suggested_m_slot = current_m_slot + one SS_Slot;
        suggested_m_slot_t = current_m_slot_t + SS_Slot_time;
        while ( suggested_m_slot is already assigned )
                suggested_m_slot = suggested_m_slot + one SS_Slot;
                suggested_m_slot_t = suggested_m_slot_t + SS_Slot_time;
        for ( each node in the active list )
                suggested_m_slot = suggested_m_slot + one SS_Slot;
                suggested_m_slot_t = suggested_m_slot_t + SS_Slot_time;
                while ( suggested_m_slot is already assigned )
                        suggested_m_slot = suggested_m_slot + one SS_Slot;
                        suggested_m_slot_t = suggested_m_slot_t + SS_Slot_time;
        for ( each node in waiting list with ascending order of est_poll_t[current node]) {
                if (est_poll_t[current node] of a node ≤ suggested_m_slot_t ) {
                        suggested_m_slot = suggested_m_slot + one SS_Slot;
                        suggested_m_slot_t = suggested_m_slot_t + SS_Slot_time;
                        while ( suggested_m_slot is already assigned )
                                suggested_m_slot = suggested_m_slot + one SS_Slot;
                                suggested_m_slot_t = suggested_m_slot_t + SS_Slot_time;
}
```

Figure 3.6. The algorithm for obtaining master suggested meeting time

master can determine the final suggested meeting time with the bridge node, which is located at the start time of the estimated meeting SS_Slot. By the whole process, the master can ensure that it reserves enough serving time between the current meeting time and the next meeting time for other slave nodes and bridge nodes within the piconet.

Thus, each time when a master node encounters an SS_Slot which is assigned to an inter-piconet link, the master will obtain the next suggested meeting SS_Slot with the bridge node by using the algorithm shown in Figure 3.6. Moreover, it will also obtain the TRAT by using the algorithm shown in Figure 3.4. Lastly, it will determine all the following SS_Slots which have already been assigned to other bridge nodes. The master will then include all the

above information in header of the packet sending to the targeted bridge node in order to negotiate for the next meeting time. In addition, the master will switch to serve another node either when it has finished serving the bridge node for MUSS or when there is no data to send between the two nodes. If we assume that a master can connect up to three bridge nodes and a master utilizes a 64-slot wrap around switch schedule, it will use 18 bits for including the suggested SS_Slot and the occupied SS_Slots in the packet.

Once the bridge node has received the information from the master, it uses the information together with its local estimated values to determine the *confirmed meeting time*. Figure 3.7 shows the algorithm for the process. At the beginning, the bridge node finds the time for the nodes to fulfill the trigger point requirement. Thus, if both nodes have already reached the trigger point, the bridge node will indicate that no waiting time is needed. However, if both nodes indicate that they do not have enough packets to reach the trigger point, the bridge node will select the smaller estimated TRAT between the master and the bridge node as the final estimated TRAT. Afterwards, the bridge node finds the closest SS_Slot with a start time exceeding both the final estimated TRAT and the start time of meeting SS_Slot suggested by the master. If the SS_Slot has already been assigned to a link, the bridge node will find the next closest empty SS_Slot. Lastly, the bridge node assigns the empty SS_Slot to the master node, and selects the start time of the SS_Slot as the confirmed meeting time. The bridge node then includes the confirmed meeting SS_Slot in header of the packet replying to the master. The master then assigns the confirmed meeting SS_Slot to the bridge node.

```
current_b_slot: the current bridge SS_Slot in unit of Bluetooth slot
current_b_slot_t: the start time of current bridge SS_Slot in unit of second
suggested_m_slot: the suggested master meeting SS_Slot in unit of SS_Slot
suggested_m_slot_t: the start time of suggested master meeting SS_Slot in unit of second
confirm_b_slot: the confirm bridge meeting SS_Slot in unit of SS_Slot
confirm_b_slot_t: the start time of confirm bridge meeting SS_Slot in unit of second
SS_Slot_time: the time duration of one SS_Slot in unit of second (constant)
est_b_trat: estimated TRAT for the bridge node
est_m_trat: estimated TRAT for the master
est_poll_t: estimated time for master to poll the bridge node

Note: when est_poll_t is set to 0, it indicates that no waiting time is need to request
      for data transmission

obtain_confirm_meeting_slot( ) {
        if ( est_b_trat equals 0 or est_m_trat equals 0 )
                est_poll_t = 0;
        else if ( est_b_trat < est_m_trat)
                est_poll_t = est_b_trat;
        else
                est_poll_t = est_m_trat;

        confirm_b_slot  = current_b_slot + one SS_Slot;
        confirm_b_slot_t = current_b_slot_t + SS_Slot_time;
        while (confirm_b_slot_t < suggested_m_slot_t or confirm_b_slot_t < est_poll_t )
                confirm_b_slot = confirm_b_slot + one SS_Slot;
                confirm_b_slot_t = confirm_b_slot_t + SS_Slot_time;
        while (confirm_b_slot is already assigned either in master or bridge)
                confirm_b_slot = confirm_b_slot + one SS_Slot;
                confirm_b_slot_t = confirm_b_slot_t + SS_Slot_time;
}
```

Figure 3.7. The algorithm for obtaining bridge confirmed meeting time

3.2.4 Resume Send

Since a master node only spends an MUSS on a connected node, sometimes the Bluetooth slots left between the two nodes may not be enough for the size of packet exchange on the current serving link. Moreover, as the master must start serving a bridge node when the scheduled meeting time has arrived, sometimes the Bluetooth slots left before a master switching to serve a bridge node may also not be enough for size of packet exchange on the current serving link.

Our scheme implements the *resume send mode* in order to handle this situation. In ASA, the master updates the connected node (i.e., slave) with the remaining time-slots available by including the value in the unused 4-bit header field of the packet to the node. On the other hand, the slave updates the master with the size of packet waiting in its queue by including the value in the 1-bit header field of the packet to the master.

Figure 3.8 shows the algorithm of how the master node and the connected node maintain the status between two nodes. In the algorithm, each time before a master sends a packet, it first checks whether the connected node has asked for a resume send in the previous turn. If the connected node has already activated the resume send mode, the master will give the turn to the connected node by sending a POLL packet. The master will de-activate the resume send mode after it receives the data packet from the connected node. As a result, it can maintain an equal chance for both nodes to utilize the bandwidth. If the resume send mode is off, the master will then check whether the Bluetooth slots available are large enough for the size of packet in the queue. If the remaining slots are not enough for the packet transmission in its queue, but are enough for packet transmission in the connected node's queue, the master again will give the turn to the connected node by sending a POLL packet. Therefore, it can reduce the number of wasted slots. Lastly, if the remaining slots are not enough for both nodes, the master will switch to serve the next node. Alternatively, if the connected node notices that the remaining slots are not enough for packet transmissions in its queue, it will indicate the activation of resume send mode in a NULL packet and send the packet to the master.

```
current_m_packet_slot:  number of slots for the current packet in master's queue
current_n_packet_slot:  number of slots for the current packet in node's queue
expect_s_packet_slot:  number of slots for the next packet in slave's queue
slot_left:  number of slots left for transaction
is_resume:  indicate whether the resume send mode is on or off

master_send_check( ) {
        if ( is_resume is on )
                update slot_left and indicate in the packet;
                send POLL packet;
        else if ( current_m_packet_slot < slot_left )
                update slot_left and indicate in the packet;
                normal send;
        else if ( expect_s_packet_slot < slot_left and current_m_packet_slot > slot_left )
                update slot_left and indicate in the packet;
                send POLL packet;
        else
                give the turn to the next possible node;
}

node_send_check( ) {
        if (current_n_packet_slot < slot_left )
                indicate the size of next packet waiting in the queue in the data packet;
                normal send;
        else
                activate resume send mode and indicate in the NULL packet;
                indicate the size of next packet waiting in the queue in the NULL packet;
                send NULL packet;
}
```

Figure 3.8. The algorithm for updating node status

3.3 Fairness Discussion

In this section, we describe how ASA can achieve the max-min fairness for all nodes in a scatternet.

3.3.1 Fairness around the Master Node

ASA can maintain the max-min fairness at the master node based on the following reasons. Once a link between a slave and the master has accumulated enough packets to reach the trigger point, the slave node is guaranteed to be in the active list in the current round and to

be served by the master with MUSS once in a around. In addition, in each round, the maximum number of links served by a master is equal to the total number of links connected to the master. Therefore, if a link generates traffic at a rate lower than or equal to the equal shared bandwidth, the link will not demand for more than one MUSS in a round. As a result, the link will always be satisfied. In addition, links with the same traffic rate will be added to the active list with the same number of times. As a result, they will receive the same amount of bandwidth.

If a link does not accumulate enough packets to reach the trigger point, it will be removed from the active list. Therefore, it will reduce the number of nodes in the active list and the duration time for each round will be shorter. As a result, the master will serve other links more frequently.

When links with higher rate and links with lower rate are present in the active list, they will receive the same amount of serving time from the master. Only when links with lower rate are removed from the active list, the links with higher rate will be able to use the residual bandwidth.

3.3.2 Fairness Around the Bridge Node

ASA achieves max-min fairness around a bridge node for the following reasons. By not serving the bridge node until the start time of confirmed meeting SS_Slot, it is analogous to placing the bridge node in the waiting list when it is not ready to receive service. The residual bandwidth will be re-allocated to other links around a bridge node by allowing them to reserve an SS_Slot before the confirmed meeting SS_Slot. Since a master serves a bridge node for MUSS at their confirmed meeting SS_Slot, it is similar to serving the node with

equal bandwidth when the node is added back to the active list. In addition, when a master suggests a meeting time, it maintains fairness around itself by reserving SS_Slots for nodes in the piconet. When a bridge node confirms the meeting time with the master, it maintains fairness around itself by ensuring the link will accumulate enough packets to reach the trigger point at the meeting time. Lastly, by skipping SS_Slot that has already been assigned to a link, ASA can prevent bridge node conflict within a scatternet.

3.4 Example

In order to demonstrate how the process works, we consider the topology shown in Figure 3.9. In this example, we assume that there are bi-directional traffic between the master node and the connected node. Moreover, all nodes are saturated senders (i.e., the nodes always have packet to send). In Figure 3.9, it also shows a switch schedule in which each slot represent an SS_Slot. Furthermore, in this example, M1 first meets with B1 at SS_Slot 1, M4 first meets with B1 at SS_Slot 2, and M2 first meets with B1 at SS_Slot 3. On the other hand, M3 first meets with B2 at SS_Slot 1, and M2 first meets with B2 at SS_Slot 2. Thus, we consider how the master node and bridge node update their first switch schedule. At SS_Slot 1, M1 meets with B1. Since there are two slave nodes connected to M1, M1 reserves the next two SS_Slots for intra-piconet scheduling. It then indicates that SS_Slot 4 is the next available slot to B1. Since SS_Slot 4 is also available for B1; therefore, B1 and M1 assign SS_Slot 4 as their next meeting time. On the other hand, at SS_Slot 1, M3 also meets with B2. Since there is one slave node connects to M3, M3 reserves the next SS_Slot for intra-piconet scheduling. It then indicates that SS_Slot 3 is the next available slot to B2. Since SS_Slot 3 is also available for B2; therefore, B2 and M3 assign SS_Slot 3 as their next

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
B1:	3	10	4	3	10	10	4	3	10	3	4	10	3	10	4	3	10	10	4	3	10	3	4	10	3
M1:	3			3				3		3			3			3				3		3			3
M2:		7	4			7	4			7	4			7	4			7	4			7	4		
B2:	8	7	8		8	7	8		8	7	8		8	7	8		8	7	8		8	7	8		8
M3:	8		8		8		8		8		8		8		8		8		8		8		8		8
M4:		10			10	10			10			10		10			10	10			10			10	

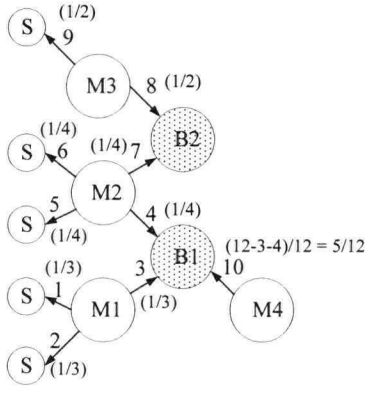

Figure 3.9. Scatternet topology and switch schedule

meeting time. At SS_Slot 2, M4 meets with B1. Since M4 does not connect with any slave node, it indicates that SS_Slot 3 is the next available slot to B1. However, since B1 has assigned SS_Slot 3 to link 4 (which is connected to M2), and SS_Slot 4 to link 3 (which is connected to M1), it then chooses SS_Slot 5 as the next meeting time. M4 and B2 then assign link 10 to SS_Slot 5. At SS_Slot 2, M2 meets with B2. As SS_Slot 3 has already been assigned to link 4, M2 cannot suggest it as the next meeting time.

Moreover, since there are two slave nodes connected to M2, M2 requires to reserve two SS_Slots for intra-piconet scheduling. Thus, M2 indicates that SS_Slot 6 is the next available slot to B2. Since SS_Slot 6 is also available for B2; therefore, B2 and M2 assign SS_Slot 6 as their next meeting time. Lastly, at SS_Slot 3, M2 meets with B1. Since there are two slave nodes connected to M2, M2 reserves the next two SS_Slots for intra-piconet scheduling. In addition, as SS_Slot 6 has already been assigned to link 7, it then indicates that SS_Slot 7 is the next available slot to B1. As SS_Slot 7 is also available to B1, B1 and

M2 then assign SS_Slot 7 as their next meeting time. Since all master nodes finish creating their first period of switch schedule, they can now setup hold mode with all its connected bridge nodes based on the switch schedule. After developing the first period of switch schedule, all master nodes and bridge nodes are able to update each period of switch schedule. Figure 3.9 shows the combined switch schedule for all master and bridge nodes until SS_Slot 25.

In order to determine the serving rate of each bridge node on the connected master nodes, we need to identify the switch schedule cycle by searching for the repeatable pattern in each bridge node. When considering the serving rate on B1, we identify that the repeatable pattern starts from SS_Slot 1 to SS_Slot 12 with three 4s, four 3s, and five 10s. The cycle length will then be 12 SS_Slots. Therefore, B1 serves M1 with 1/3 cycle, M2 with 1/4 cycle, and M4 with 5/12 cycle. On the other hand, when considering the serving rate on B2, we identify that the stable repeatable pattern starts from SS_Slot 1 to SS_Slot 4 with the sequence of one 7, and two 8s. The cycle length will then be 4 SS_Slots. Therefore, B2 serves M2 with 1/4 cycle, and M3 with 1/2 cycle. The reason for 1/4 cycle is not being used is that during the remaining 1/4 cycle, both M2 and M3 are busy serving on their pure slave nodes within their piconet. Therefore, by using the switch schedule on the master nodes and bridge nodes, it prevents the bridge node conflict and maintains the max-min fairness for all the links around a bridge node.

We can use the same method to find the serving rate of each master on its connected nodes. When considering the serving rate on M1, we identify that the repeatable pattern starts from SS_Slot 1 to SS_Slot 12 with four 3s, and eight empty SS_Slots. The cycle length will then be 12 SS_Slots. Therefore, M1 serves B1 with 1/3 cycle. Since the empty SS_Slots are

shared by two slaves nodes, M1 serves each slave with 1/3 cycle. When considering the serving rate on M2, we identify that the repeatable pattern starts from SS_Slot 2 to SS_Slot 5 with one 7, one 4, and two empty SS_Slots. The cycle length will then be 4 SS_Slots. Therefore, M2 serves B2 with 1/4 cycle, B1 with 1/4 cycle. Again, since there are two slave nodes sharing the empty SS_Slots, M2 serves each slave with 1/4 cycle. When considering the serving rate on M3, we identify that the repeatable pattern starts from SS_Slot 1 to SS_Slot 2 with one 8 and one empty SS_Slot. Therefore, M3 serves B2 with 1/2 cycle and slave with 1/2 cycle. Lastly, when considering the serving rate on M4, we identify that the repeatable pattern starts from SS_Slot 1 to SS_Slot 12 with five 10s and seven empty SS_Slots. Therefore, M4 serves B1 with 5/12 cycle. Thus, by reserving SS_Slots for intra-piconet scheduling and by updating the switch schedule at the master, it can maintain max-min fairness for all the links around a master node as well.

3.5 Summary

In this chapter, we proposed an adaptive scatternet scheduling scheme which is able to allocate bandwidth to each link based on traffic change, maintain fairness for all nodes in a scatternet, prevent bridge node conflict, and combine both inter-piconet and intra-piconet scheduling as a single design unit. By placing a traffic estimator and checking the size of queue on each node, ASA can estimate the current traffic condition. The master can utilize the traffic information to arrange the nodes in both active and waiting lists in order to fairly allocate bandwidth to all connected nodes. Moreover, as the master adds nodes to the active list according to their TRAT, it can reduce the average packet delay in the piconet. Besides, a master and the connected bridge nodes use the dynamic switch schedules to organize the time

for them to meet with each other. The master and the connected bridge node will update their switch schedules at the beginning of their meeting time. In the update process, the master maintains fairness for nodes within a piconet by reserving serving time for intra-piconet scheduling before it decides the suggested meeting time. On the other hand, the bridge node not only maintains fairness for nodes around itself by checking the TRAT for each connected link but also avoids the bridge node conflict by checking the reserved meeting time with other piconets before it decides the confirmed meeting time. Our proposed ASA integrates both intra-piconet and inter-piconet scheduling.

Chapter 4 – Performance Evaluation

In this chapter, we compare the performance of our proposed Adaptive Scheduling Algorithm (ASA) with Flexible Scatternet-wide Scheduling (FSS) [10] and Credit Based Scheduling (CBS) [13][14] through simulations. These two schemes are chosen because there are several features which are common to ASA, FSS, and CBS. Similar to FSS and CBS, ASA organizes the time for the master to meet with its bridge nodes, allocates bandwidth to each link based on the traffic conditions, and integrates both intra-piconet and inter-piconet scheduling. However, FSS does not maintain fairness for nodes in the scatternet and CBS cannot prevent the bridge node conflict. Different from these two schemes, ASA also includes these two issues as design requirements. In the last part of the chapter, we repeat some of the experiments in the paper of Fair and Traffic Dependent Scheduling Algorithm [19] on ASA for performance comparison.

The Bluetooth simulation model is developed from the network simulator (ns-2) [25]. The simulation model maps the traffic model from ns-2 with the customized medium access control models, which implement ASA, FSS, and CBS. For performance comparisons, we use the topology shown in Figure 4.1 and the time for all simulation runs is 60s. This scatternet topology has two slave-slave bridge nodes, five master nodes, and twenty pure slave nodes. This topology allows us to investigate how the masters and the bridge nodes fairly share the bandwidth on their connected links under different traffic conditions.

We first compare the fairness of all three schemes. Afterwards, we compare their relative performance of throughput and delay with User Datagram Protocol (UDP) traffic. The UDP packets are generated according to either Constant Bit Rate (CBR) or bursty on-off

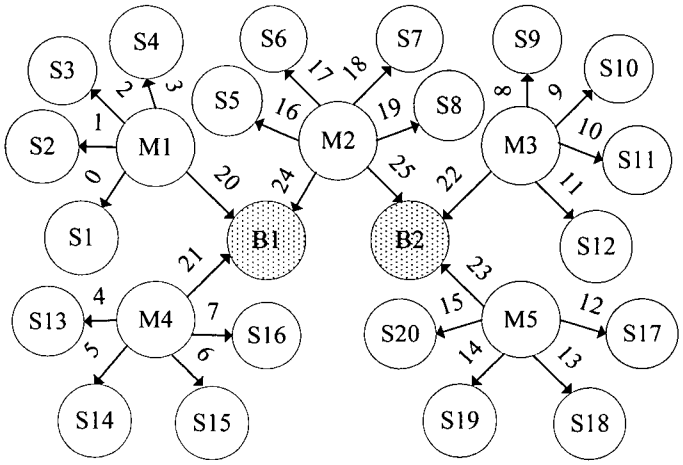

Figure 4.1. Topology of a scatternet

traffic models. Lastly, we compare the average end-to-end delay for transmitting a file from one node to another node with Transmission Control Protocol (TCP) traffic.

4.1 Fairness Comparison

In this section, we investigate whether or not each of the three schemes can achieve this max-min fairness for all the nodes within the scatternet. Three test cases are used to compare the performance between all three schemes. Throughout the test cases, we assume that there is only bi-directional traffic between two nodes and no traffic runs across a bridge node. Moreover, all traffic streams generate CBR UDP traffic and DH1 packets are used. In the first test case, we focus on a balance traffic load within a piconet. In the second test case, we focus on an unbalance traffic load around a master node. In the last case, we focus on an unbalance traffic load around a bridge node. In all test cases, we consider both situations in which there is packet loss due to interference and there is no packet loss in the ideal perfect environment. For simplicity, in ASA, we assume that when a packet loss occurs, the master and the bridge node still receive the information to schedule the next meeting time.

For all test cases, we use the same parameters stated in [10] for FSS. The parameters

α = 10, β = 5%, and len_q_{max} = 800. Each frame in the switch table is set to two Bluetooth slots. In [10], it does not mention the values for P_{max} and R_{max}. However, since we can duplicate the simulation results on [10] by setting P_{max} = 35 and R_{max} = 6, we use these two values for the simulation on FSS. For CBS, N_{switch_th} is set to 0. Since for value larger than 0, the chance of a master to meet with a bridge node will decrease and will lead to a very low throughput on all the inter-piconet links. The T_{sniff} value for links around M1, M4, M3, and M5 is set to 10, and the T_{sniff} value for links around M2 is set to 12. Lastly, T_{poll} is set to 100. For ASA, *SS_Slot* and *MUSS* are set to two Bluetooth slots. The values of *avg_interval* and *restart_ratio* in traffic estimator are set to 15s and 5 respectively. Lastly, the value of *waiting_interval* for trigger point estimation is set to the time duration for 100 Bluetooth slots.

4.1.1 Balance Traffic Load

In this experiment, all nodes within the piconet generate packets with the same traffic rate; thus, all links within the same piconet should receive the same amount of bandwidth. Table 4.1 shows the traffic pattern for the simulation. When we ignore the packet collisions due to interference from neighboring piconets, the *maximum aggregated throughput in each piconet* (*MATP*) is $27\times8/625\times10^{-6}$ kbps = 345.6 kbps. In Figure 4.1, since masters M1, M3, M4 and M5 have five links each, the ideal max-min bandwidth sharing for each link is 1/5×345.6 kbps = 69.12 kbps. Since master M2 has six links, the ideal max-min bandwidth sharing for each link is 1/6×345.6 kbps = 57.6 kbps. The simulation results from each scheme are compared with the ideal max-min fairness bandwidth allocation.

Figure 4.2 shows that ASA achieves the same result as with the ideal values when we do not consider packet loss. When there is packet loss due to interference, ASA achieves

Link	Flow	Traffic Rate (kbps)	Traffic Rate / MATP	Ideal Max-Min Shared Ratio	Ideal Shared Bandwidth (kbps)
0	M1<->S1	69.12	0.20	0.20	69.12
1	M1<->S2	69.12	0.20	0.20	69.12
2	M1<->S3	69.12	0.20	0.20	69.12
3	M1<->S4	69.12	0.20	0.20	69.12
4	M4<->S13	69.12	0.20	0.20	69.12
5	M4<->S14	69.12	0.20	0.20	69.12
6	M4<->S15	69.12	0.20	0.20	69.12
7	M4<->S16	69.12	0.20	0.20	69.12
8	M3<->S9	69.12	0.20	0.20	69.12
9	M3<->S10	69.12	0.20	0.20	69.12
10	M3<->S11	69.12	0.20	0.20	69.12
11	M3<->S12	69.12	0.20	0.20	69.12
12	M5<->S17	69.12	0.20	0.20	69.12
13	M5<->S18	69.12	0.20	0.20	69.12
14	M5<->S19	69.12	0.20	0.20	69.12
15	M5<->S20	69.12	0.20	0.20	69.12
16	M2<->S5	57.6	0.166	0.166	57.6
17	M2<->S6	57.6	0.166	0.166	57.6
18	M2<->S7	57.6	0.166	0.166	57.6
19	M2<->S8	57.6	0.166	0.166	57.6
20	M1<->B1	69.12	0.20	0.20	69.12
21	M4<->B1	69.12	0.20	0.20	69.12
22	M3<->B2	69.12	0.20	0.20	69.12
23	M5<->B2	69.12	0.20	0.20	69.12
24	M2<->B1	57.6	0.166	0.166	57.6
25	M2<->B2	57.6	0.166	0.166	57.6

Table 4.1. Traffic pattern with fixed rate

bandwidth slightly lower than the ideal values. On the other hand, in Figure 4.3, it shows that the FSS scheme can achieve close result as the ideal value on master-bridge links but not on master-slave links. Since FSS pre-empts on master-bridge traffic, it does not allocate fair amount of bandwidth to master- slave links.

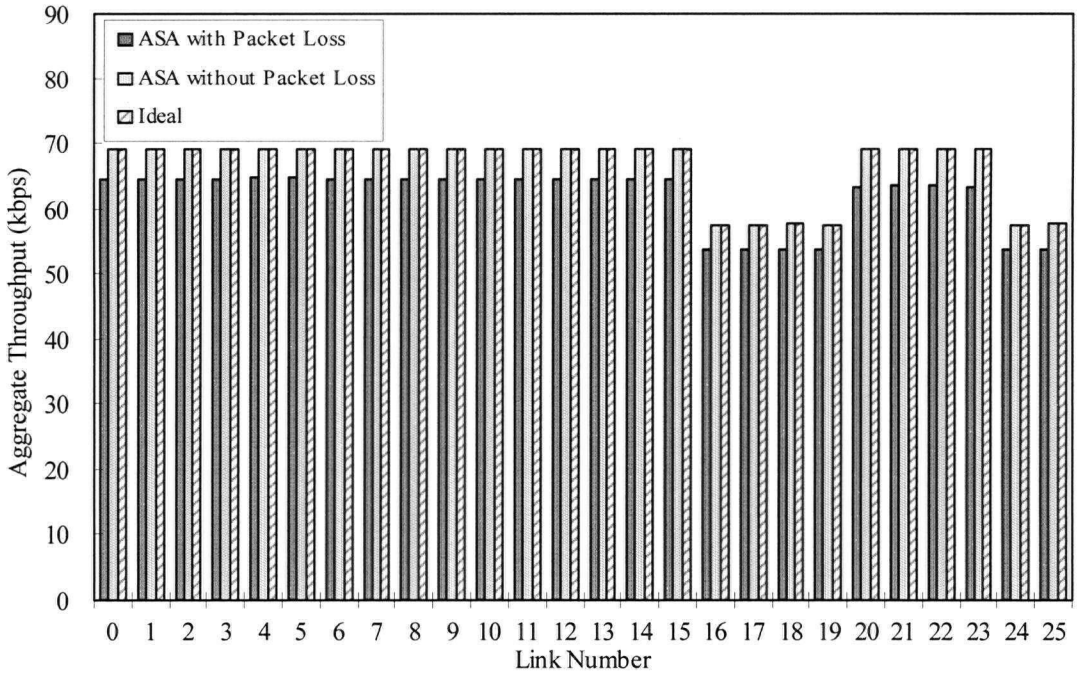

Figure 4.2. Fairness comparison for ASA in general

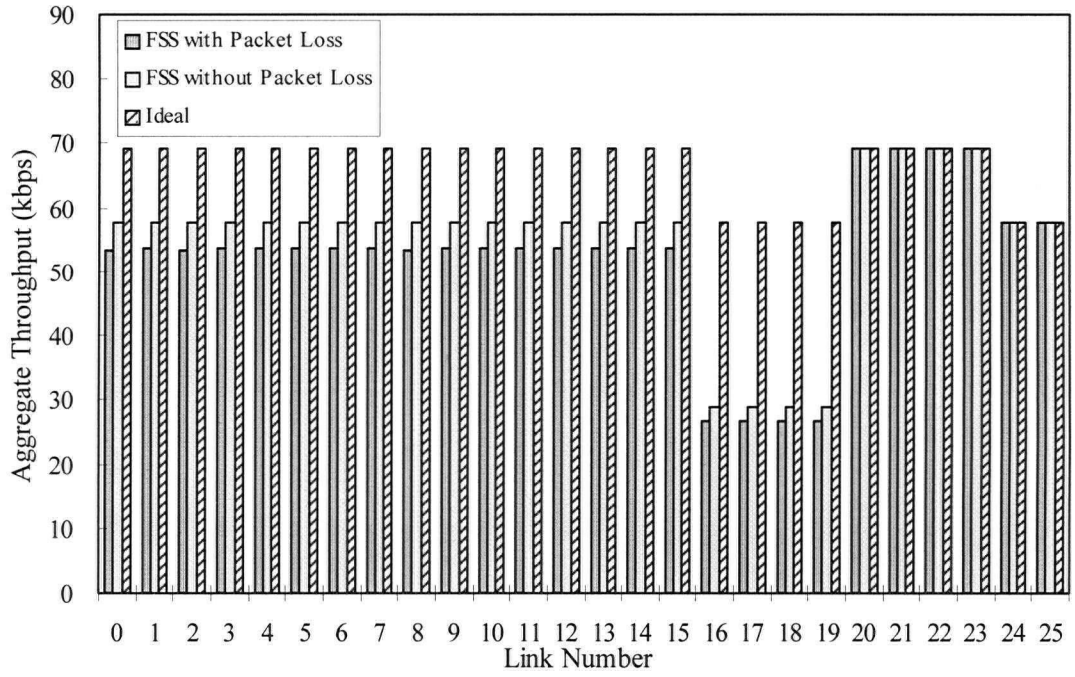

Figure 4.3. Fairness comparison for FSS in general

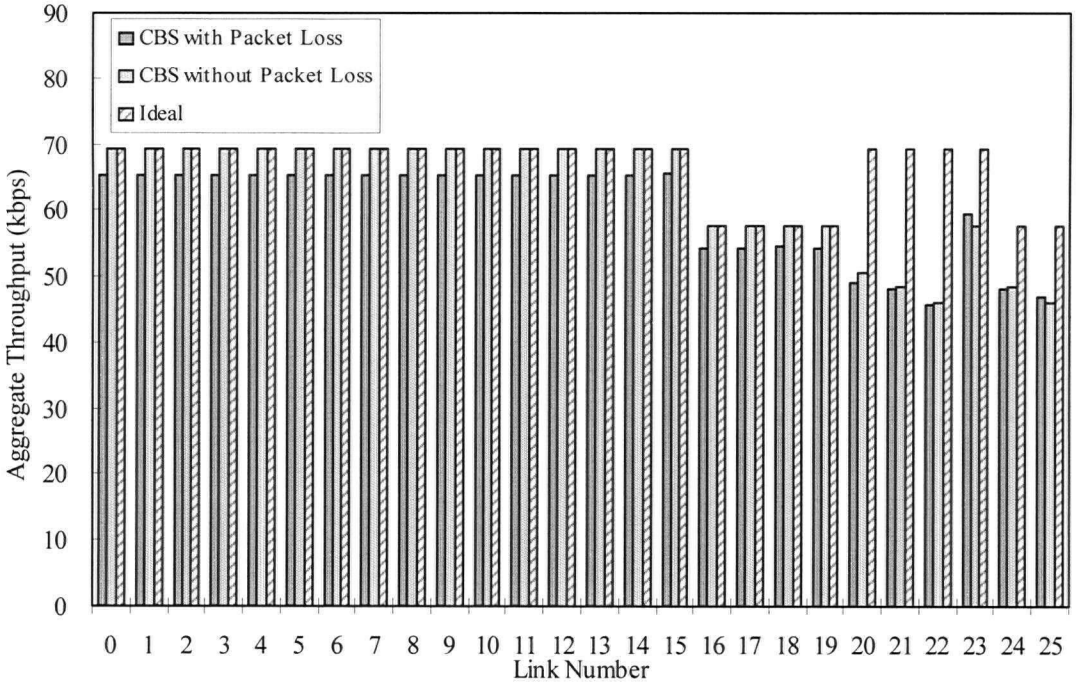

Figure 4.4. Fairness comparison for CBS in general

Lastly, in Figure 4.4, it shows that the CBS scheme can achieve the close result as the ideal value on master-slave links but not on master-bridge links. Since CBS does not prevent bridge node conflict, it leads to a lower bandwidth allocation on the master-bridge links.

4.1.2 Unbalance Traffic Load around a Master

In this experiment, we decrease the traffic rate on one link, and increase the rate on other links within a piconet. We investigate how each scheme re-allocates the bandwidth around the master node according to the max-min fairness criterion. Table 4.2 shows the traffic pattern. In the piconets of M1, M3, M4, and M5, each piconet has 3 links generating traffic at a rate of 0.225×MATP, the other 2 links are generating traffic at rate of 0.4×MATP and 0.1×MATP, respectively. Since one link only utilizes 0.1×MATP of bandwidth, which is lower than the equal bandwidth sharing, according to max-min fairness the remaining

Link	Flow	Traffic Rate (kbps)	Traffic Rate / MATP	Ideal Max-Min Shared Ratio	Ideal Shared Bandwidth (kbps)
0	M1<->S1	77.76	0.225	0.225	77.76
1	M1<->S2	77.76	0.225	0.225	77.76
2	M1<->S3	77.76	0.225	0.225	77.76
3	M1<->S4	34.56	0.1	0.1	34.56
4	M4<->S13	77.76	0.225	0.225	77.76
5	M4<->S14	77.76	0.225	0.225	77.76
6	M4<->S15	77.76	0.225	0.225	77.76
7	M4<->S16	34.56	0.1	0.1	34.56
8	M3<->S9	77.76	0.225	0.225	77.76
9	M3<->S10	77.76	0.225	0.225	77.76
10	M3<->S11	77.76	0.225	0.225	77.76
11	M3<->S12	34.56	0.1	0.1	34.56
12	M5<->S17	77.76	0.225	0.225	77.76
13	M5<->S18	77.76	0.225	0.225	77.76
14	M5<->S19	77.76	0.225	0.225	77.76
15	M5<->S20	34.56	0.1	0.1	34.56
16	M2<->S5	57.6	0.166	0.166	57.6
17	M2<->S6	57.6	0.166	0.166	57.6
18	M2<->S7	57.6	0.166	0.166	57.6
19	M2<->S8	57.6	0.166	0.166	57.6
20	M1<->B1	138.24	0.4	0.225	77.76
21	M4<->B1	138.24	0.4	0.225	77.76
22	M3<->B2	138.24	0.4	0.225	77.76
23	M5<->B2	138.24	0.4	0.225	77.76
24	M2<->B1	57.6	0.166	0.166	57.6
25	M2<->B2	57.6	0.166	0.166	57.6

Table 4.2. Traffic pattern for testing bandwidth allocation around master

bandwidth should be distributed to other links. The ideal sharing bandwidth for the link will then be 0.1×MATP = 34.56 kbps. Since all other links generate traffic at a rate higher than the equal sharing bandwidth, the remaining bandwidth will be evenly redistributed to them. The ideal sharing bandwidth for each link will then be [0.2+(0.1/4)]×MATP = 0.225×MATP = 77.76 kbps. The traffic rate for all the links in piconet of M2 stays the same.

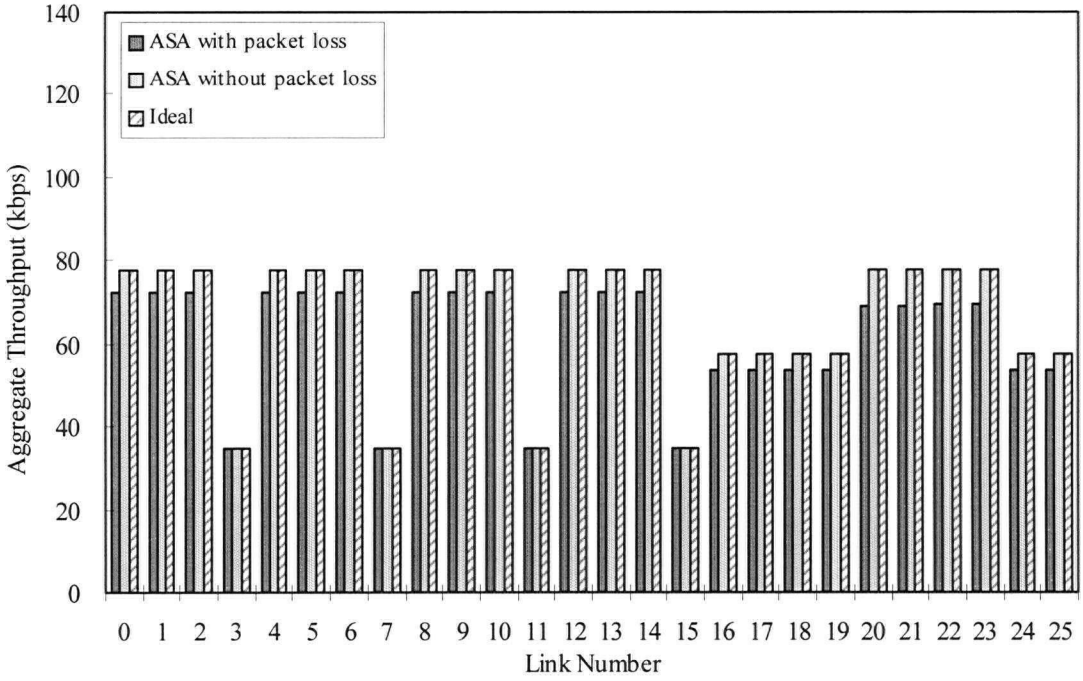

Figure 4.5. Fairness comparison for ASA on master

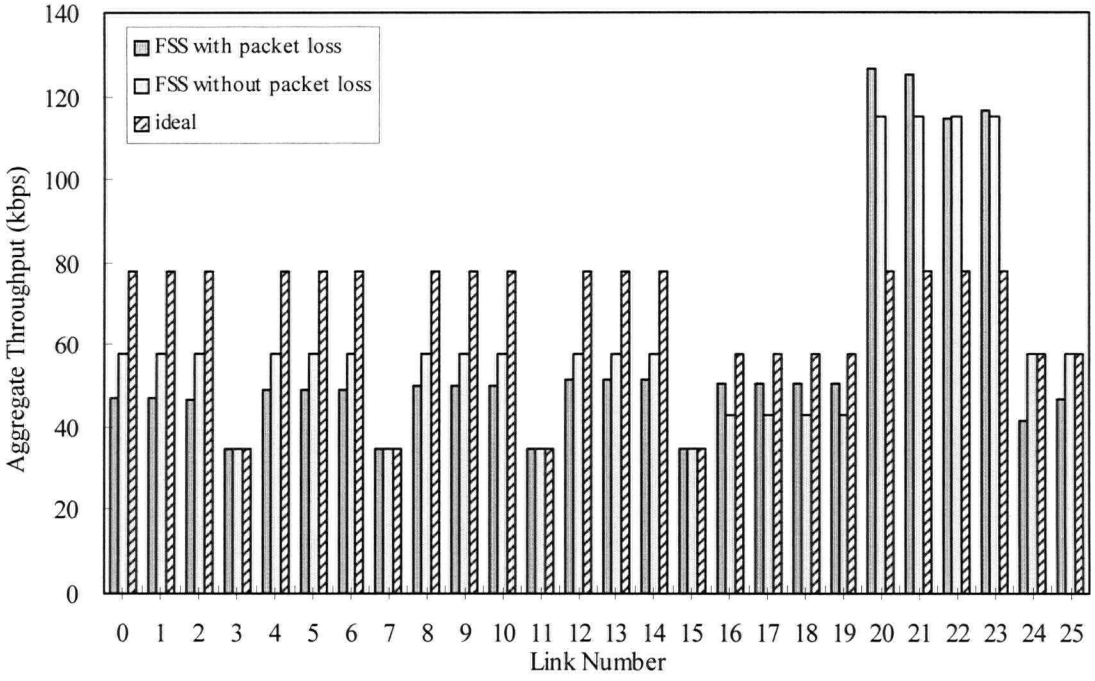

Figure 4.6. Fairness comparison for FSS on master

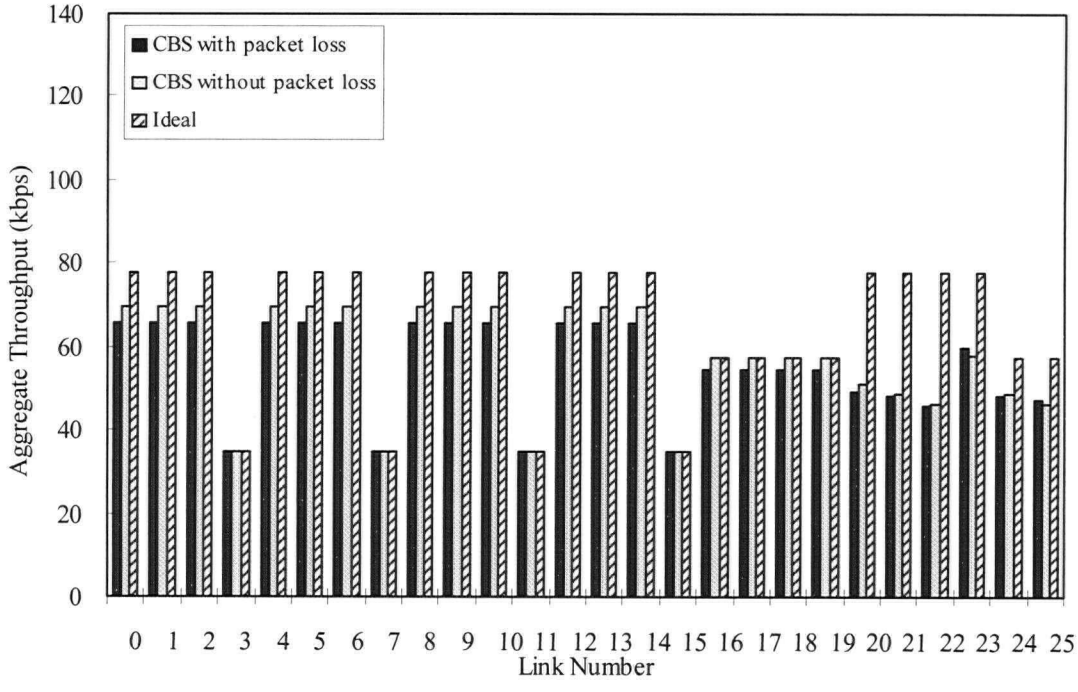

Figure 4.7. Fairness comparison for CBS on master

The simulation results from each scheme are compared with the ideal max-min fairness bandwidth allocation. Figure 4.5 shows the comparison for ASA with the ideal bandwidth distribution. Without packet loss or collision, ASA achieves the same results as the ideal bandwidth allocation either on master-slave or master-bridge links. When there is packet loss, ASA achieves a bandwidth slightly lower than the ideal values. Results in Figure 4.6 show that bandwidth allocation by FSS is similar to the ideal case only on the link with the lowest traffic rate. For all other master-slave links, FSS allocates a lower bandwidth to them. In addition, FSS allocates a higher bandwidth to most of the master-bridge links. Since FSS gives a higher priority to master-bridge links, the master-bridge links receive more bandwidth than the master-slave links. Results in Figure 4.7 show that bandwidth allocation by CBS is similar to the ideal value on master-slave links but not on master-bridge links. Since CBS does not prevent bridge node conflict, it leads to a lower bandwidth allocation on

the master-bridge links.

4.1.3 Unbalance Traffic Load around a Bridge

In this experiment, we investigate how each scheme can re-allocate the bandwidth around the bridge node according to the max-min fairness criterion. Table 4.3 shows the traffic pattern. We reduce the traffic rate for all master-slave links in piconets of M1, M3, M4, and M5 to 0.1×MATP. Therefore, the remaining bandwidth should be reallocated to the master-bridge links. As link 24 in piconet of M2 generates traffic at a rate of 0.166×MATP, 0.834×MATP of bandwidth can be equally shared between links 20 and 21. Similarly, as link 25 in piconet of M2 generates traffic at a rate of 0.166×MATP, 0.834×MATP of bandwidth can be equally shared between links 22 and 23. Although the traffic rate for links 20, 21, 22, and 23 each generates traffic at a rate of 0.6×MATP, the ideal sharing bandwidth for them will only be (0.834/2) ×MATP = 0.417×MATP = 144.115 kbps.

The simulation results from each scheme are compared with the ideal max-min fair bandwidth allocation. Figure 4.8 shows the comparison for the ASA scheme with the ideal bandwidth distribution. ASA achieves similar results as the ideal bandwidth allocation except for links 20 to 23. The reason for not fully utilizing the remaining bandwidth on links 20 to 23 is that in order to avoid conflict between all the connected master-bridge links, sometimes a bridge node may need to delay the meeting time with a master node since the ideal meeting time has already reserved for another master. Therefore, the bridge node has to sacrifice some time slots to prevent bridge node conflict.

Link	Flow	Traffic Rate (kbps)	Traffic Rate / MATP	Ideal Max-Min Shared Ratio	Ideal Shared Bandwidth (kbps)
0	M1<->S1	34.56	0.1	0.1	34.56
1	M1<->S2	34.56	0.1	0.1	34.56
2	M1<->S3	34.56	0.1	0.1	34.56
3	M1<->S4	34.56	0.1	0.1	34.56
4	M4<->S13	34.56	0.1	0.1	34.56
5	M4<->S14	34.56	0.1	0.1	34.56
6	M4<->S15	34.56	0.1	0.1	34.56
7	M4<->S16	34.56	0.1	0.1	34.56
8	M3<->S9	34.56	0.1	0.1	34.56
9	M3<->S10	34.56	0.1	0.1	34.56
10	M3<->S11	34.56	0.1	0.1	34.56
11	M3<->S12	34.56	0.1	0.1	34.56
12	M5<->S17	34.56	0.1	0.1	34.56
13	M5<->S18	34.56	0.1	0.1	34.56
14	M5<->S19	34.56	0.1	0.1	34.56
15	M5<->S20	34.56	0.1	0.1	34.56
16	M2<->S5	57.6	0.166	0.166	57.6
17	M2<->S6	57.6	0.166	0.166	57.6
18	M2<->S7	57.6	0.166	0.166	57.6
19	M2<->S8	57.6	0.166	0.166	57.6
20	M1<->B1	207.36	0.6	0.417	144.115
21	M4<->B1	207.36	0.6	0.417	144.115
22	M3<->B2	207.36	0.6	0.417	144.115
23	M5<->B2	207.36	0.6	0.417	144.115
24	M2<->B1	57.6	0.166	0.166	57.6
25	M2<->B2	57.6	0.166	0.166	57.6

Table 4.3. Traffic pattern for testing bandwidth allocation around bridge

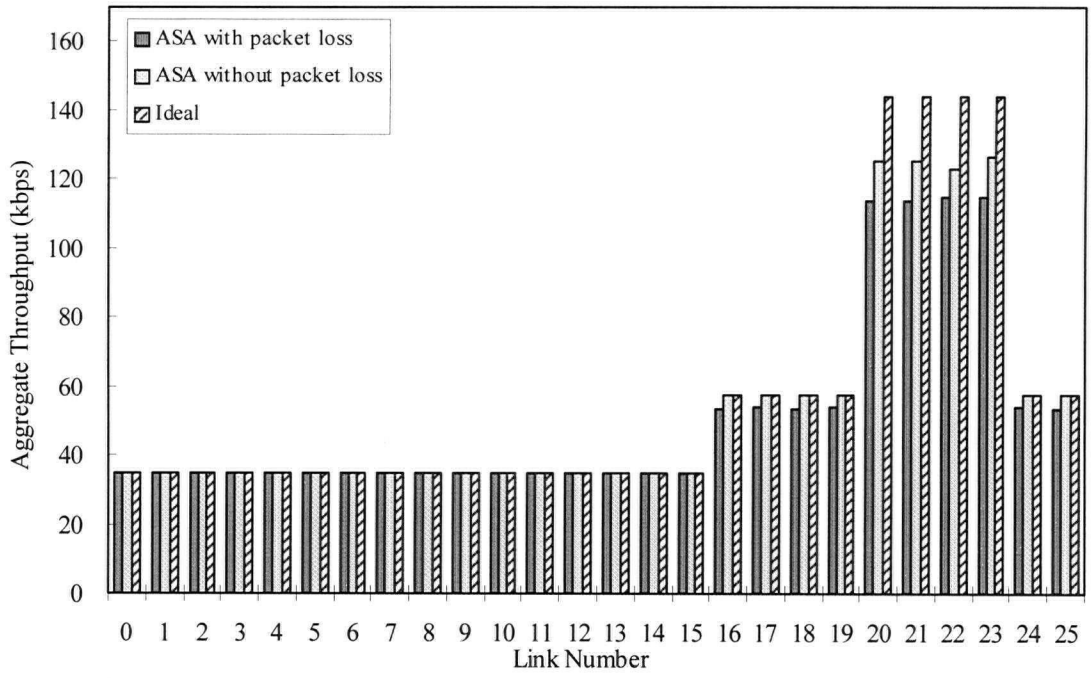

Figure 4.8. Fairness comparison for ASA on bridge

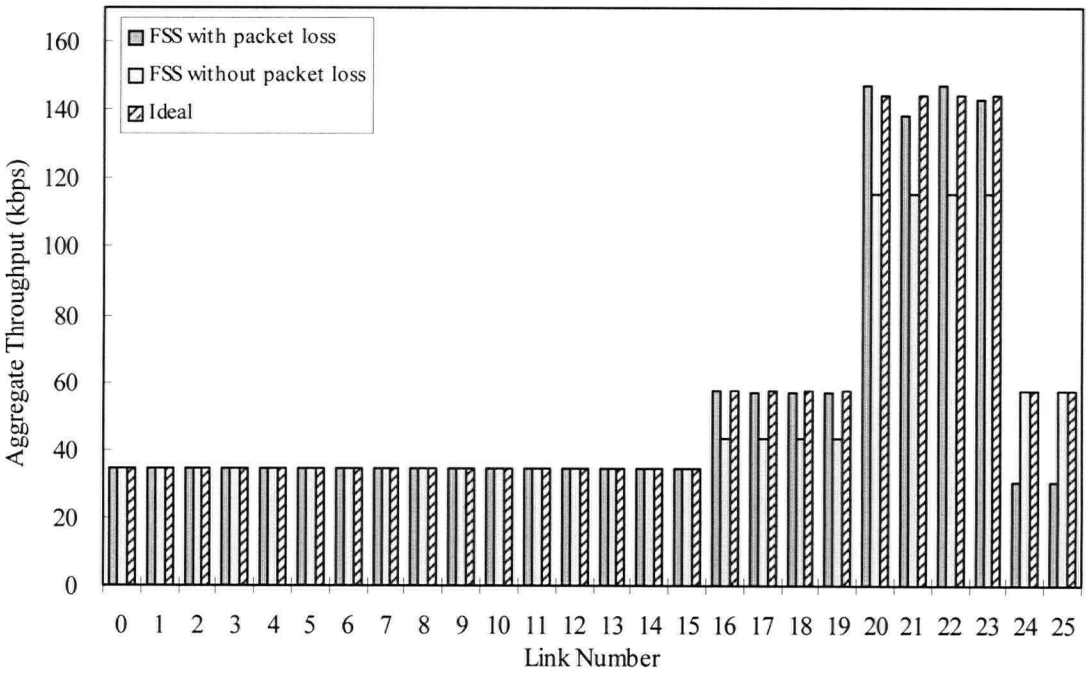

Figure 4.9. Fairness comparison for FSS on bridge

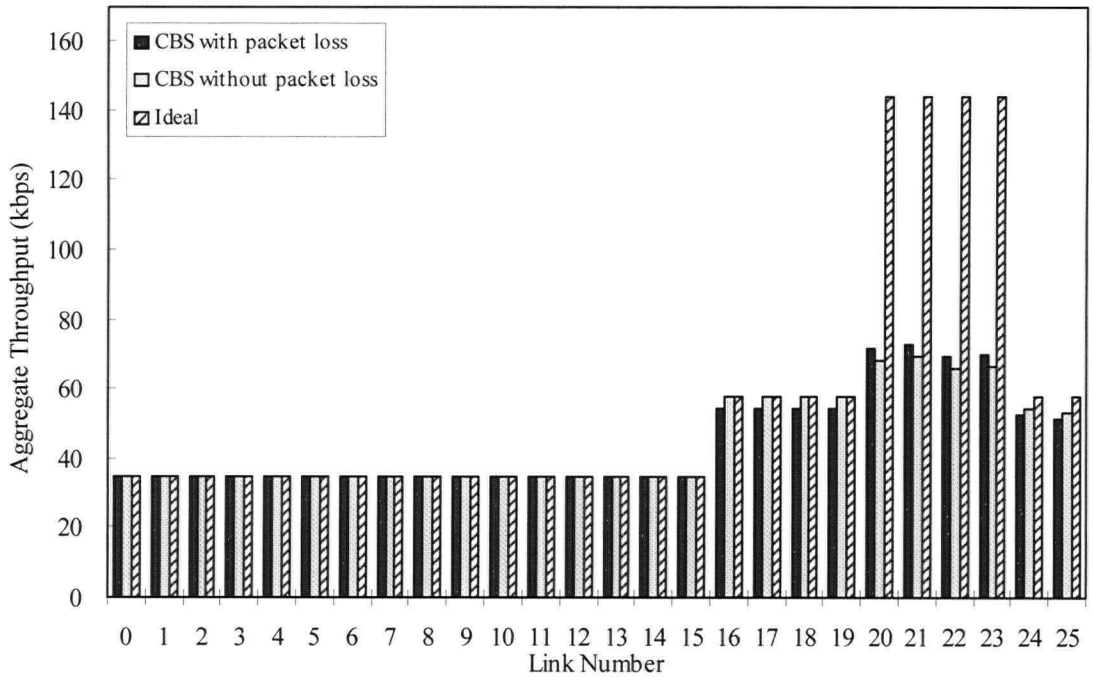

Figure 4.10. Fairness comparison for CBS on bridge

On the other hand, in Figure 4.9, it shows that FSS achieves similar results as our scheme when there is no packet loss. Again, in order to prevent bridge node conflict, FSS cannot fully utilize the allocated bandwidth on links 20 to 23. However, since the traffic rate on all master-slave links are relatively low, FSS can achieve the ideal bandwidth on all master-slave links. When there is packet loss, FSS does not allocate bandwidth in a fair manner between inter-piconet links. Since some time frames in switch table for links 24 and 25 are borrowed to link 20 to 23, FSS allocates a higher than expected bandwidth to links 20 to 23 and a lower than expected bandwidth to links 24 to 25. Lastly, in Figure 4.10, it shows that CBS cannot achieve the ideal bandwidth allocation on links 20 to 23 because of bridge node conflicts.

4.2 Comparison with UDP Traffic

In this section, we focus on how all three schemes perform under UDP traffic. In the first case, we focus on CBR UDP traffic. In the second case, we focus on bursty on-off UDP traffic. In both cases, DH1 packets are used.

For all test cases, the parameters for FSS are set as follows: α =10, β = 5%, len_q_{max} = 800, P_{max} = 35, and R_{max} = 6. Each frame in the switch table is set to two Bluetooth slots. For CBS, N_{switch_th} is set to 0. The T_{sniff} value for links around M1, M4, M3, and M5 is set to 10, and the T_{sniff} value for links around M2 is set to 12. T_{poll} is set to 100. For ASA, *SS_Slot* and *MUSS* are set to two Bluetooth slots. The values of *avg_interval* and *restart_ratio* in traffic estimator are set to 15s and 5 respectively. The value of *max_waiting_interval* for trigger point estimation is set to the time duration for 100 Bluetooth slots.

4.2.1 CBR Scenario

We compare the system throughput and delay for all three schemes on CBR UDP bi-directional traffic between a master and a slave, which is specified as *M-S traffic*, and CBR UDP bi-directional traffic between a master and another master through a bridge node, which is specified as *M-B-M traffic*. In this experiment, a bridge node will not be either a sender or receiver, but is only responsible for bypassing packets to another connected master. Table 4.4 shows the traffic pattern for the simulation.

We set the traffic rate on each node according to the assumption that each link within a piconet demands for the same amount of shared bandwidth. Therefore, each link in piconet of M1, M3, M4, and M5 has a shared bandwidth of 1/5×MATP = 69.12 kbps; each node involving in M-S traffic generates traffic at a rate of 69.12 kbps/2 = 34.56 kbps×β. In the first

set of simulation, β is set to 1. β is a multiplying factor which is used to vary the traffic rate on each node. Since M1, M3, M4, and M5 each has two M-S-M traffic flows on a link, each of them generates traffic at a rate of 34.56 kbps/2 = 17.28 kbps×β to individual M-S-M traffic flow. On the other hand, since each link in the piconet of M2 has a shared bandwidth of 1/6×MATP = 57.6 kbps, each node involving in M-S traffic generates traffic at a rate of 57.6 kbps/2 = 28.8 kbps×β. Again, since M2 has two M-S-M traffic flows on a link, it generates traffic at a rate of 28.8 kbps/2 = 14.4 kbps×β to individual M-S-M traffic flow. Lastly, we vary the value of β from 0 to 1.2 to observe the performance change. In Figure 4.11 and Figure 4.12, results show that ASA achieves the highest aggregate throughput and the lowest average delay when compared to FSS and CBS. As shown in Section 4.1.1, CBS has a better performance on master-slave links than master-bridge links; since there is more traffic involving slave nodes than bridge nodes in the simulation, CBS has a close performance with our scheme.

In the next experiment, we compare the system throughput and delay for all three schemes on CBR UDP bi-directional traffic between a slave and another slave through a master, which is specified as *S-M-S traffic*, and CBR UDP bi-directional traffic between a slave and another slave through two masters and a bridge node, which is specified as *S-M-B-M-S traffic*. In this experiment, both masters and bridge nodes will not be senders or receivers. They are only responsible for bypassing packets. Table 4.5 shows the traffic pattern for the simulation.

Flow M_i<->S_i	**M_i (kbps)**	**S_i (kbps)**	**Flow M_i<-B_j->M_k**	**M_i (kbps)**	**M_k (kbps)**
M1<->S1	34.56 * β	34.56 * β	M1<-B1->M2	17.28 * β	14.4 * β
M1<->S2	34.56 * β	34.56 * β	M4<-B1->M1	17.28 * β	17.28 * β
M1<->S3	34.56 * β	34.56 * β	M2<-B1->M4	14.4 * β	17.28 * β
M1<->S4	34.56 * β	34.56 * β	M3<-B2->M2	17.28 * β	14.4 * β
M4<->S13	34.56 * β	34.56 * β	M5<-B2->M3	17.28 * β	17.28 * β
M4<->S14	34.56 * β	34.56 * β	M2<-B2->M5	14.4 * β	17.28 * β
M4<->S15	34.56 * β	34.56 * β			
M4<->S16	34.56 * β	34.56 * β			
M3<->S9	34.56 * β	34.56 * β			
M3<->S10	34.56 * β	34.56 * β			
M3<->S11	34.56 * β	34.56 * β			
M3<->S12	34.56 * β	34.56 * β			
M5<->S17	34.56 * β	34.56 * β			
M5<->S18	34.56 * β	34.56 * β			
M5<->S19	34.56 * β	34.56 * β			
M5<->S20	34.56 * β	34.56 * β			
M2<->S5	28.8 * β	28.8 * β			
M2<->S6	28.8 * β	28.8 * β			
M2<->S7	28.8 * β	28.8 * β			
M2<->S8	28.8 * β	28.8 * β			

Table 4.4. Traffic pattern for M-S and M-B-M traffic generating

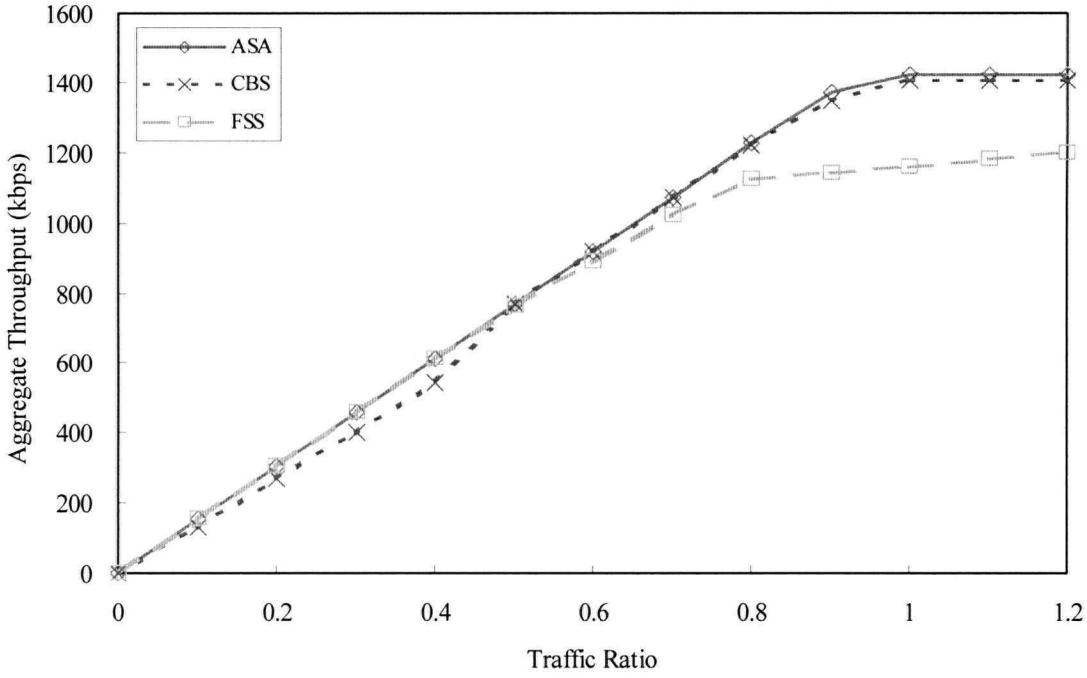

Figure 4.11. Aggregate throughput for M-S and M-B-M traffic

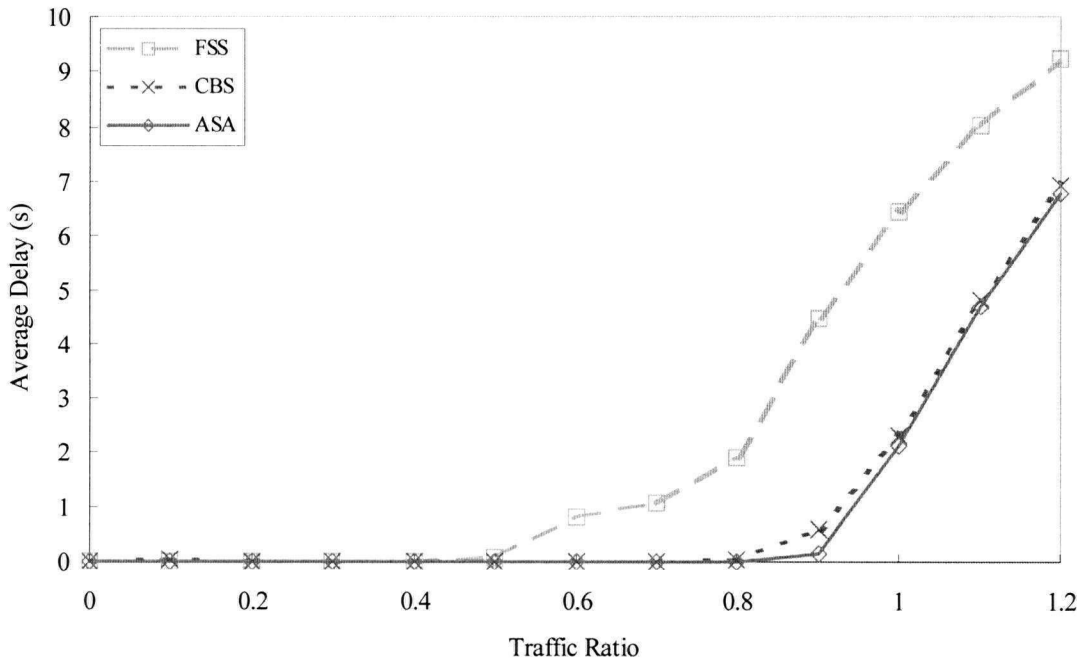

Figure 4.12. Average delay for M-S and M-B-M traffic

Flow S_i<-M_j->S_k	S_i (kbps)	S_k (kbps)	**Flow** S_i<-M_j-B_k-M_l->S_m	S_i (kbps)	S_m (kbps)
S1<-M1->S2	34.56 * β	34.56 * β	S12<-M3-B2-M5->S20	34.56 * β	34.56 * β
S17<-M5->S18	34.56 * β	34.56 * β	S19<-M5-B2-M2->S8	34.56 * β	28.8 * β
S13<-M4->S14	34.56 * β	34.56 * β	S15<-M4-B1-M2->S6	34.56 * β	28.8 * β
S9<-M3->S10	34.56 * β	34.56 * β	S3<-M1-B1-M2->S5	34.56 * β	28.8 * β
			S4<-M1-B1-M4->S16	34.56 * β	34.56 * β
			S11<-M3-B2-M2->S7	34.56 * β	28.8 * β

Table 4.5. Traffic pattern for S-M-S and S-M-B-M-S traffic

In the simulation, all slave nodes have the same traffic rate defined in the previously. Therefore, slave nodes in piconet M1, M3, M4, and M5 generate packet at a rate of 34.56×β, and slave nodes in piconet of M2 generate packets at a rate of 28.8×β kbps. Lastly, we vary the value of β from 0 to 1 to observe the performance change. In Figure 4.13 and Figure 4.14, results show that ASA still achieves the highest aggregate throughput and the lowest average delay when compares to CBS and FSS. However, in this simulation, there are more traffic flows involving bridge nodes than slave nodes. Thus, as FSS has better performance on master-bridge links than master-slave links, it has a close aggregate throughput performance with our scheme. Nevertheless, as shown in Section 4.1.2, FSS allocates more than the ideal amount of bandwidth to master-bridge links around a master; therefore, when all nodes are generating traffic at the saturated rate, FSS allocates more bandwidth to master-bridge links than master-slave links. Since the path for S-M-B-M-S traffic is longer than S-M-S traffic, FSS achieves a higher average delay when compared to our scheme.

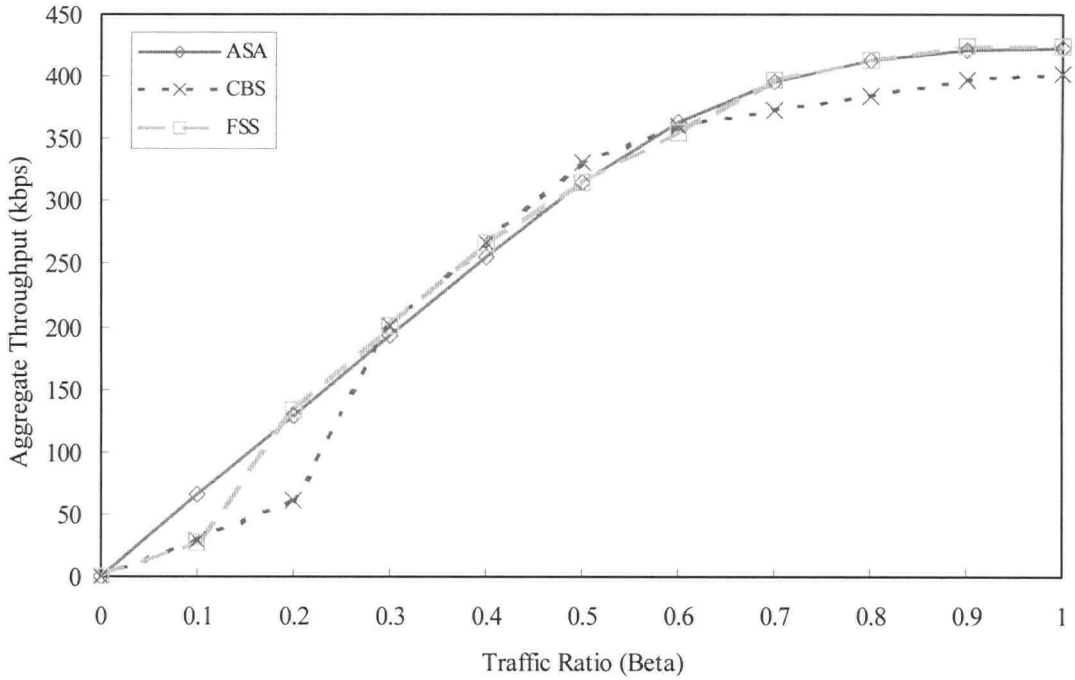

Figure 4.13. Aggregate throughput for S-M-S and S-M-B-M-S traffic

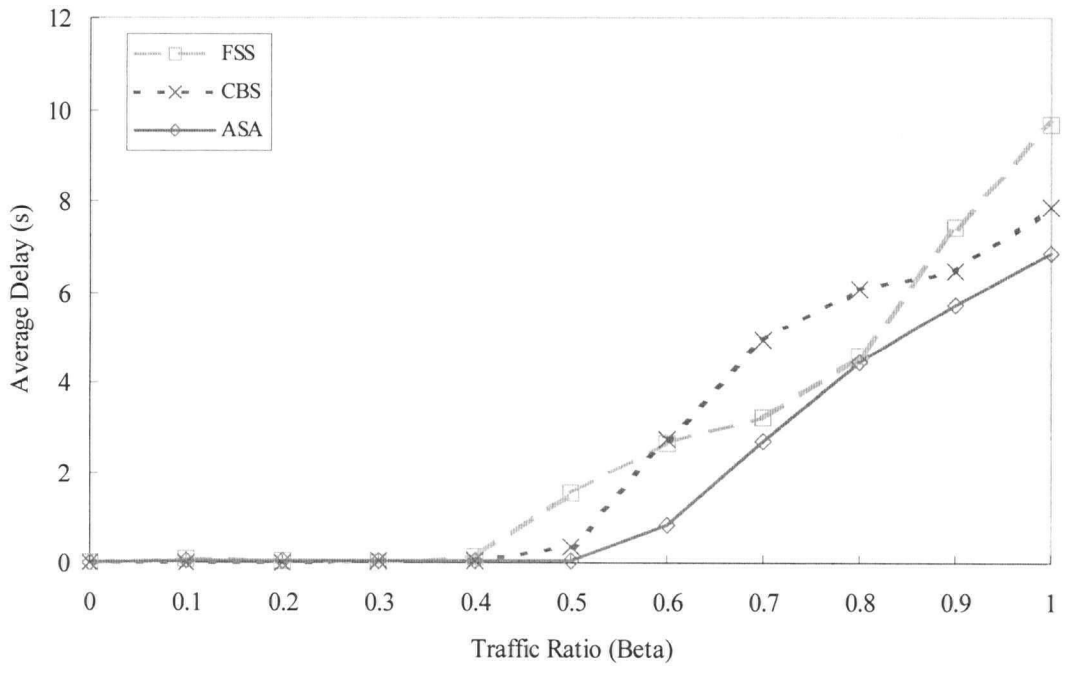

Figure 4.14. Average Delay for S-M-S and S-M-B-M-S traffic

4.2.2 Bursty On-Off Traffic Scenario

In this experiment, we compare the system throughput and delay under bursty on-off UDP traffic. We use the same traffic pattern as shown in Table 4.4. Again, we assume that a bridge does not generate packets, and is only responsible for forwarding packets to another connected master.

For all the test cases, the parameters for FSS are set as α=10, β= 5%, len_q_{max} = 800, P_{max} = 35 and R_{max} = 6. For CBS, N_{switch_th} is set to 0. The T_{sniff} value for links around M1, M4, M3, and M5 is set to 10, and the T_{sniff} value for links around M2 is set to 12. T_{poll} is set to 100. For ASA, *SS_Slot* and *MUSS* are set to two Bluetooth slots. The *avg_interval* and *restart_ratio* in traffic estimator are set to 15s and 5 respectively. The *max_waiting_interval* for trigger point estimation is set to the time duration for 100 Bluetooth slots.

We assume that both *on* and *off* periods follow the exponential distributions. In this first experiment, we set the *on* and *off* periods with an average duration of 1s and 1s, respectively. We vary the value of β from 0 to 2 to observe the performance change. The results are shown in Figure 4.15 and Figure 4.16. In the second experiment, we set the *on* and *off* periods with an average duration of 1s and 2s, respectively. We vary the value of β from 0 to 3 to observe the performance change. The results are shown in Figure 4.17 and Figure 4.18. In the last experiment, we set the *on* and *off* periods with an average duration of 2s and 1s, respectively. We vary the value of β from 0 to 1.8 to observe the performance change. The results are shown in Figure 4.19 and Figure 4.20. For all cases, results show that ASA achieves the highest aggregate throughput and lowest average delay for all different on-off periods when compared with FSS and CBS.

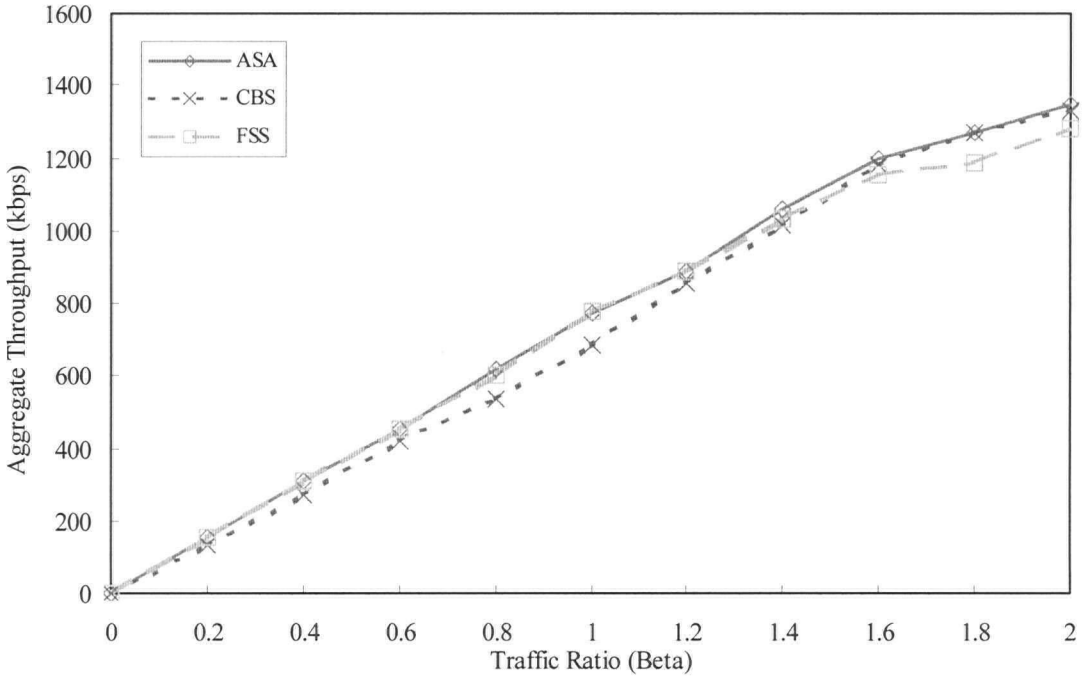

Figure 4.15. Aggregate throughput with ON:OFF ratio of 1:1

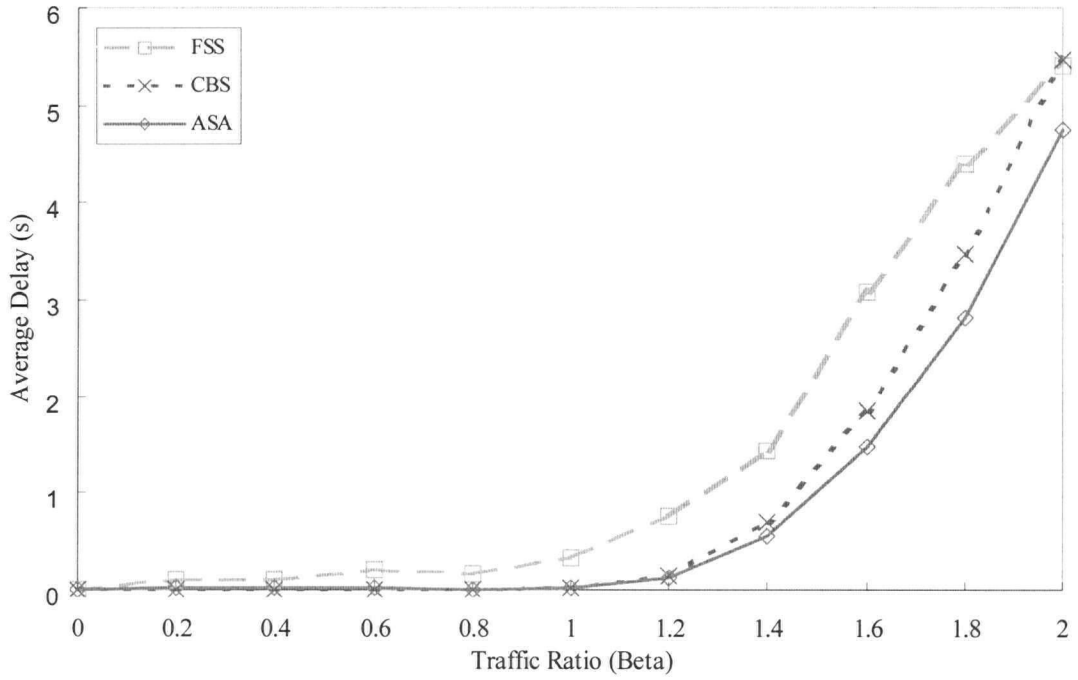

Figure 4.16. Average delay with ON:OFF ratio of 1:1

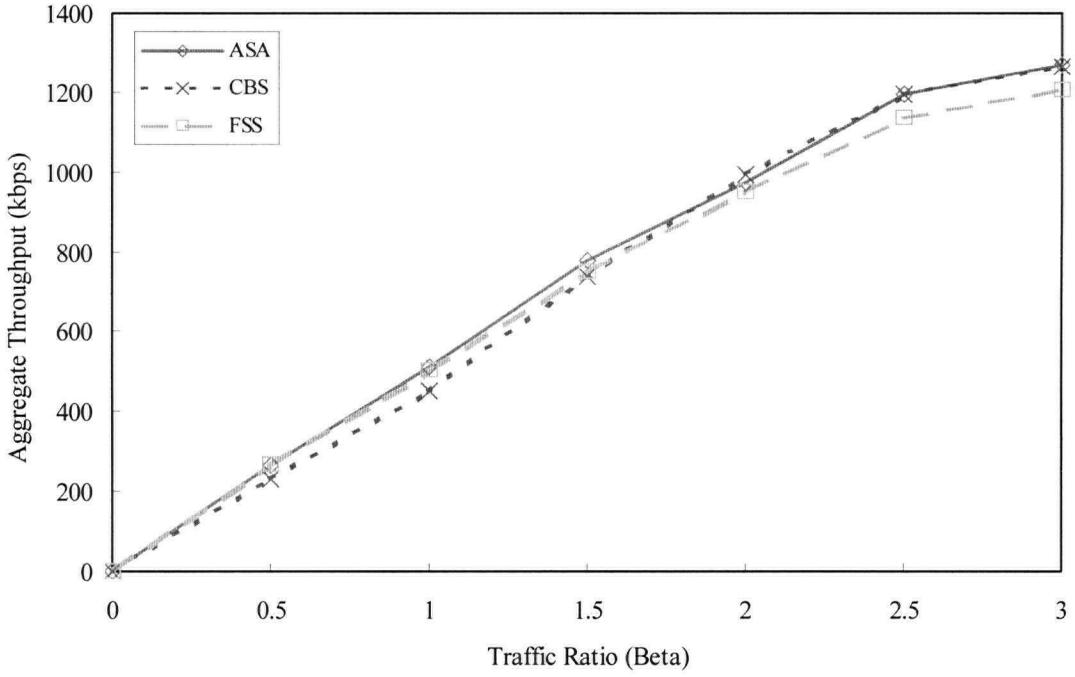

Figure 4.17. Aggregate throughput with ON:OFF ratio of 1:2

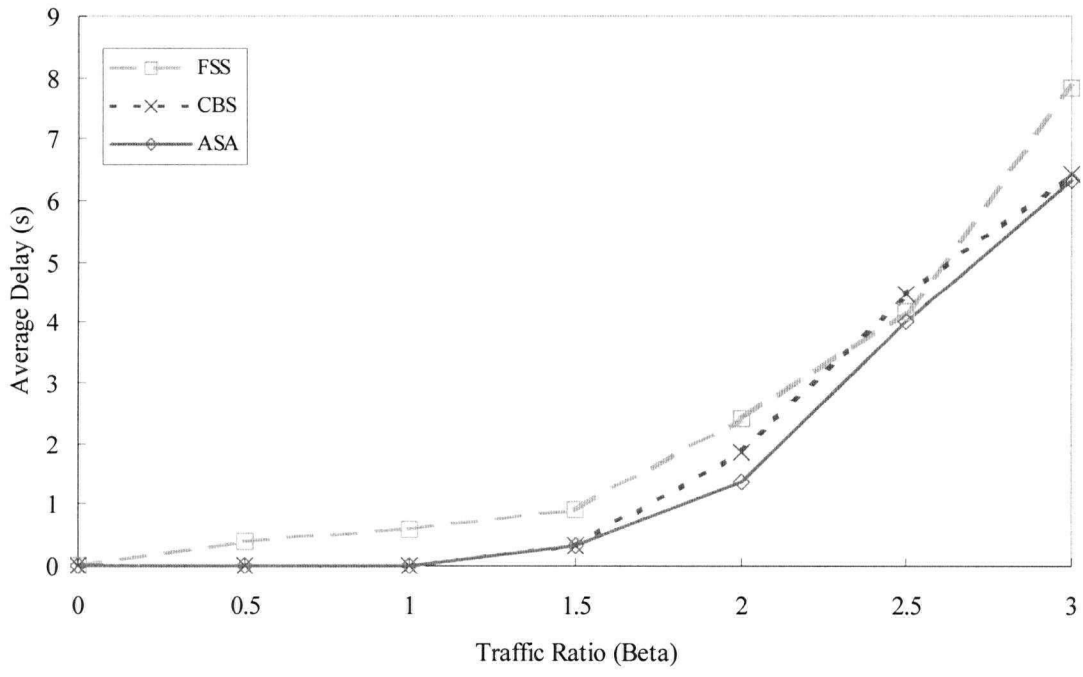

Figure 4.18. Average delay with ON:OFF ratio of 1:2

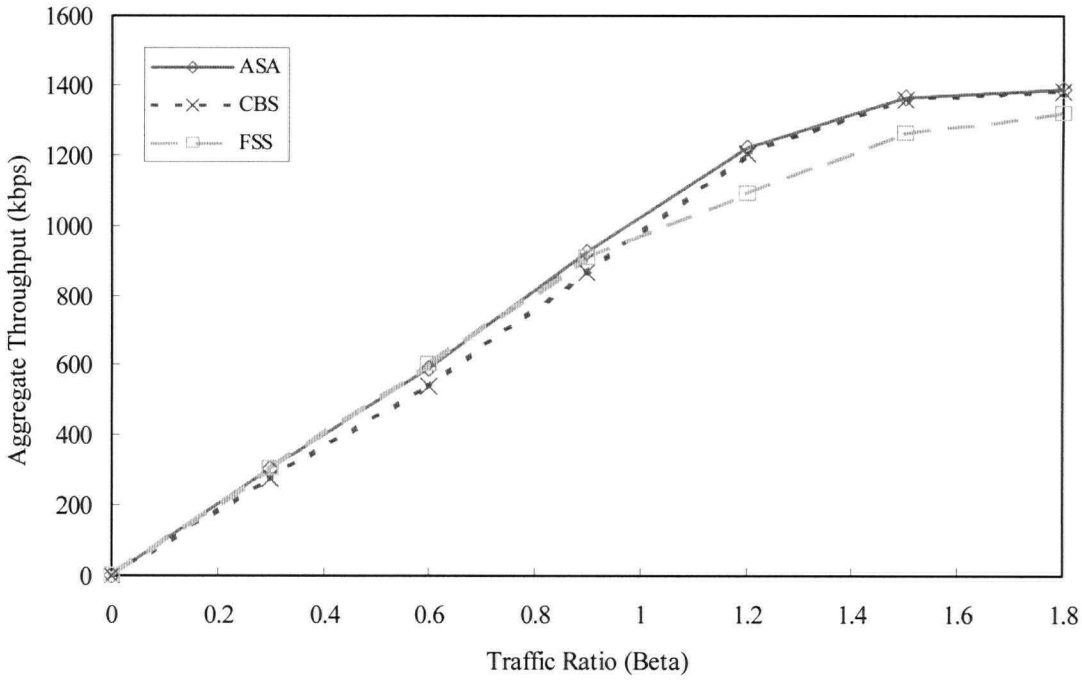

Figure 4.19. Aggregate throughput with ON:OFF ratio of 2:1

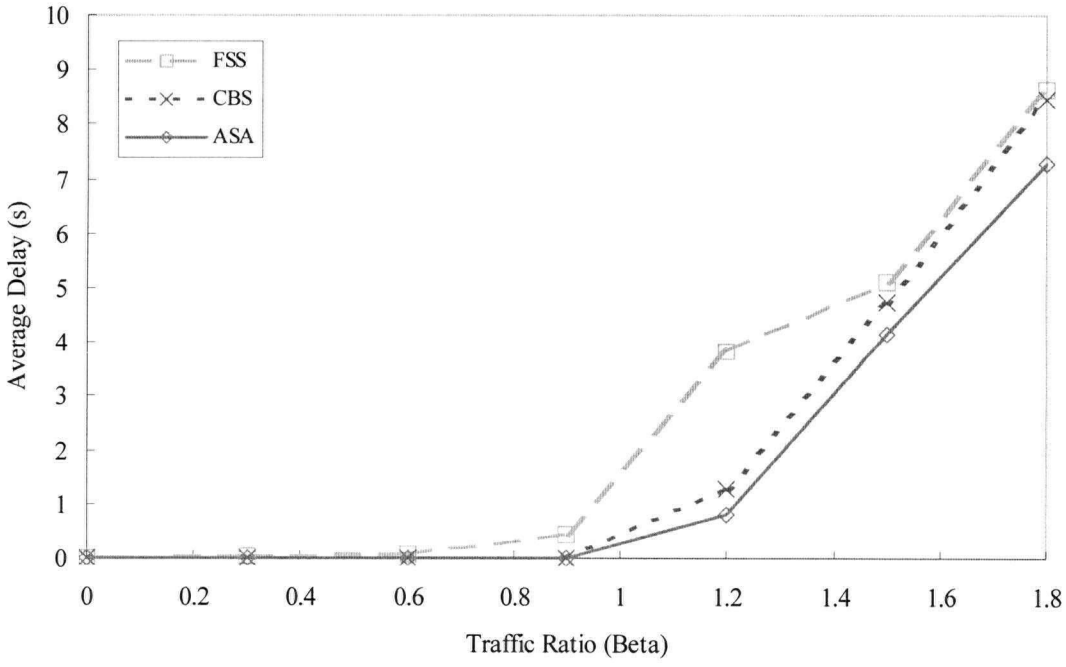

Figure 4.20. Average delay with ON:OFF ratio of 2:1

Flow	Flow	Flow	Flow	Flow
M1->S1	M3->S9	M5->S17	M4->S13	M2->S5
M1->S2	M3->S10	M5->S18	M4->S14	M2->S6
M1->S3	M3->S11	M5->S19	M4->S15	M2->S7
M1->S4	M3->S12	M5->S20	M4->S16	M2->S8
M1->B1->M2	M3->B2->M2	M5->B2->M2	M4->B1->M2	

Table 4.6. Traffic Pattern for M-S and M-B-M

4.3 Comparison with TCP Traffic

In this section, we compare the average end-to-end transfer delay for sending a file from one node to another node using TCP between all three schemes.

In the first part, the parameters for FSS are set as α =10, β = 5%, len_q_{max} = 800, P_{max} = 35, and R_{max} = 6. Each frame in the switch table is set to 10 Bluetooth slots. For CBS, N_{switch_th} is set to 0. The T_{sniff} value for links around M1, M4, M3, and M5 is set to 50, and the T_{sniff} value for links around M2 is set to 60. T_{poll} is set to 1200. For ASA, *SS_Slot* and *MUSS* are set to 10 Bluetooth slots. The *avg_interval* and *restart_ratio* in traffic estimator are set to 15s and 5, respectively. The value of *max_waiting_interval* for trigger point estimation is set to the time duration for 100 Bluetooth slots.

In the first test case, we assume that only masters transmit data. Table 4.6 shows the traffic pattern for the simulation. Each master sends a file to its slaves. The file message will be segmented into either DH1, DH2 or DH3 packets according to the Bluetooth specification for packet segmentation. We refer this as the *M-S traffic*. Moreover, masters M0, M1, M3, and M4 each sends the same size file through a bridge node to M2. We refer this as *M-B-M traffic*. We vary the file size from 0.1 MB to 0.5 MB to observe the performance change. The simulation results are shown in Figure 4.21 and Figure 4.22.

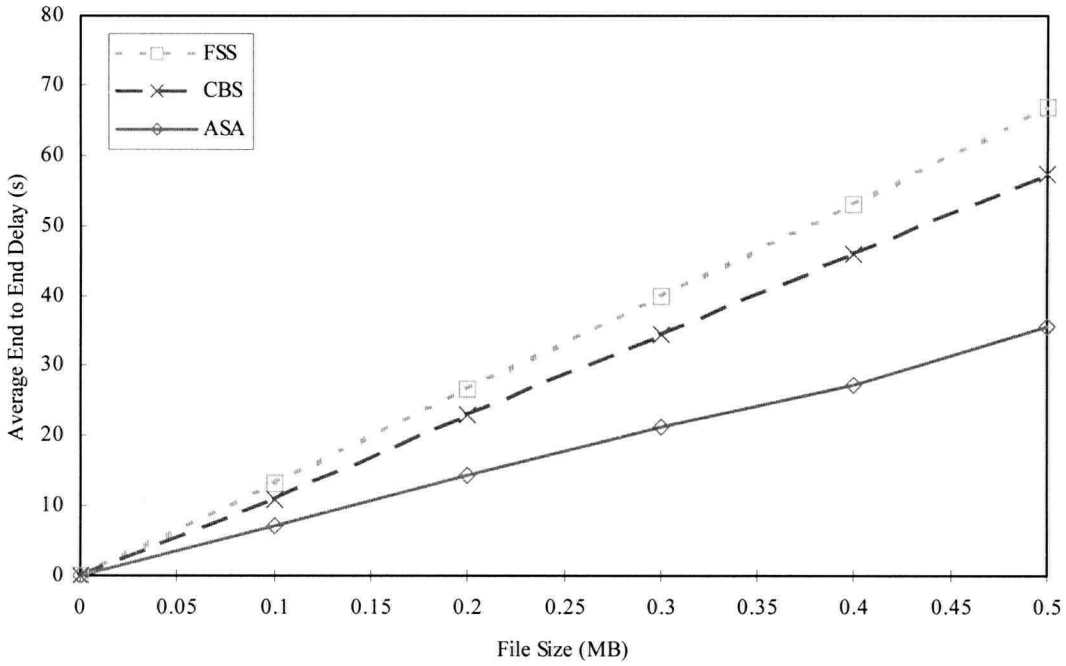

Figure 4.21. TCP Traffic between M-S

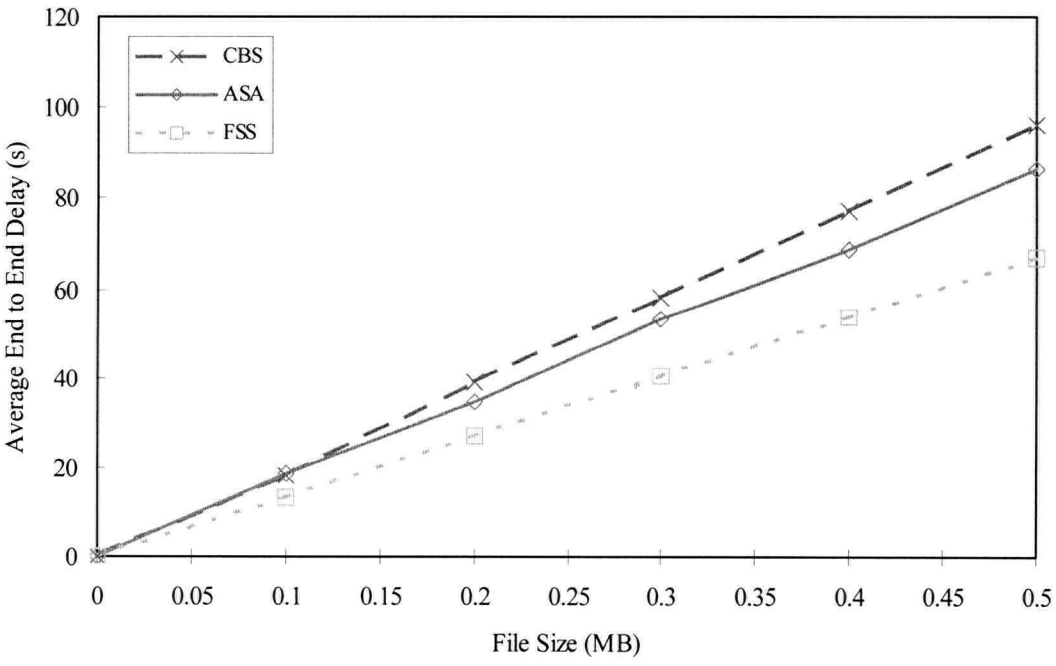

Figure 4.22. TCP Traffic between M-S-M

Flow	Flow	Flow	Flow
S1->M1->S2	S9->M3->S10	S13->M4->S14	S17->M5->S18
S3->M1-B1-M2->S5	S11->M3-B2-M2->S7	S15->M4-B1-M2->S6	S19->M5-B2-M2->S8

Table 4.7. Traffic Pattern for S-M-S and S-M-B-M-S

Figure 4.21 shows that ASA has the lowest average transfer delay on M-S traffic and FSS has the highest delay. Figure 4.22 shows that ASA and CBS have similar performance for the average transfer delay on M-B-M traffic, while FSS has a better performance. This is due to the fact that FSS gives a higher priority on master-bridge link.

In the second simulation, the parameters for FSS are set as α =10, β = 5%, len_q_{max} = 800, P_{max} = 35, and R_{max} = 6. Each frame in the switch table is set to 10 Bluetooth slots. For CBS, N_{switch_th} is set to 0. The T_{sniff} value for links around M1, M4, M3, and M5 is set to 40, and the T_{sniff} value for links around M2 is set to 60. T_{poll} is set to 1200. For ASA, *SS_Slot* and *MUSS* are set to 10 Bluetooth slots. The *avg_interval* and *restart_ratio* in traffic estimator are set to 15s and 5 respectively. The value of *max_waiting_interval* for trigger point estimation is set to the time duration for 100 Bluetooth slots.

In this test case, we assume that only slave nodes transmit data. Table 4.7 shows the traffic pattern for the simulation. We set up a one-way TCP traffic flow from one slave to another slave through a bridge node. We refer this as the *S-M-S traffic*. We also set up another one-way TCP traffic flow from one slave to another slave through two masters and a bridge node. We refer this as the *S-M-B-M-S traffic*. We again vary the file size from 0.1 MB to 0.5 MB to observe the performance change. The results are shown in Figure 4.23 and Figure 4.24. The figures show that ASA has the lowest average end-to-end delay when considered both S-M-S and S-M-B-M-S traffic.

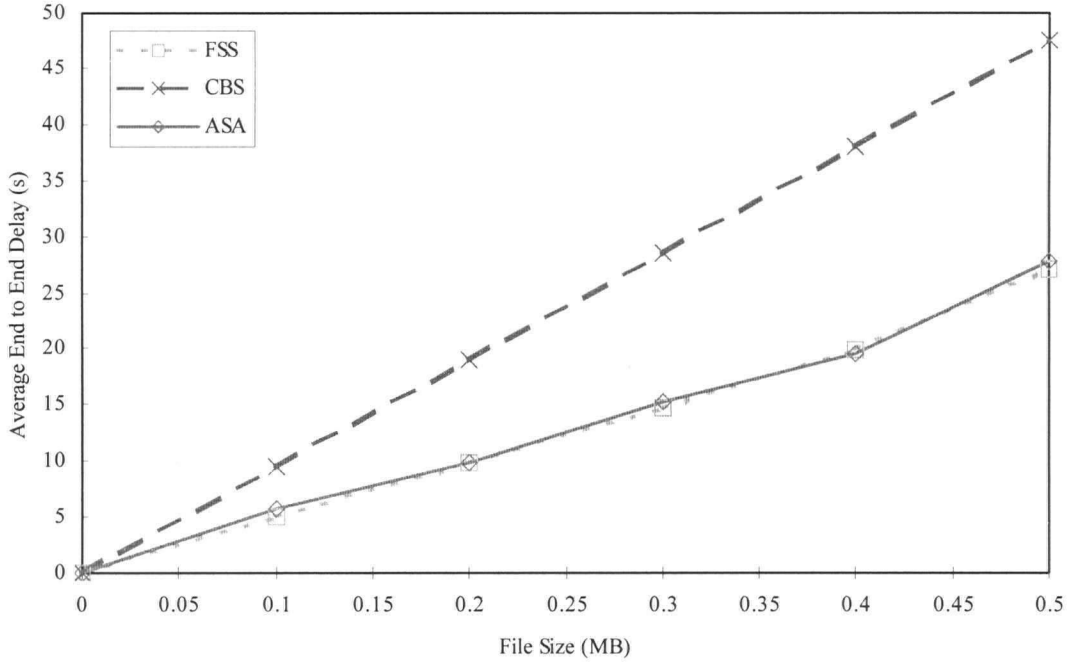

Figure 4.23. TCP Traffic between S-M-S

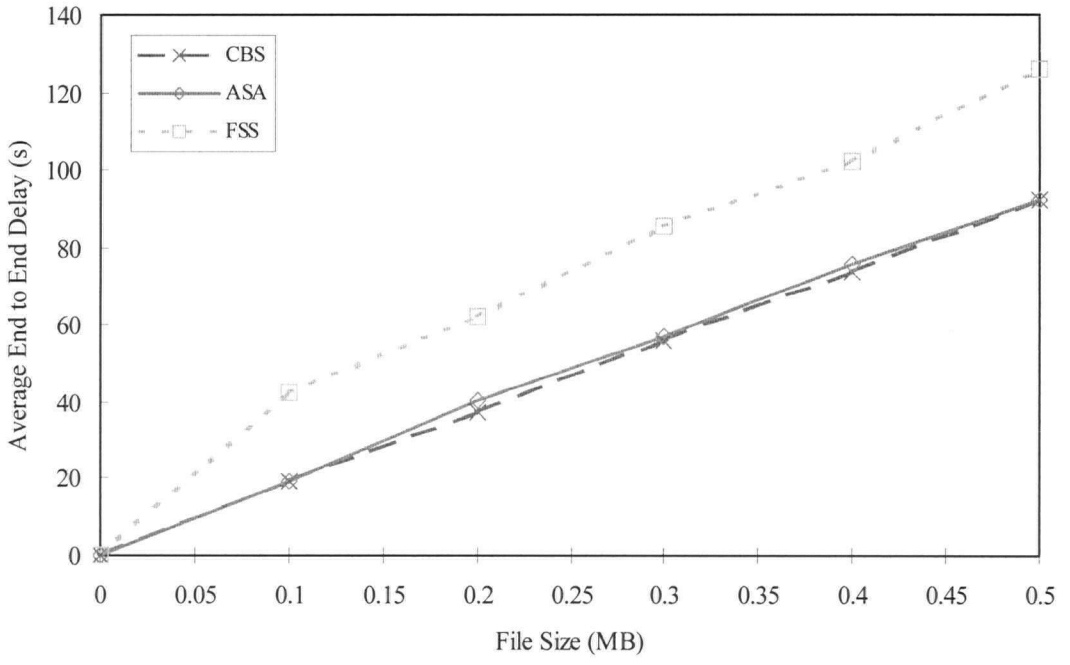

Figure 4.24. TCP Traffic between S-M-B-M-S

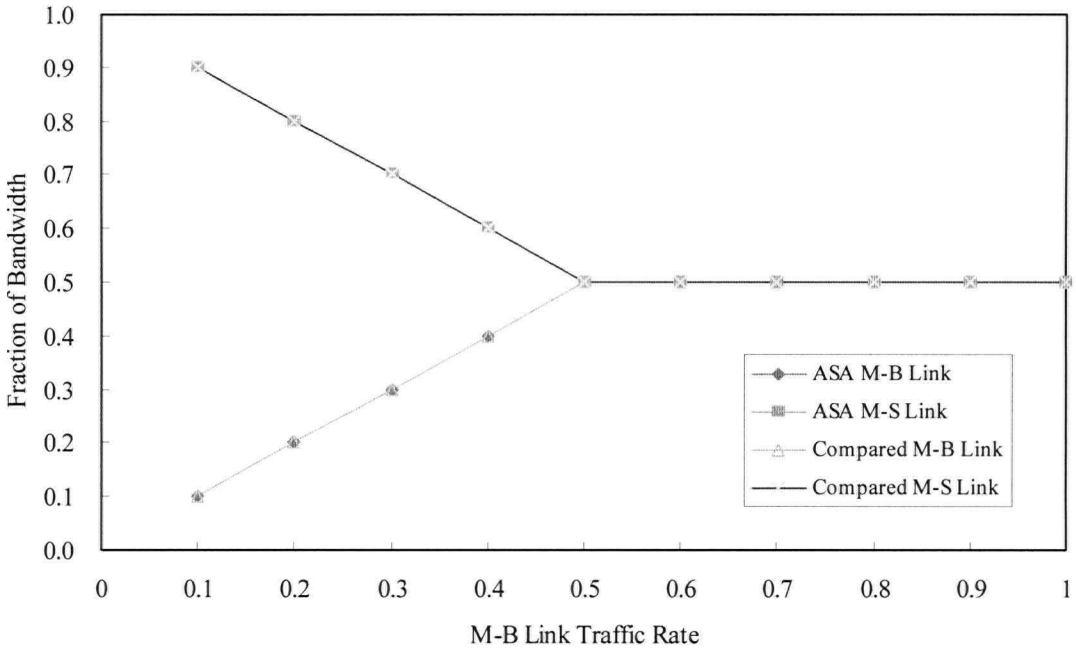

Figure 4.25. Sharing of bandwidth within a piconet

4.4 Repeated Experiments

In this section, we run some of the experiments in the paper of Fair and Traffic Dependent Scheduling Algorithm on ASA in order to compare the two schemes. All experiments focus on CBR UDP traffic and DH1 packets are used.

4.4.1 Single Bridge Node in Two Piconets

In the first experiment, there are two piconets with number I and II joined by a bridge node. The master in each piconet is connected with a slave node and the traffic rate on the master-slave (M-S) link is equal to MATP. On the other hand, both master-bridge (M-B) links generate traffic with the same rate. We then vary the traffic rate from 0.1 × MATP to MATP on both M-B links to observe the performance change. Figure 4.25 shows the results for one

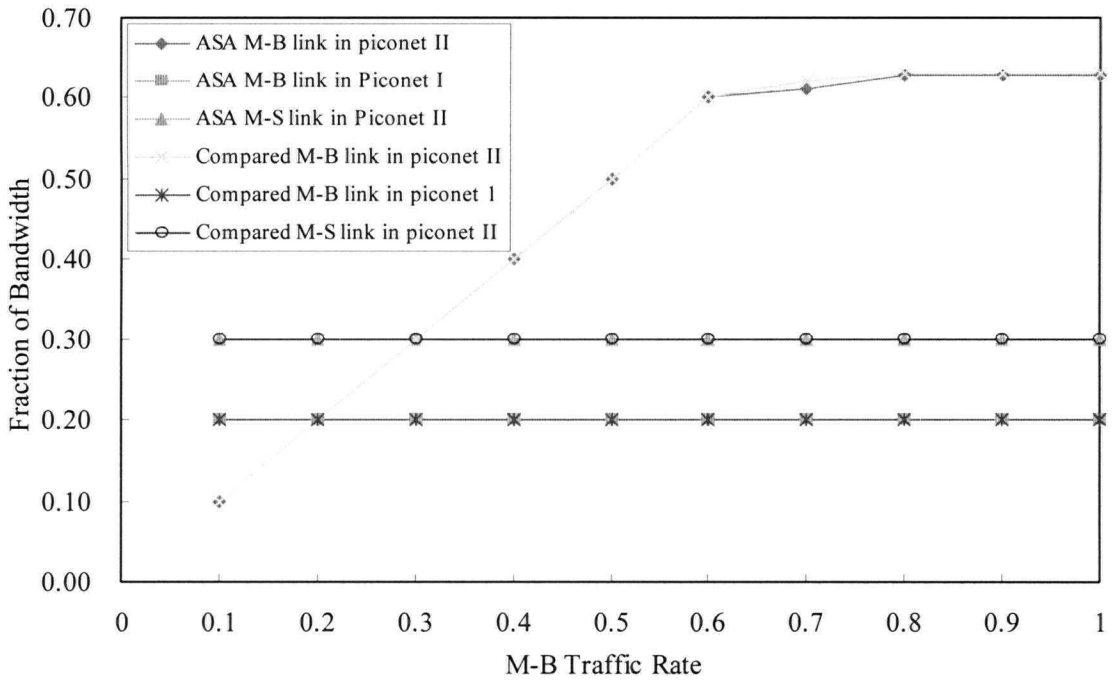

Figure 4.26. Sharing of bandwidth between bridge nodes and slave node

of the piconet. In Figure 4.25, it shows that the results for ASA and Fair and Traffic Dependent Scheduling Algorithm are very close.

In the second experiment, we use the same topology. However, the traffic rate on each M-S link is equal to 0.3 × MATP. The traffic rate on the M-B link in piconet I is equal to 0.2 × MATP. We then vary the traffic rate from 0.1 × MATP to MATP on the M-B link in piconet II to observe the change. Figure 4.26 shows the results for the bandwidth shared between the two M-B links in each piconet and the M-S link in piconet II. The results show that both ASA and Fair and Traffic Dependent Scheduling Algorithm can fairly share the bandwidth between bridge nodes and slave node.

4.4.2 Different Number of Slaves

In this experiment, there are two piconets with numbered I and II joined by a bridge node.

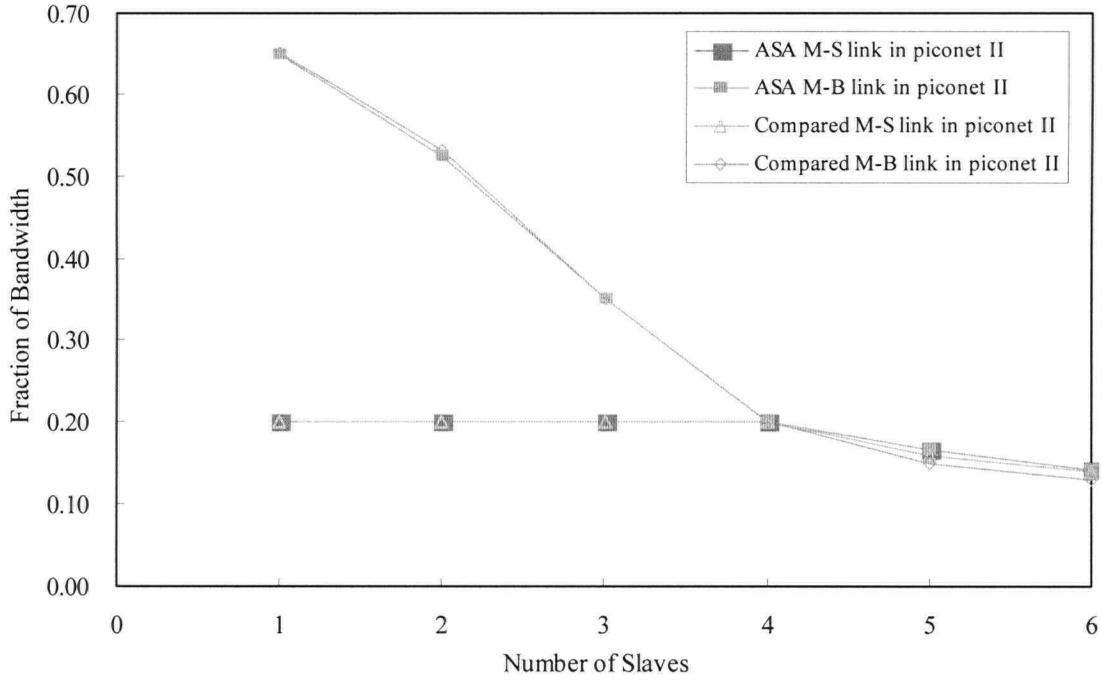

Figure 4.27. Sharing of bandwidth between bridge node and slave node in piconet II

The master in piconet I has 3 slaves. We vary the number of slave nodes connected to the master in piconet II from 1 to 6 to observe the performance change. The traffic rate on each M-S link is equal to 0.2 × MATP. On the other hand, the traffic rate on M-B link in piconet I is equal to 0.3 × MATP and the traffic rate on M-B link in piconet II is equal to 0.8 × MATP. Figure 4.27 shows the results for the bandwidth shared between the M-B link and the M-S link in piconet II. In Figure 4.27, it shows that both ASA and the Fair and Traffic Dependent Scheduling Algorithm can fairly share the bandwidth for the nodes. Sometimes, ASA performs better than Fair and Traffic Dependent Scheduling Algorithm, and sometimes the Fair and Traffic Dependent Scheduling Algorithm performs better than ASA.

4.4.3 Single Bridge Node Between Three Piconets

In this experiment, a bridge node is connected with three piconets with number I, II and III.

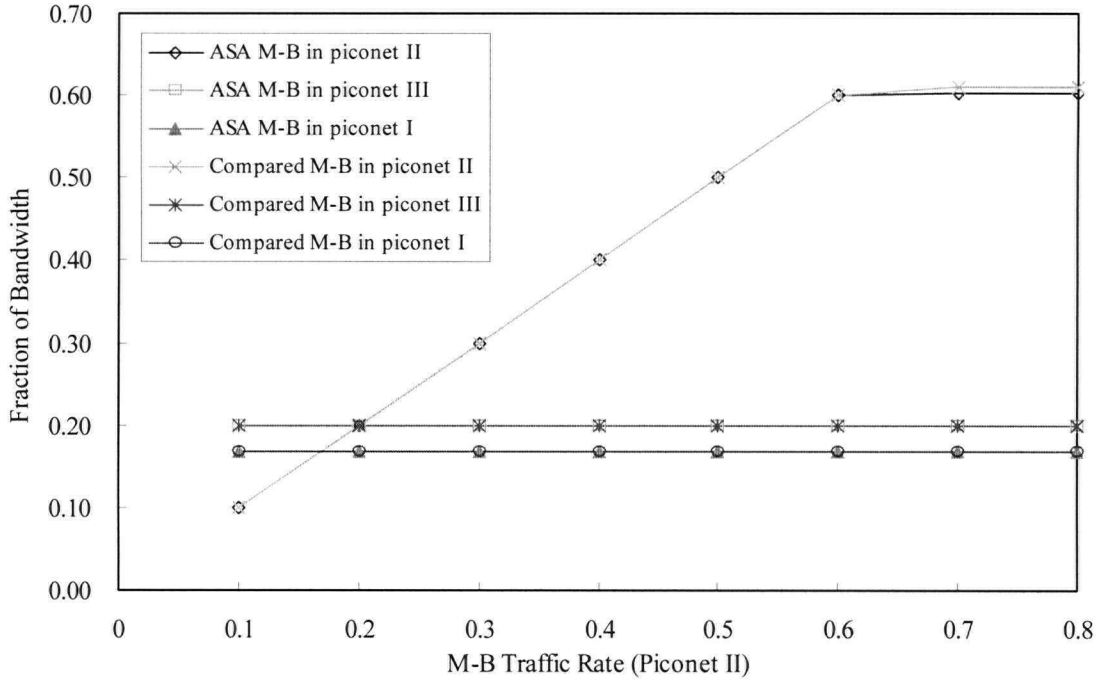

Figure 4.28. Sharing of bandwidth for all M-B links

The master in piconet I has 5 slaves, the master in piconet II has 1 slave, and the master in piconet III has 4 slaves. All M-S links generate traffic at a rate of 0.2 × MATP. On the other hand, the M-B link in piconet I has a traffic rate of 0.2 × MATP, the M-B link in piconet III has a traffic rate of 0.3 × MATP. We then vary the traffic rate on the M-B link in piconet II from 0.1 × MATP to 0.8 × MATP to observe the performance change. Figure 4.28 shows the bandwidth shared between all three M-B links. The results show that ASA achieves the same results as the Fair and Traffic Dependent Scheduling Algorithm.

4.4.4 Piconet with Two Bridge Nodes

In this experiment, bridge node B1 is connected between piconet I and II. On the other hand, bridge node B2 is connected between piconet II and III. Furthermore, the master in piconet I has 6 slaves, the master in piconet II has 2 slaves, and the master in piconet III has 4 slaves.

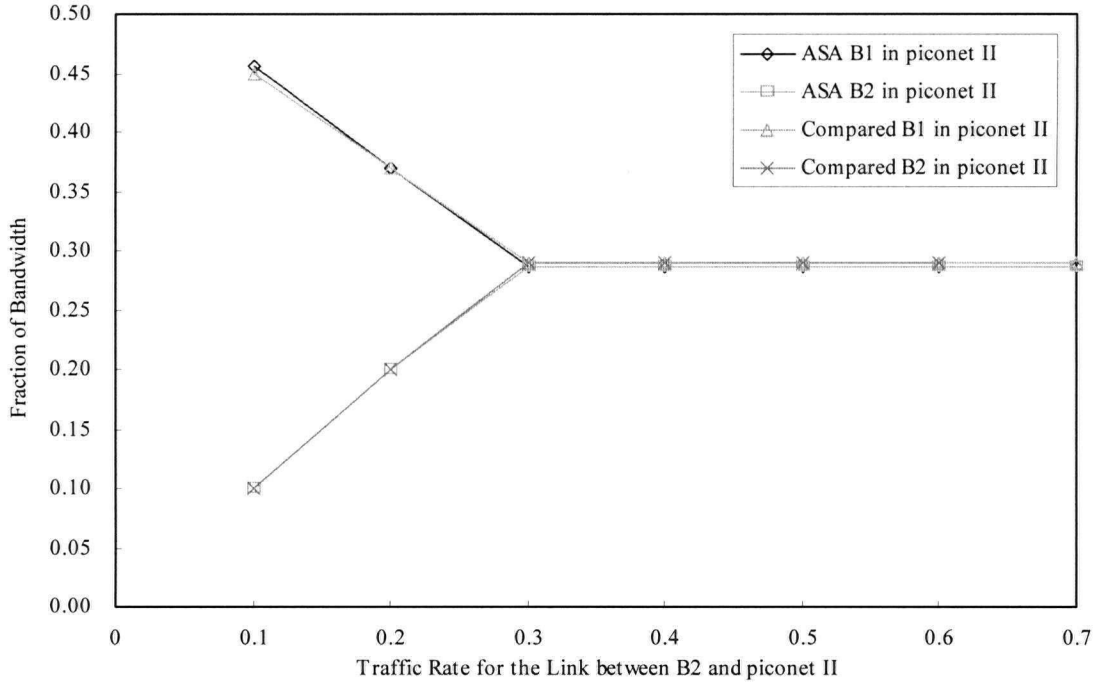

Figure 4.29. Sharing of bandwidth between B1 and B2 in piconet II

All M-S links generate traffic at a rate of 0.2 × MATP. The traffic rate on the link between B1 and piconet I is 0.2 × MATP, the traffic rate on the link between B1 and piconet II is 0.5 × MATP. In addition, the traffic rate on the link between B2 and piconet III is 0.2 × MATP. Lastly, we vary the traffic rate on the link between B2 and piconet II from 0.1 × MATP to 0.7 × MATP to observe the performance change. Figure 4.29 shows the sharing of bandwidth between B1 and B2 in piconet II. The results show that both ASA and the Fair and Traffic Dependent Scheduling Algorithm achieve similar results.

4.5 Summary

In this chapter, we compare the performance of ASA on fairness, throughput, and delay with FSS and CBS. From the simulations on fairness comparison, results show that ASA can achieve the max-min fairness under different traffic conditions. Other than fairness comparison, we also compare the performance of all three schemes with UDP and TCP traffic. We first determined how each scheme performs when traffic is CBR. From the simulations, results show that ASA achieves the highest aggregate throughput and lowest average delay on M-S, M-B-M, S-M-S, S-M-B-M-S traffic when compared to FSS and CBS. We then determine how each scheme performs under bursty on-off traffic. Simulation results show that ASA also achieves the highest aggregate throughput and lowest average delay under different combinations of on-off periods when compared to FSS and CBS. Lastly, we compare the average end-to-end transfer delay for sending a file from one node to another node using TCP. Simulation results show that ASA achieves the lowest average transfer delay on M-S, S-M-S, and S-M-B-M-S traffic. FSS achieves better average transfer delay on M-S-M traffic since it gives higher priority to master-bridge links than master-slave links. However, FSS achieves the highest average transfer delay on M-S and S-M-B-M-S traffic. Therefore, when we consider all types of traffic, ASA still has the best performance for file transfer using TCP. At the end of the chapter, we redo some of the experiments in the paper of the Fair and Traffic Dependent Scheduling Algorithm by using ASA. From the simulations, results show that ASA achieves similar results with the Fair and Traffic Dependent Scheduling Algorithm. Since both schemes focus on max-min fairness, they will allocate bandwidth to each link in a similar manner.

Chapter 5 – Conclusions

We conclude the thesis with a summary of our work and areas for future work.

5.1 Summary

Our work began with a study of piconet scheduling algorithms and scatternet scheduling algorithms in Bluetooth networks. From the study, we found that it is necessary to develop a scatternet scheduling scheme which can maintain fairness for all nodes in a scatternet, avoid bridge node conflict, allocate bandwidth to each link based on traffic condition, and integrate both intra-piconet and inter-piconet scheduling.

- In Chapter 3, we proposed the Adaptive Scheduling Algorithm (ASA) for Bluetooth scatternets. In order to determine the traffic change, a traffic estimator is placed on each node to estimate the packet arrival rate. By checking with the estimated packet arrival rate and the size of the queue, a master maintains an active and waiting lists to organize the serving order of slave nodes within a piconet. Since the master adds the node to the active list according to bandwidth demand, ASA can allocate bandwidth on each link based on traffic condition. Moreover, since the master serves each node in the active list with the same amount of time, ASA can maintain fairness to all nodes within a piconet. In order to organize the meeting time between a master and a bridge node, both master and bridge nodes use the dynamic switch schedule to arrange for a meeting time. Since the master and bridge nodes consider the fairness on their connected links before setting the meeting time, ASA integrates both intra-piconet and inter-

piconet scheduling as a single design module. In addition, ASA can also maintain fairness for all nodes within a scatternet. Lastly, as the switch schedule does not allow a node to schedule for a conflict time, ASA can also prevent the bridge node conflict in a scatternet.

- In Chapter 4, we compared between ASA, FSS, and CBS. Through the simulations, results show that ASA can achieve the max-min fairness under different traffic conditions. Moreover, ASA can maintain a high aggregate throughput and low delay on either CBR or bursty on-off UDP traffic when compared with FSS and CBS. Moreover, the simulations also show that ASA can maintain a small average transfer delay for TCP traffic when compared with FSS and CBS. At the end of the chapter, we compared ASA with the Fair and Traffic Dependent Scheduling Algorithm. The simulations show that both scheme allocate similar amount of bandwidth to each links based on max-min fairness.

5.2 Future Work

In the course of the investigations reported in this thesis, a number of interesting problems have been discovered which merit further research.

- **Packet Collisions Condition:** In our scheme, it uses the hold mode to allow a bridge node to switch between different piconets. Therefore, every time when a master meets with a bridge node, they will negotiate for the next meeting time in order to understand the next holding period. However, if there is a packet loss due to collision, a master may not be able to negotiate for the next meeting time with the bridge node. Therefore, an enhancement of the scheme is to investigate how

the master and the bridge node schedule for the next meeting time in this condition.

- **Low-power Mode:** In our scheme, there are some situations that a node is able to go into low-power mode. When a slave node moves to a waiting list, it can save power by turning into low-power mode. Moreover, when a master or bridge node is not scheduled to serve a connected node, it is also possible for the node to turn into low-power mode. Therefore, another enhancement of the scheme is to take low-power mode as a design factor as well.
- **Interference:** Besides the interference between different piconets, the interference between Bluetooth network and IEEE 802.11 WAN will also be a future design issue for developing a scatternet scheduling scheme.

Bibliography

[1] Specification of the Bluetooth System – Core vol. 2 v1.2, Available at http://www.bluetooth.com.

[2] A. Capone, M. Gerla, and R. Kapoor, "Efficient Polling Scheme for Bluetooth Picocells," in *Proc. of IEEE International Conference on Communications (ICC)*, Helsinki, Finland, June 2001.

[3] D. Yang, G. Nair, B. Sivaramakrishnan, H. Jayakumar, and A. Sen, "Round Robin With Look Ahead: A New Scheduling Algorithm for Bluetooth," in *Proc. of International Conference on Parallel Processing Workshops (ICPPW)*, August 2002.

[4] Bluehoc Simulator, Available at http://www-124.ibm.com/developerworks/oss/bluehoc/

[5] M. Shreedhar and G. Varghese, "Efficient Fair Queueing Using Deficit Round Robin," in *Proc. of ACM SIGCOMM*, Boston, Massachusetts, pp. 231-242, August 1995.

[6] M. Kalia, D. Bansal, and R. Shorey, "MAC Scheduling and SAR Policies for Bluetooth: A Master Driven TDD Pico-Cellular Wireless System," in *Proc. of IEEE International Workshop on Mobile Multimedia Communications (MoMuC)*, San Diego, CA, USA, November 1999.

[7] M. Kalia, D. Bansal, and R. Shorey, "Data Scheduling and SAR for Bluetooth MAC," in *Proc. of IEEE Vehicular Technology Conference (VTC)-Spring*, Tokyo, Japan, May 2000.

[8] P. Johansson, M. Kazantzidis, R. Kapoor, and M. Gerla, "Bluetooth: An Enabler for Personal Area Networking," *IEEE Network*, vol. 15, issue 5, September/October 2001.

[9] P. Johansson, R. Kapoor, M. Kazantzidis, and M. Gerla, "Rendezvous Scheduling in Bluetooth Scatternets," in *Proc. of IEEE International Conference on Communications (ICC)*, New York, NY, USA, May 2002.

[10] W. Zhang, and G. Cao, "A Flexible Scatternet-wide Scheduling Algorithm for Bluetooth Networks," in *Proc. of IEEE International Performance, Computing, and*

Communications Conference, Phoenix, AZ, USA, April 2002.

[11] T. Salonidis, P. Bhagwat, L. Tassiulas, and R. LaMaire, "Distributed Topology Construction of Bluetooth Personal Area Networks," in *Proc. of IEEE INFOCOM'01*, Anchorage, Alaska, USA, April 2001.

[12] N. Johansson, F. Alriksson, and U. Jonsson, "JUMP Mode – A Dynamic Window-based Scheduling Framework for Bluetooth Scatternets," in *Proc. of ACM International Symposium on Mobile Ad hoc Networking and Computing*, Long Beach, CA, USA, October 2001.

[13] S. Baatz, M. Frank, C. Kuhl, P. Martini, and C. Scholz, "Adaptive Scatternet Support for Bluetooth using Sniff Mode," in *Proc. of IEEE Conference on Local Computer Networks (LCN)*, Tampa, FL, USA, November 2001.

[14] S. Baatz, M. Frank, C. Kuhl, P. Martini, and C. Scholz, "Bluetooth Scatternet: An Enhanced Adaptive Scheduling Scheme," in *Proc. of IEEE INFOCOM'02*, New York, USA, June 2002.

[15] L. Tassiulas and S. Sarkar, "Maximin Fair Scheduling in Wireless Networks", in *Proc. of IEEE INFOCOM'02, New York*, USA, June 2002.

[16] A. Raez, G. Miklos, F. Kubinszky, and A. Valko, "A Pseudo Random Coordinated Scheduling Algorithm for Bluetooth Scatternets," in *Proc. of ACM International Symposium on Mobile Ad hoc Networking and Computing*, Long Beach, CA, USA, October 2001.

[17] J. Misic and V. B. Misic, "Bridges of Bluetooth County: Topologies, Scheduling, and Performance," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 2, February 2003.

[18] G. Tan and J. Guttag, "A Locally Coordinated Scatternet Scheduling Algorithm," in *Proc. of IEEE Conference on Local Computer Networks*, November 2002.

[19] R. Kapoor, A. Zanella, and M. Gerla, "A Fair and Traffic Dependent Scheduling Algorithm for Bluetooth Scatternets," *ACM Journal on Special Topics in Mobile Networking and Applications (MONET)*, vol. 9, no. 1, February 2004.

[20] L. H. Shai, R. Kofman, A. Segall, and G. Zussman, "Load-Adaptive Inter-Piconet Scheduling in Small-Scale Bluetooth Scatternets," *IEEE Communications Magazine*, vol. 42, issue 7, pp. 136-142, July 2004.

[21] V. B. Misic, J. Misic, and K. L. Chan, "Improving the performance of Bluetooth piconets with synchronous and asynchronous traffic," in *Proc. of IEEE Globecom'03*, San Francisco, CA, December 2003.

[22] R. Kapoor, L. Chen, Y. Lee and M. Gerla, "Bluetooth: Carrying Voice over ACL Links," in *Proc. of Mobile and Wireless Communications Networks (MWCN),* Stockholm, Sweden, September 2002.

[23] W. Fang, N. Seddigh, and B. Nandy, "A Time Sliding Window Three Colour Marker (TSWTCM)," *IETF RFC 2859*, June 2000.

[24] D. D. Clark and W. Fang, "Explicit Allocation of Best-Effort Packet Delivery Service," *IEEE/ ACM Transactions on Networking*, vol. 6, no. 4, pp. 362-373, August 1998.

[25] "Network Simulator (NS-2)," http://www.isi.edu/nsnam/ns/.

Glossary of Acronyms

ACL	Asynchronous Connection-Less
ASA	Adaptive Scheduling Algorithm
AFP	Adaptive Flow-based Polling
B	Bridge Node
BTCP	Bluetooth Topology Construction Protocol
CBR	Constant Bit Rate
CBS	Credit Based Scheduling
DRR	Deficit Round Robin
ERR	Exhaustive Round Robin
FEC	Forward Error Correction
FH	Frequency Hopping
FSS	Flexible Scatternet-wide Scheduling
HOL	Head-of-Line
HOL-KFP	HOL K-Fairness Policy
HOL-PP	HOL Priority Policy
IS	Idle State
ISM	Industrial Scientific Medical
K-limited RR	K-limited Round Robin
K-Look-Ahead RR	K-Look-Ahead Round Robin
LAA	Load Adaptive Algorithm

LARR	Look Ahead Round Robin
LCS	Locally Coordinated Scatternet Scheduling Algorithm
LWRR	Limited and Weighted Round Robin
M	Master Node
MATP	Maximum Aggregated Throughput in each Piconet
MDRP	Maximum Distance Rendezvous Point
MP	Maximum Priority
MQS	Max Queue Size
MTS	Max Time-share
MUSS	Maximum Usable Serving Slots
PCSS	Pseudo-Random Coordinated Scatternet Scheduling
PRR	Pure Round Robin
RP	Rendezvous Point
RR	Round Robin
RW	Rendezvous Window
S	Slave Node
SCO	Synchronous Connection-Oriented
SS_Slot	Switch Schedule Slot
Sticky-AFP	Sticky Adaptive Flow-based Polling
TC	Time Commitment
TCP	Transmission Control Protocol
TDD	Time Division Duplex

TRAT	Transmission Request Arrival Time
TSW	Time-sliding Window
UDP	User Datagram Protocol