Bluetooth QoS Scheduler

Björn Jonsson

Master's Thesis in
Teletrafic Systems
The Royal Institute
of Technology,

Stockholm, Jan 2003

Examiner: Prof. Gerald Q. Maguire Jr.
Royal Institute of Technology, Stockholm.
Department of Microelectronics and Information Technology.

Supervisor: Niklas Johansson
Ericsson Research, Kista
Department of Wireless Access Networks

Abstract
Bluetooth is a low cost, short-range radio technology that enables electronic devices to communicate wirelessly via ad-hoc networks. Different kinds of applications may run over these networks, some of these applications with particular Quality of Service (QoS) requirements. One such application could for instance be an Audio application which need data frames to be delivered at regular interval to work properly.

The communication over Bluetooth links is totally controlled by one of the participating nodes. Since this node controls the traffic and thereby controls the QoS within the network its communicating behaviour is of great importance for maintaining QoS reservations. A central part in maintaining QoS reservations is the scheduler, which prioritises between different tasks to be done such as which device that is allowed to transmit next.

There is no standardised way to implement the Bluetooth scheduler. However, a Bluetooth scheduler should be able to handle QoS reservations, maintain fairness among the participating nodes, and utilise the available bandwidth efficiently. Furthermore a key requirement in Bluetooth is simplicity, therefore a Bluetooth scheduling algorithm should be of low complexity.

This master thesis presents a new patented solution for an intra-piconet scheduling algorithm that is capable of maintaining QoS reservation, while being fair and also meet the low complexity requirement. This algorithm is presented in detail in the thesis and has been implemented in an ns2-based Bluetooth simulator. Simulations in this simulator are presented that verifies the scheduling algorithms functionality.

Sammanfattning
Bluetooth är en standard för trådlös ad-hoc kommunikation. Denna standard definierar i antalet noder begränsade nätverk (piconet) som kan kopplas samman till större nätverk (scatternet). Många olika typer av applikationer kan tänkas använda dessa nätverk, vissa av dessa med speciella krav på Quality of Service (QoS). Ett exempel så en dylik applikation är ljudöverföringar som kräver att data överförs med jämna mellanrum för att kunna fungera tillfredställande.

Kommunikationen över ett Bluetooth-piconet kontrolleras helt av en av de deltagande enheterna. Eftersom denna enhet kontrollerar trafiken kontrollerar den också upprätthållandet av QoS reservationerna. En central del i detta upprätthållande är schemaläggaren, som är den del i en Bluetoothenhet som prioriterar mellan olika saker som ska utföras. Ett exempel på en sådan prioritering kan vara vilken enhet i nätverket som ska få kommunicera härnäst.

Bluetooth standarden beskriver inte hur denna schemaläggare ska implementeras. Grundläggande krav på en Bluetooth schemaläggare är att den ska kunna hantera och upprätthålla QoS reservationer samtidigt som den delar den befintliga kapaciteten på ett rättvist och effektivt sätt mellan de deltagande enheterna. Vidare bör den ha en låg komplexitet eftersom enkelhet är ett krav på alla delar i ett Bluetoothsystem.

Detta examensarbete presenterar en ny patenterad algoritm för schemaläggning i begränsade Bluetoothnät som är kapabel att upprätthålla QoS-reservationer, samtidigt som den distribuerar den tillgängliga kapaciteten på ett rättvist sätt. Algoritmen möter också upp till kravet på låg komplexitet. Rapporten innehåller vidare simuleringar av algoritmens funktion, dessa har utförts i en ns2-baserad Bluetooth miljö och verifierar algoritmens funktionalitet.

Acknowledgements

Ericsson
I would like to thank my industrial supervisor Niklas Johansson at Ericsson Research in Kista, as he has been my greatest source for understanding the subject and a good guide who has helped me to stay on track towards the goal.

Other people I would like to thank at Ericsson research in Kista are Fredrik Alriksson, Ulf Jönsson, and Tony Larsson for their support during the implementation phase of my work.

Furthermore I want to thank Jan Åberg and Martin Van der Zee for their answers to Bluetooth related questions.

KTH
I also wish to thank my educational supervisor and examiner Prof. Gerald Q. Maguire Jr. from the Royal Institute of Technology, Stockholm at the Department of Microelectronics and Information Technology for his sometimes unbeliably rapid respones that have helped me in making the report what it is.

Table of Contents

List of Figures

List of Tables

1 Introduction

The number of devices in our surroundings equipped with data- and telecommunication capabilities is constantly increasing. At the same time we can see an increased demand for mobility and portability by the users of these devices. When carrying several portable devices it is often desirable to limit the redundancy among them, one way of doing that is to interconnect them, thus enabling them to utilise services from each other. Users would like to have ubiquitous and pervasive service, capable of both voice and data transmission since then all users get service whenever and wherever they want. A further step in making mobile services more accessible for the users is to provide the service without: using cables, having large battery packs, requiring the user do any manual connection configuration, or explicitly going through different access technologies. The number of solutions for these problems enabling networks connecting devices in a personal operating space is limited.

In the last few years much attention has been given to enabling research and development of such personal networks, often denoted as Personal Area Networks (PANs). These networks are composed of portable personal devices, such as cellular phones, Personnel Digital Assistants (PDAs), and laptops, in close proximity to each other with possibility to communicate within the networks. Bluetooth is one such emerging PAN technology replacing cables and enabling ubiquitous and pervasive communication between devices connected wirelessly via short-range ad hoc networks [1], [2], [6]. Since its announcement in late spring 1998, Bluetooth technology has attracted a lot of attention and a great number of papers have been published [2], [6], [10], [11]. When developing the Bluetooth standard both low cost and low power have been explicit requirements enabling the technology to be installed in virtually any electronic device. Initially Bluetooth was designed to replace cables between portable devices, today a much more widespread use of the technology is envisioned with products enabling interconnection between individuals (e.g. phones), computing devices (e.g. PDAs, Laptops, and printers), and machines (so called embedded solutions)

In its simplest form a Bluetooth network resembles a single cell in a cellular network, with one central unit in charge of the capacity allocation and several units associated with this central unit. In Bluetooth such a cell is denoted a *piconet*, where the central unit is denoted *master* (equivalent to the base station controller in a cellular network) and the units associated with it are denoted slaves (equivalent to the user terminals in a cellular network). Multiple piconets with overlapping coverage area may form larger networks called *scatternets*. Units participating in scatternets can only be active in one piconet at a time, making them unreachable for masters in other piconets (unless the units are equipped with multiple radio interfaces). The slaves scatternet activities, current radio conditions, and amount of data available for transmission are not completely known by the master, which consequently has incomplete knowledge about the slaves regarding (instant) reachability and their current need of capacity. Therefore it is very important how the master allocates the capacity (i.e. prioritises between different tasks) within the network since this determines the system performance, especially when guaranteed traffic is considered, since a guaranteed service is delivered when there has been an agreement that a certain service level will be upheld. The service level can be specified with respect to delay and bandwidth. Traffic relying on these guarantees is often denoted Quality of Service (QoS) traffic. The Bluetooth specification has support for QoS in the form of a mechanism to negotiate a set of parameters for each connection.

QoS reservations are useful whenever an application has particular requirements on the traffic. For instance, a real time video application will need video frames delivered at regular intervals and frames that are delayed too long are no longer useful. Another example is a wireless mouse or a keyboard that needs low bandwidth and low latency, but where data is always useful even if delayed. There are also many applications that need bandwidth reservations in order to work properly.

Consequently the aim of this masters thesis project is to further refine Ericsson´s capacity allocation algorithm[1] for Bluetooth by enhancing it with Quality of Service (QoS) capabilities. The scheduler, which is responsible for maintaining the QoS (reservations) prioritises between different tasks to be done, e.g. which device to transmit to or receive from. The algorithm must take into consideration other aspects than QoS such as limited processing resources, varying radio conditions, and at the same time efficiently utilise the available bandwidth in a fair way.

[1] Henceforth, this will be called the scheduling algorithm.

2 Background

The Bluetooth standard was initially been developed by a number of leading telecommunication, computing, and digital signal processing companies. Five companies (Ericsson, Nokia, IBM, Toshiba and Intel) originally formed the Bluetooth[2] Special Interest Group (SIG) in 1998. Today there are four additional promoter members contributing to the development of the standard: 3Com, Agere, Microsoft, and Motorola. Over 2000 other companies have joined the group as associate or adopter members. In 1999 the SIG work resulted in the first Bluetooth standard, version 1.0, and the current release is version 1.1.

The SIG has formed working groups each trying to include more functionality in the specification, via expert groups, study groups, and improvement groups. The three groups are supposed to come up with new recommendations within certain areas. The QoS study group aims to define new QoS enhancements in Bluetooth. The SIG has also defined qualification procedures for testing of products that are to use the Bluetooth trademark. This ensures interoperability between different manufactures.

The key issues when developing the Bluetooth specification are depicted in Figure 1 and further explained in this section. Bluetooth should be usable all around the world, i.e. either via a single radio or reconfiguration of a single radio. Further, Bluetooth should be manufactured at a low cost, resulting in a low price for the end user. The Bluetooth chipsets have a target price of $5, the current price is constantly decreasing and predicted to reach the target price this year [18]. The product must be small in size, thereby making it possible to integrate Bluetooth into almost any portable and non-portable device imaginable. Bluetooth is designed to have a short-range radio interface, which is to be used in a person's operating space, i.e. around 10 meters. Further it should work in an ad hoc fashion, i.e. work as a network without a predefined infrastructure, where nodes can come and go as they like and where there are no differences between the units in the system (even if the units can play different roles in the network). For example, there should be no distinction between base stations and terminals as there are in a cellular telephone network. Since portable devices have limited power resources, Bluetooth is designed to have low power consumption. The standard should be a robust wireless technology replacing cables and for connecting portable devices.

Today Bluetooth can be found in a vast range of products such as mobile phones, printers, PDAs PCMCIA cards, access points, headsets, laptop computers, cars... The trend within portable computing and communication devices such as mobile phones, laptops, and PDAs is an aggregation of multiple functions into the same device and at the same time a reduction in size. However, all integration is not beneficial since smaller, multi-use devices may not be optimised for specific tasks, but instead are designed to work reasonable well for many applications at the cost of optimisation. Bluetooth provides us with the possibility to retain multiple devices and hence they may each be optimised for their own purpose while still being connected almost as if they were one device.

Because this connection is done wirelessly, Bluetooth removes the need for many cables, for example instead of connecting a wire from your laptop to the printer at home or at the office a Bluetooth equipped printer will be detected and connected to your laptop when you are in range. Many car manufacturers are currently developing cars with built in Bluetooth. The idea is to have a headset built in the car and utilise for instance the car radios display, this would remove the need for taking the mobile phone out of your pocket when entering the car. Further, the user's mobile phone could also be used for wireless downloads of traffic information and send it via Bluetooth to the car radios display. The use of telematics using Bluetooth will also allow the components of the car to speak to the driver or via the mobile phone (automatically) prepare the repair shop for what needs to be done the next time the car is serviced. Instead of using cables to connect all sorts of mobile devices

Embedded solutions will not only be seen in products such as mobile phones and PDAs, for example ABB uses Bluetooth in some of their robots, thus making them more robust since it removes the need for fragile cables. In the future any machine, static or moving, located indoors or out could have embedded wireless solutions. Possible candidates for embedded Bluetooth solutions are for instance monitoring devices (e.g. Healthcare monitors or security cameras), vending machines, cargo containers and fork lifts.

[2] The fact that the Bluetooth standard unites different kinds of devices and that part of the work with Bluetooth has been done in Scandinavia explains the name "Bluetooth" which refers to a Danish Viking King, King Harlad Blåtand (in English Bluetooth), who united part of Scandinavia during the 10th-century.

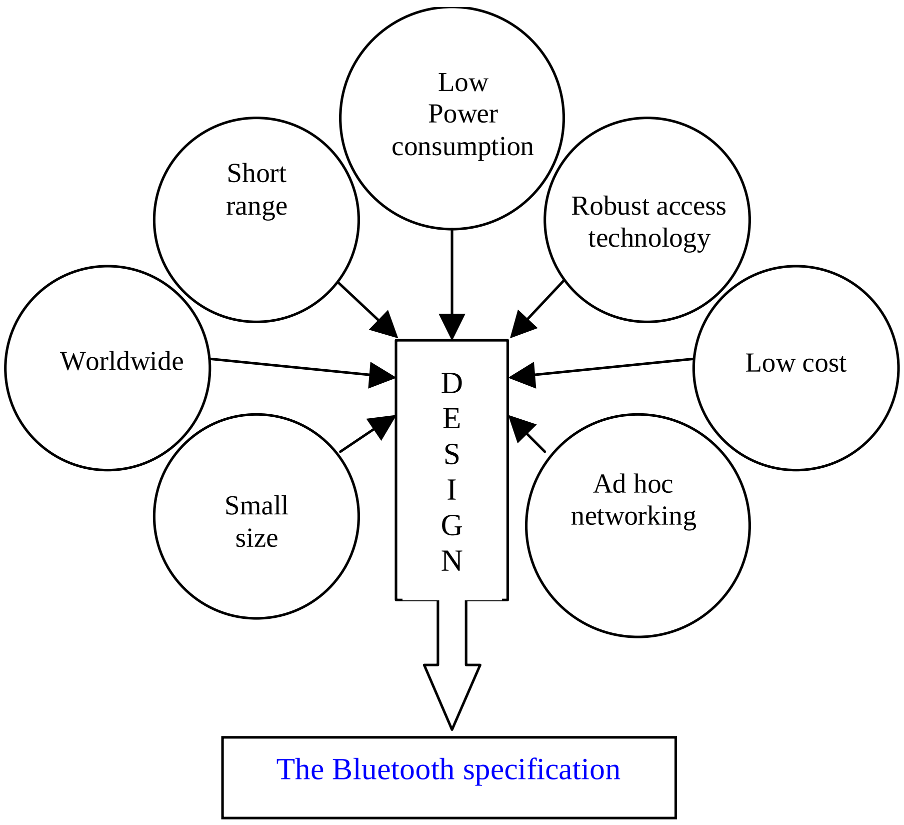


Figure 1. The Bluetooth specification design issues

3 Technical Overview

This section describes the Bluetooth technology [1]. The Bluetooth protocol stack is depicted in Figure 2. The radio is the medium over which all Bluetooth communications is performed. The baseband functions include, among other things, packet transmission, timing, flow control, and error detection and correction. Further it is capable of establishing two different types of physical links: Synchronous Connection-oriented (SCO) and Asynchronous Connection-less (ACL) links. The link manager protocol (LMP), manages link related issues such as polling, power control, link supervision, QoS, and packet type selection. The Logical Link Control and Adaptation Protocol (L2CAP), provides a data link with function for segmentation and reassembly, protocol multiplexing, and QoS negotiation. The reason for having QoS negotiation in L2CAP while the QoS maintaining functionality is via the LMP protocol is because L2CAP provides the data link between units while LMP is local to each device. The network layer is not specified in the Bluetooth standard, however, when routing in scatternets a network adaptation layer is introduced in-between the L2CAP and the network layer. The Bluetooth radio, Baseband, LMP, and L2CAP are grouped together in a Host controller while the higher layers generally run on the host. In-between these two Bluetooths implement a Host controller interface (HCI). The HCI provides connectivity between the application and the Bluetooth device. More detailed information about the different layer’s capabilities are given later in this section.

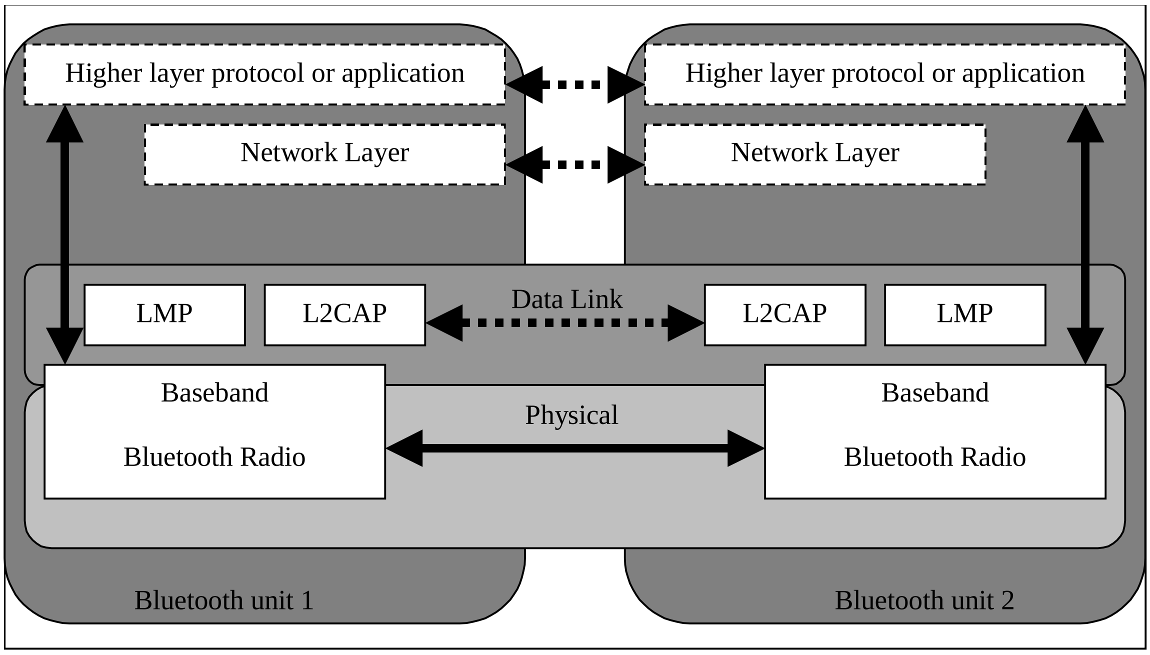


Figure 2. Bluetooth protocol architecture

3.1 General description

Bluetooth provides both point-to-point links and point to multi-point connections. When 2 or more units connect and share the same channel they form a **piconet**. A piconet consists of 1 master and up to 7 slaves. As described before any Bluetooth device can be the master of the piconet. The current channel and the frequency-hopping scheme used in the piconet are derived from the masters' clock and MAC address. In order for all slaves to be synchronised with the master they add an offset to their own clock to track the master's clock. Hence a node can only be master in one piconet. The master clock is exchanged with the slaves during set-up of the link and updated regularly. Figure 3 further illustrate this via an example of a Bluetooth piconet with 1 master and 5 slaves with their respective clock offsets. The major reason for this partitioning of the users into piconets and limiting them to groups with a maximum of 8 devices is due to the fact that the ISM band is unlicensed and has several unpredictable sources of interference such as microwave ovens, garage door openers, and WLANs. Dividing the network into groups, which hop incoherently minimises channel interference from sources outside of the piconet and at the same time the ISM band can be utilised more efficiently than a system that only has one large 79Mhz channel. Multiple piconets can coexist and thus increasing capacity in a region. To quantify this, an example is given. Consider 50 units equally sharing the same 1 MHz channel, then the theoretical bandwidth/user would only be 20 kHz, i.e. at 1b/Hz this would be 20 kbps each with an aggregated throughput of 1Mb/s.

On the other hand if the fifty units were divided into independent groups of 5 units each, then the theoretical throughput/unit would be 200 kbit/unit and the aggregated throughput would be 200* 50 =10Mbit/s. These calculations assume that all units are always ready and waiting to transmit, that the channel is shared equally among the slaves, and that no interference is present (i.e. the groups are independent). This later is of course false when multiple devices in different piconets hop to the same frequency channel at one time. Further arguments for the partitioning of the channel into smaller networks is that, the overhead in addressing units is lower (for piconets with maximum of 8 units only 3 bits are needed), within a radius of 10 meters the number of handheld electronic devises is not likely to be many (as Bluetooth initially was designed for a 10 meter range). Moreover making narrowband radio transceivers with a high immunity to interference helps to keep down the price of Bluetooth chips. The method that Bluetooth use to solve the interference problem is called fast frequency hop spread spectrum (FFH) technique and is further described in section 3.2.

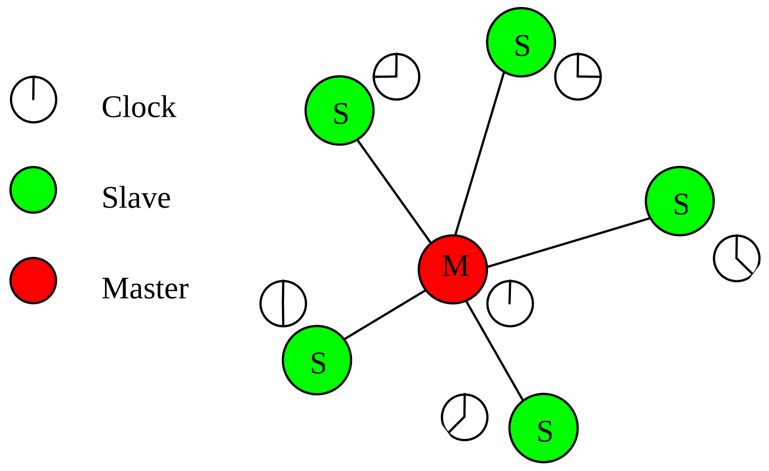


Figure 3. A piconet example consisting of one master and five slaves with their respective clock offset shown

Several piconets can be established and linked together in an ad hoc fashion, thus forming a so-called scatternet in which each piconet is identified by different frequency hopping sequences. **Scatternets** are formed when a Bluetooth unit participates in more than one piconet at the same time, thus functioning as a bridge. However, a Bluetooth unit cannot be a master in more then one piconet. This is because a piconet is defined by its hopping sequence (as long as the sequences are uncorrelated this enables more than one piconet to share the same spectral band, thus providing the independence of piconets from one another) and that sequence is derived from the master's clock. So two piconets with the same master have the same hopping sequence, i.e. they are not two different piconets, hence only one piconet exists under these conditions. In order to prevent traffic from using twice as much bandwidth as necessary when traffic goes from slave 1 to slave 2 via the master, it is better to set up a direct connection between slave 1 and slave 2 by making one of them the master of a new piconet and linking the two piconets together in a scatternet. A unit that is participating in multiple piconets will in this thesis be called a Participating in Multiple Piconets (PMP) unit. Figure 4 extends Figure 3 to show 3 interconnected piconets.

When sending traffic in a given channel Bluetooth uses Time Division Duplex (TDD) multiplexing of the channel. The channel is slotted into 0.625ms slots during which packets are exchanged. The TDD concept means that the master sends a packet to a slave that **must** respond in the following time slot.

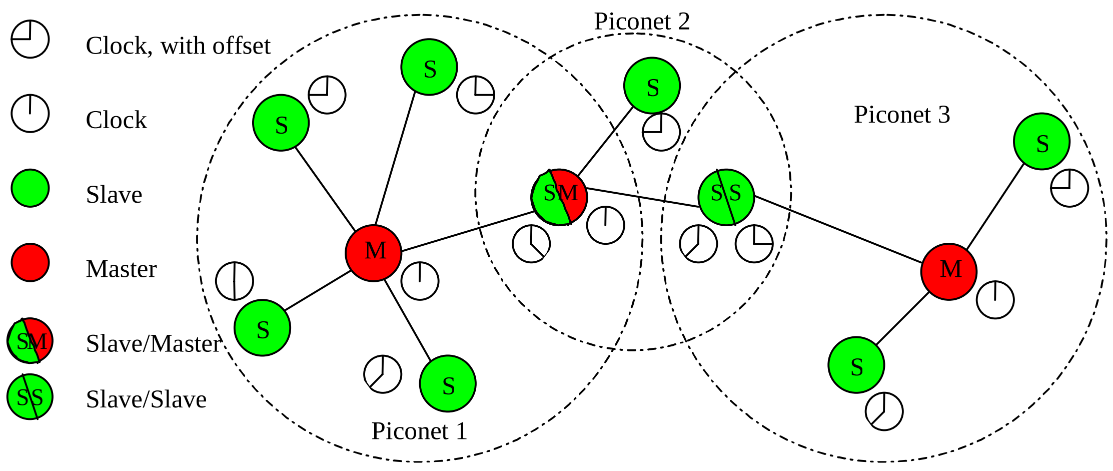


Figure 4. Example scatternet consisting of 3 interconnected piconets. Note that nodes participating in several piconets have to keep track of more then one clock offset, one per piconet (Unless they are a master). The different masters clocks are NOT synchronised.

Both the master and the slave are allowed to use packets having the duration of 1,3, or 5 time slots, see section 3.3. The reason for only using an odd number of slots is to stay synchronised, thus the master always start transmitting in an even numbered slot and the opposite holds for the slaves. The fact that different piconets are not synchronised will, when switching between two different piconets, lead to the loss of at least one TDD frame. This is because a unit in either of the piconets has to have to wait at least one TDD frame in order to switch to the next frequency of one piconet and then switch back again. This is depicted in Figure 5 where a slave starts in piconet 1 and moves to piconet 2 and then back again after some time.

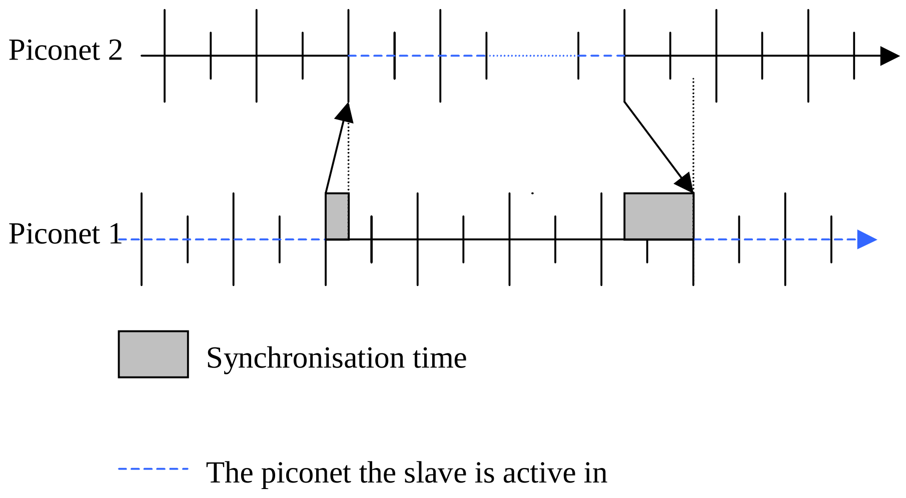


Figure 5. Lost transmission time when moving between piconets.

3.2 Bluetooth Radio

Bluetooth was designed to be used globally, this lead to the use of the unlicensed 2.4 GHz Industrial-Scientific-Medical band (ISM) for Bluetooth transceivers. The spectrum 2400 – 2483.5 MHz that Bluetooth operates in is divided into 79[3] radio channels each with a 1 MHz bandwidth and two guard bands of 2 and 3.5 MHz bandwidth. Further as previously stated the 2.4 GHz ISM band is unlicensed and globally available so there is a lot of interference originating from different sources such as WLANs and Microwave ovens, therefore it is necessary for the Bluetooth technology to have a mechanism to reduce the impact of such interference. The Bluetooth solution to this problem is that the channel to be used is determined by a frequency-hopping scheme, which divides the frequency band into several hop channels. During a connection, radio transceivers hop from one channel to another in a pseudo-random fashion with a new frequency every 0.625 ms[4]. If a range of frequencies is being interfered with, then for a period of time the lost slots will, thanks to the frequency-hopping scheme, be limited and randomly distributed over all possible master-slave connections. This distribution effect only occurs if we are looking at a long time period, for a bounded (i.e. a limited number of hops) this may not be the case. It should also be noted that Bluetooth utilises other mechanisms to reduce the impact of air interference such as error correction schemes, and acknowledgements/retransmissions (see section 3.3), in addition the limited transmission range also reduces the occurrence of interference. However, Bluetooth does not avoid bands of interference it just has to loose packets through each such hop, unlike ODFM, which reduces the rate in channels with interference. The transmitted signal is modulated with a Gaussian Frequency Shift Key (GFSK), which gives a symbol rate of 1Msymbols/s.

[3] In some countries (France and Japan) the ISM band is limited, and hence the number of Bluetooth channels reduced to 23.
[4] When multi slot packets are used the hop interval is changed, this is discussed further in section 3.3.

3.3 Baseband

The Bluetooth baseband provides transmission capabilities based upon the physical radio layer and defines key features such as packet types, error handling, link types, and different low power modes for the units. These features will be discussed in this section.

The standard packet format used in Bluetooth is shown in Figure 6. A Bluetooth packet consists of an access code, a header, and the packet payload. The access code comprises of the 72 first bits of the packet and is derived from the master's identity. Every packet exchanged on the channel is preceded by this access code, which is unique for this channel. This uniqueness is used to separate valid packets on the piconet form invalid packets as each recipient of an incoming signal compares it with the expected access code. If the two do not match, then the rest of the signal content is ignored. Besides identification the access code also contains information used for synchronisation and DC offset compensation.

The access code is followed by the packet header, which consists of 6 fields containing control information as depicted in Figure 6. The fields are:

AM_ADDR	The 3-bit local address used for identifying slaves within a piconet in both master to slave and slave to master traffic. Note that all traffic originating from the slaves within the piconet is sent to the master, therefore the master does not need a link local address. The all zero address is used for ACL broadcast messages from the master to the slaves. Apart from this local address each Bluetooth unit has a unique Bluetooth Device Address (BD-ADDR). This unique address is used for link set-up, security, and identification purposes.
TYPE	Specifies which type of packet this is, see Table 1.
FLOW	The flow control in Bluetooth is supported by this one bit flow control flag; that indicates whether a unit can receive more data or not, i.e. whether the receiving buffer is full or not. At the transmitting side this resembles a STOP and GO protocol, i.e. indicates whether the transmitter is allowed to send more packets or not.
ARQN	The retransmission scheme used in Bluetooth is supported by this one bit acknowledge indication. It is used to inform the source of the success of last data transmission[5]. This positive and negative acknowledgement resembles a software ARQ protocol.
SEQN	A one-bit sequence number, is used to detect duplicate packets.
HEC	Header checksum, is used to ensure header integrity.

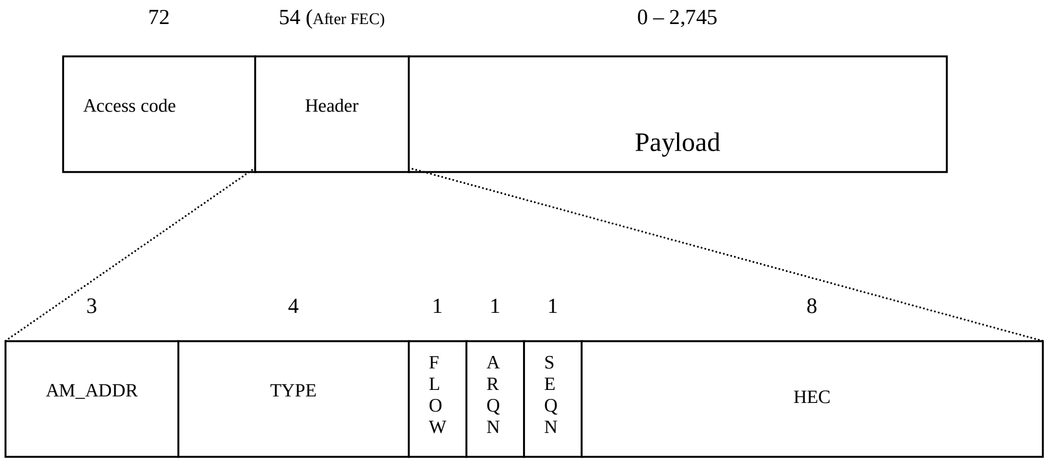


Figure 6. The Bluetooth standard packet, the packet header is enlarged, and the number of bits in each section is indicated.

[5] See Table 1 for packet types affected by the ARQN protocol.

To protect the header against errors a 1/3 Forward error correction (FEC) code is appended resulting in a 54 bit long header (this is described later in this section). The header may or may not (depending on the packet type) be followed by the payload which size can range from 0- 2745 bits. The payload of data packets is preceded by a payload header (i.e. all ACL packets and the DV packets contain a header, except for control packets). A field in this header indicates whether the packet is the start of a new L2CAP (see section 3.4) packet or a continuation of a fragmented one. There is also a FLOW indicator, controlling the flow on the L2CAP level. The packet types available in Bluetooth can be divided into four groups as depicted in Table 1, these details can be found in the Bluetooth specification [1]. The first group contains control packets, the second single slot packets, the third and fourth are packets with duration of 3 respective 5 slots. The control packets are common to both Bluetooth link types (SCO and ACL), see further down in this section. The POLL and NULL packet are sent without payload. A POLL packet is sent from the master to a slave who must respond with a packet in the next timeslot. If a polled slave does not have any data to send in reply to the master, when addressed, it sends a NULL packet. The NULL packet is used for exchanging link information with the sender, such as the success of last transmission and if the unit is prepared to receive more data, it can also be interpreted as a sign that the sending slave does not have any data to send. This can be utilised in a scheduling algorithm as a prediction of the slave's buffer status. The ID packet is used in the page and inquiry processes (the Bluetooth neighbour discovery process further explained later in this section) to identify Bluetooth units, and it contains the Bluetooth unit's access code.

Group	SCO link	ACL link	User payload	FEC	CRC	Symetric Max. rate [kb/s]	Asymetric Max Rate [kb/s]	
							Forward	Reverse
1 Control packets	NULL	NULL	Na	No	No	na	na	na
	POLL	POLL	Na	No	No	na	na	na
	FHS	FHS	18	2/3	Yes	na	na	na
	DM1	DM1	0-17	2/3	Yes	108.8	108.8	108.8
	ID packet	ID packet	Na	Na	Na	na	na	na
2 Single slot packets		DH1	0-27	No	Yes	172.8	172.8	172.8
	HV1		10	1/3	No	64	na	na
	HV2		20	1/3	No	64	na	na
	HV3		30	2/3	No	64	na	na
	DV		10 +(0-9)	2/3 (on data part)	Yes (on data part)	64+57.6 (data)	na	na
		AUX1	0-29	No	No	185.6	185.6	185.6
3 Three slot packets		DM3	0-121	2/3	Yes	258.1	387.2	54.4
		DH3	0-183	No	Yes	390.4	585.6	86.4
4 Five slot packets		DM5	0-224	2/3	Yes	286.7	477.8	36.3
		DH5	0-339	No	Yes	433.9	723.2	185.6

Table 1. The Bluetooth packet types

The Frequency Hopping Synchronisation (FHS) packet is a Bluetooth control packet used for exchanging information for the page and inquiry process (described later in this section), the master slave switch (when the node role in the piconet is changed), and clock synchronisation between the master and the slaves.

The other packets are optimised [1] for voice, data, and combined voice data transmissions over SCO and ACL links. The most interesting feature for a scheduling algorithm is their lengths, which are 1, 3, or 5 time slots. A 5 slot packet give a better utilisation of the channel since the packet overhead is less compared to the overhead of smaller packets (1, and 3 slot packets). There is also a difference in robustness between packets with different slot sizes, as the robustness is increased at the cost of reduced bandwidth, i.e. smaller number of sequential slot times. The difference between DMx (Data Medium rate) and DHx (data high rate) packets is the difference in potential data rate and error detection and correction capabilities. DHx packets can contain larger payloads compared to DMx packets of same slot size, but at the cost of reduced protection against errors.

To protect data packets against error there are two Forward Error Correction (FEC) codes used in Bluetooth. The first code is used for protection of the packet header. It is a 3-bit repetition code (i.e. each bit is transmitted 3 times) [14], denoted 1/3 in Table 1. The second code is used in DMx packets data fields and is a shortened (15,10) Hamming code [14] (i.e. five parity bits are added to each block of 10 bits), denoted 2/3 in Table 1. The data packet's payload are protected by a Cyclic Redundancy Check (CRC) code [23].

There are cases when the error detection and correction schemes alone not are satisfactory, for instance if a packet is lost or corrupted the sender must be noticed in order for a retransmission to take place. This is enable through the ARQ scheme, which is supported for both data and data/voice packets, the receiver determines whether the payload was received correctly or not and then sends an acknowledgement (or gives a negative acknowledgement) to the sender in the next transmitted packet. Unsuccessful transmissions will (i.e. if a negative acknowledgement is received) be retransmitted until a successful transmission is made or a time out is exceeded. This timeout is set by the flush timeout parameter for a timer that was set upon the first transmission of the packet. A packet is discarded if this timer expires. The flush timeout timer's function is depicted in Figure 7.

In order to (at the baseband level) determine if a packet was lost a one-bit sequence number (SEQN) was introduced. If the sequence number has a value other than expected (comparing the new number with the last) the data is regarded as incorrect and will be discarded and a negative acknowledgement (NACK) will be sent in the next transmission.

Regardless of whether a sent packet is acknowledged or not, the frequency-hopping scheme described in section 3.2 changes frequency after every slot, which means that Bluetooth makes 1600 frequency hops/s. The exception is when a slave uses multi-slotted packets, in this case the frequency is not shifted until after the transmission of the complete packet at which time it is shifted retroactively, i.e. if it was a 3 slot packet, then there is a shift of three frequency hops. This is depicted in Figure 8 where the three different packet sizes and the corresponding frequency hops are shown.

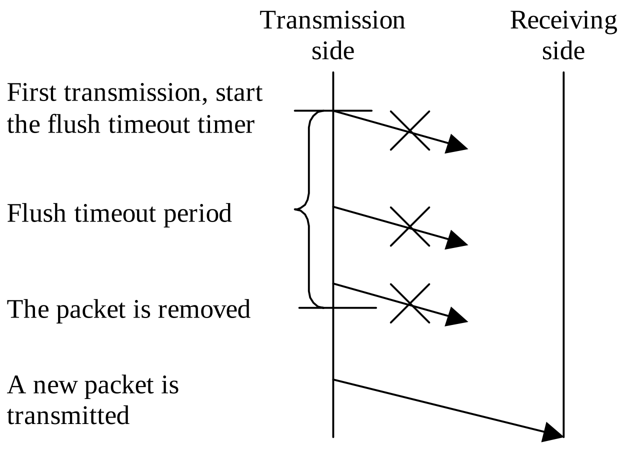

Transmission side
Receiving side
First transmission, start the flush timeout timer
Flush timeout period
The packet is removed
A new packet is transmitted

Figure 7. The Bluetooth flush timeout period

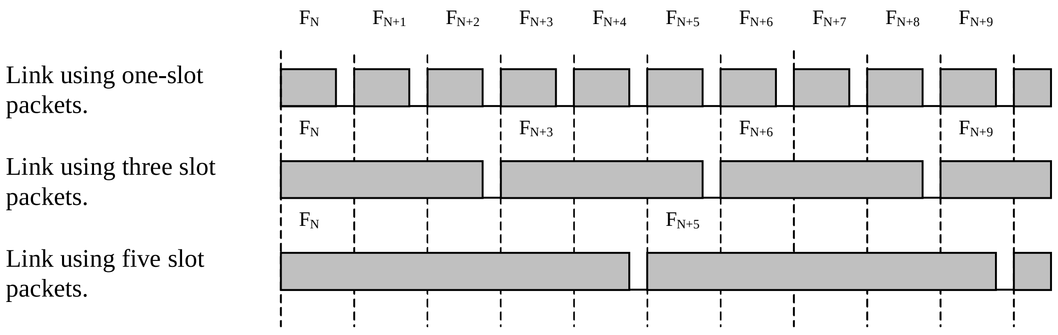

F_N
F_{N+1}
F_{N+2}
F_{N+3}
F_{N+4}
F_{N+5}
F_{N+6}
F_{N+7}
F_{N+8}
F_{N+9}
Link using one-slot packets.
Link using three slot packets.
Link using five slot packets.

Figure 8. Frequency shifts using 1, 3, and 5 slot packets. Note that the master and slave do not need to use the slot sizes symmetrically, for instance the master could use a 1 slot packet and the slave a 3 slot packet.

As mentioned earlier and stated in Table 1, different packet types are used on the two different physical links that are supported by Bluetooths baseband: Synchronous Connection-Orientated (SCO) and Asynchronous Connection-Less (ACL) links.

The SCO link emulates a circuit switched point-to-point link where slots are reserved for transmission at a fixed interval. This type of service is intended for applications with fixed delay or bounded delay traffic, i.e. services such as voice transmission. SCO links are used for real time voice traffic and the transmission rate is fixed at 64kb/s. The master sends SCO packets to the slave in the agreed slots and the slave responds with SCO traffic in the next slot, only single slot packets are used (see Table 1). The SCO slots are scheduled at a fixed time interval and this traffic has precedence (explained further in section 4.2) over other traffic types. This makes the scheduling of the SCO traffic easy for the scheduler.

The ACL link on the other hand is a packet switched link where the master and slave exchange packets in a less strict manner. A slave is only allowed to transmit in the slot(s) directly after the slot when the master has addressed it, i.e. after the master has polled the slave. The polling process on ACL and SCO links are depicted in Figure 9 where the master sends packets/Polls the slaves followed by their responses in the following slot. The figure also shows an example of mixing SCO and ACL traffic and shows the fact that different packet types are used for the ACL traffic (1,3, and 5 slot packets are allowed). The master also has the option not to poll anyone resulting in an idle period for the piconet until the master starts polling the slaves again. The master can also send ACL packets to multiple slaves, i.e. ACL can be used for broadcasting. However, the slaves are only allowed to respond to their master's transmission if addressed by the master, since the ACL broadcast packets are not addressed to a specific slave, i.e. the slaves are not allowed to respond.

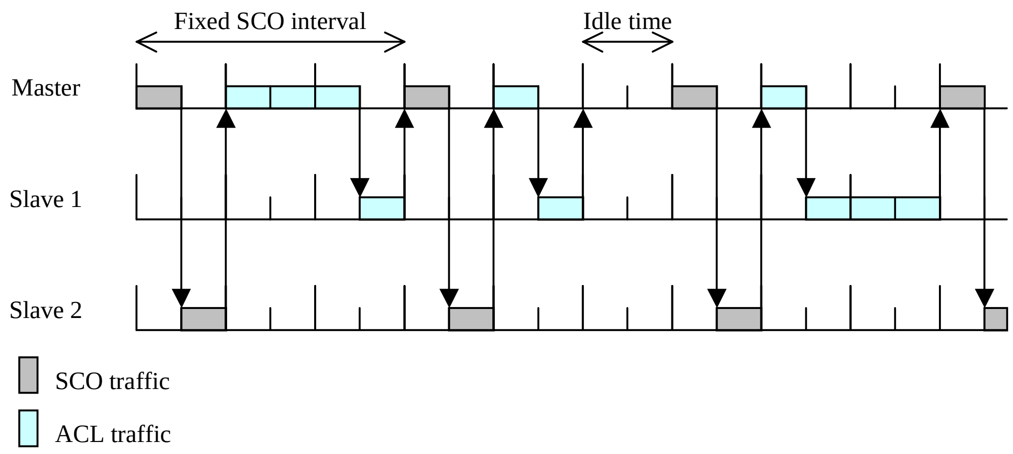


Figure 9. Example of mixing SCO and ACL traffic

The bandwidth on an ACL link is negotiated (as opposed to the fixed bandwidth on SCO links) during the set up of the link (it can be dynamically renegotiated). Setting a minimum polling frequency for each of the slaves ensures the negotiated bandwidth. The duration of the polling interval is stated in the parameter T_{poll}, which is further discussed in sections 3.4 and 4.2.

A master driven system such as Bluetooth removes the need for a collision-detecting scheme, but this comes at the cost of polling overhead. If the master does not have any data to send to the polled slave the result will be wasted bandwidth, similarly if a slave addressed by the master does not have data to send it is also a waste of bandwidth. Since the master decides when each slave is allowed to send data there is no contention sources and two slaves (within the same piconet) cannot transmit at the same time. If a there is a communication between master and slave where no data (or other useful information) was exchanged, i.e. there was a POLL/NULL sequence, no only is bandwidth wasted but so is energy.

The Bluetooth specification supports a set of low power modes used by nodes in order to reduce the power consumption when there is no need to transmit data for a period of time. These modes can also be utilised in a scatternet or when the master has more then 7 slaves that want to be part of the piconet. In theses modes the slaves have different levels of reduced activity and different types of connections to the piconet. In order for the Bluetooth units to return (from some of the low power modes) to the active mode, i.e. fully connected mode or when the initial connection between master and slave is made a procedure called **page** has been introduced. Before the units can establish a connection (i.e. perfom paging) they must be aware of each other, to discover other Bluetooth devices in range an **inquiry** procedure is performed, i.e. inquiry procedure is the neighbour discovery process of Bluetooth. The order of establishing a connection and the typical times for doing so is depicted in Figure 10.

The set of power saving modes in Bluetooth are called: Hold, Park, and Sniff modes. The differences between them are the level of activity and how the connection to the piconet is handled, i.e. if the device is disconnected (has given up its AM_ADDR address) vs. still connected but sleeping. Which mode to use clearly involves a trade-off between responsiveness and the amount of power saved. In the hold mode the slave and master each decides upon a period of time when no transmission between them will occur. Since the slave retains its AM-ADDR (i.e. its active member address) through out the process it can restart the data transfer immediately when making the transition out of Hold mode, this scheme can be used for managing low power devices. A Bluetooth unit in the Park mode remains synchronised with the master, but gives up its AM_ADDR. Instead it is given a so called PM_ADDR (Parked Member Address), i.e. it does not participate in the communication in the piconet. This enables more than 7 slaves to be synchronised to the same master at the same time. In sniff mode the slave listens to the piconet at a reduced rate, thereby reducing its duty cycle. This is done by negotiating specific slots (sniff slots) where communication between the master and slave can occur. The duration of the sniff slots can be dynamically extended if either of the two devices is likely to have more data to send. The transmission ends when either of the nodes decides to stop and the other unit will discover this after a timeout period. Figure 11 depicts a scenario when a master communicates with one slave in sniff mode. At the first and fifth sniff interval the slave does not get any response from the master and goes back into a power saving mode. The second interval is used until the master (in this case) decides to end it, which the slave finds out after a timeout period. When having a reduced duty cycle in one piconet the slave can be active in another, thus Sniff mode can be utilised when building scatternets.

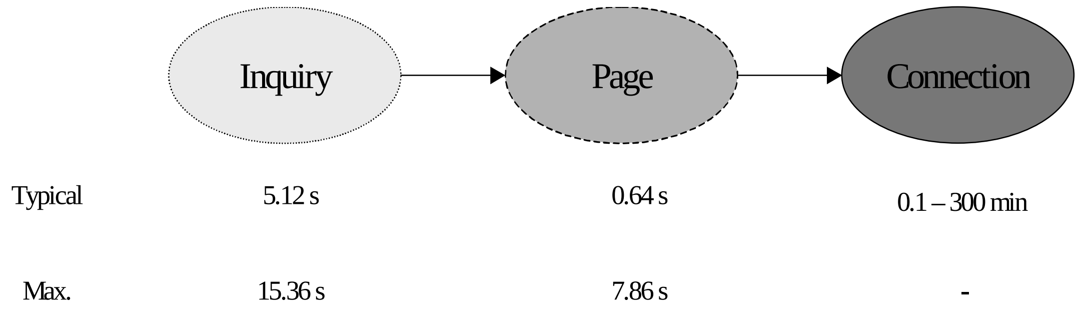


Figure 10. The connection establishment process and the time associated with each step

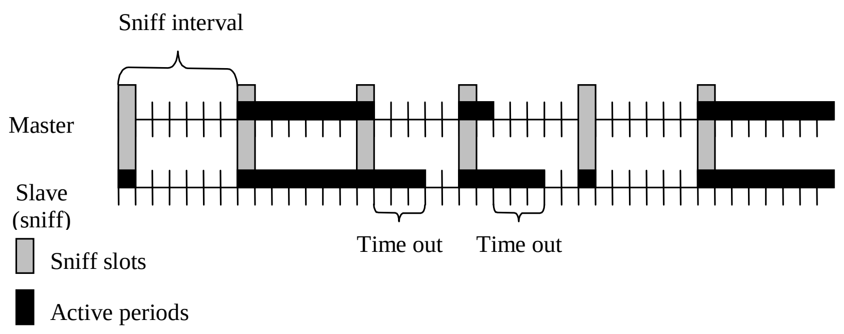


Figure 11. Connection between master and slave using sniff mode.

3.4 Link Management – LMP and L2CAP

The link manager takes care of scanning for other devices, connecting to other devices, and maintaining connections, i.e. it provides basic functions for the ACL/SCO link setup and release procedures. LMP messages are exchanged to tell what a slave is allowed to do and this is done by requests and replies between the slaves and the master. Examples of this include determining available baseband packet types and establishing the T_{poll} interval (negotiated using LMP messages between master and the slave). The T_{poll} parameter is the longest interval between two consecutive polls from the master to a certain slave, this can be used as a low level QoS parameter ensuring both a given minimum bandwidth and maximum delay for the link. However, there are several deficiencies related to this parameter as a QoS guarantee, some of those are stated in section 4.2. The HCI provides a standardised interface between LMP and L2CAP. The HCI provides a flow control mechanism since the packet transmitted from the L2CAP layer is stored in a HCI buffer which has a limited size, thus forcing the L2CAP layer to stop its transmission when this buffer is full.

L2CAP provides connection-oriented and connectionless data services to upper layer protocols with protocol multiplexing capabilities, packet segmentation and reassembly, and the conveying of QoS information. An L2CAP packet is segmented into one or more Baseband packets, the first contains the L2CAP length. Each of the Baseband has indicators of whether it is the start or a continuation of a L2CAP packet. A L2CAP transmission must be completed before a new L2CAP transmission is started over the same ACL link, which implies that interleaving of L2CAP segments over the HCI interface is not allowed.

4 Scheduling

When many devices share the same resource, rules have to be constructed describing who will use the resource, when, how much, and for how long. One approach is to let the system be contention basesd such as the medium access method CSMA/CD. Another approach is to set up scheduling rules for the resource. In context of Bluetooth and this thesis the resource that needs scheduling is the channel. The piconet master is in charge of this scheduling. According to**Error! Reference source not found.** a scheduling algorithm should be:

- Easy to implement in terms of software or hardware. This is especially important for Bluetooth where the algorithm should be of low complexity since the computational resources in a Bluetooth chip are limited.
- Fair; so that all users get a fair amount of the resource. The fairness measurement can be calculated depending on many aspects such as the amount of traffic, priority of the traffic, etc. Implementing a fair algorithm will automatically protect against devices taking an unfair share of the resource. Though the fairness doesn't imply that all traffic streams should receive the same amount of capacity. It simply means that traffic within a certain group with the same priority level and the same QoS demands should each get an equal share, i.e. get a fair share. When only best effort (BE) traffic is considered all participating units should receive an opportunity to utilise the same amount of the resource. A model for fair packet scheduling is the fluid fair queuing model, which is explained in section 4.1.

- Give users (if requested) a guaranteed performance bound so they know what to expect. These bounds can be expressed in terms of bandwidth, delay, delay jitter, and loss.
- The algorithm should have an easy admission control mechanism to see whether a new request for the resource should be permitted or not and to make this decision quickly. When providing QoS it is important to limit the amount of promised service since the resource is limited.
- The algorithm should utilise the resource as efficiently as possible and at the same time provide the required service(s).

When developing a new scheduling algorithm the goals can vary depending on whether best effort traffic or QoS traffic is considered. However, in both cases, to give as much service to the users as possible the limited resource has to be utilised as efficient as possible. This will in the case of Bluetooth lead to minimising the number of POLL/Null sequences where no data is transmitted while at the same time ensuring high throughput and low delay. For BE traffic the goal is to share the resource equally among the active slaves and effectively utilise the available bandwidth. For QoS traffic the goal is to give the slaves the amount of the resource that was negotiated in the setup of the connection.

The algorithm presented in this thesis needs to work well for both cases, but not necessarily when the two types of traffic are present at the same time, since priority will be given to the QoS traffic. Service differentiation such as giving QoS parameters does not improve the system's performance, but simply gives better performance to one flow at the expense of another.

4.1 Fairness - The fluid fair queuing model

The fair fluid model is often used as a model for wire-line networks [36]. The model could be visualised as a set of fluid flows flowing through the same pipe. This pipe has a capacity C and each flow f is assigned a weight r_f over an infinitesimally small window of time Δt. Each backlogged flow (backlogged flow means a flow with a busy queue) is given a channel capacity of $C \cdot \Delta t \cdot (r_f / \sum_{i \in B(t)} r_i)$, where B(t) is the set of all backlogged flows at time t. There are algorithms for both wire line and wireless networks using this model [36]. The fairness in the fluid fair queuing model is upheld as long Equation 1 holds, looking over an interval (t_1,t_2) during which the two flows are continuously backlogged. The equation states that if for any backlogged flow the capacity given to it $(W_i(t_1,t_2))$ in the time interval (t_1,t_2) divided by its assigned weight (share of the medium) is equal to the same quota for any other backlogged flow (i.e. belongs to B) during the same time period. If this is the case, the system is considered to be fair.

$$\forall i, j \in B(t_1,t_2) \left| \frac{W_i(t_1,t_2)}{r_i} - \frac{W_j(t_1,t_2)}{r_j} \right| = 0$$

Equation 1. The Fluid Fair Queuing fairness equation.

However, this model is not sufficient for wireless systems, they are more complex since the "pipe" (the medium we are transmitting over) is unreliable compared to transmitting over a wire. This unreliability for wireless pipes can (on occasions) make the flows unable to transmit even though they are backlogged, which is not taken into consideration in the fluid fair queuing model. However, in Bluetooth the fast frequency hopping sequence will, as stated before, spread the errors due to interference from different sources equally among the slaves when looking over long periods of times. Furthermore the flow's number of channel access times (number of polls) should be considered rather than their throughput. This is motivated by the fact that in Bluetooth slaves can transmit using different packet sizes. A slave (A) using 1 slot packets will introduce more overhead than a slave (B) using 5 slot packets. Using only throughput as the fairness measurement would entitle slave A to be polled 12.56 times[6] for every poll of slave B (unless different weights are assigned to the slaves, i.e. a dynamic weighting system for all slaves) leading to an increased overhead. Therefore this thesis uses a modified version of the fluid fair queuing equation (Equation 1) which is stated in Equation 2, with the difference from Equation 1 being that the number of polls (P) for backlogged flows are considered, and the behaviour the equation is considered fair if the maximum value, d, for the difference of all backlogged flows is less or equal to a positive number • .

[6] DH5 packet size / DH1 packet size = 339/27.

$$\forall i, j \in B(t_1, t_2), \max\left|\frac{P_i(t_1,t_2)}{r_i} - \frac{P_j(t_1,t_2)}{r_j}\right| = d, d \le \zeta$$

Equation 2. The fairness equation

So the goal for the algorithm from a fairness perspective is to satisfy Equation 2. By using this equation the fairness among the slaves can be measured. Furthermore in this thesis fair requires that QoS traffic receives service (in terms of delay and bandwidth) as negotiated and for BE traffic Equation 2 is the only fairness criteria.

4.2 Bluetooth scheduling

The main task in Bluetooth scheduling is to utilise the bandwidth as effectively as possible, have a fair distribution of the available bandwidth, and at the same time have small packet delay, high throughput, and low packet dropping probability (for QoS traffic, which may have strict delay bounds, for instance in a real time application). One means of doing this is to minimise the amount of void POLLS, i.e. minimise the amount of wasted bandwidth. A void POLL or a Poll/Null sequence is when neither the master nor the slave has something to send to each other, but is still given access to the channel. To be able to completely avoid the void Polls the master must know about the status of both the master's and the slave's transmission queues. As Bluetooth is designed today, the master only has knowledge about the status of its *own* queue. One way to obtain the slave's queue status information would be to change the interpretation of existing bits in the Bluetooth specification (there are for instance two bits in the packet header that not are specified). Another would be to introduce new packets (or use the first bits in the payload) that can be exchanged between the master and slaves. Both these approaches are out of the scope of this thesis since one requirement of this thesis (see section 6) was that the solution must follow the specification. Instead of changing the specification there are other signals (at least in case of bursty traffic, such as BE traffic) that can serve as indicators for whether the slave has more traffic to send or not. One such sign is a slave who responds to a poll with a packet containing data (i.e. not a null packet) which could be used as an indication that the slave has more data to send (this applies for busty traffic, like BE traffic), this idea is utilised in the Fair Exhaustive Poller (FEP) algorithm which is explained in more detail in section 5. However, the traffic may not be bursty thus leading to false assumptions using this idea, hence another traffic differentiation should then be used to overcome this problem.

Bluetooth scheduling algorithms are based upon a set of parameters in the L2CAP and LMP level, which are negotiated between the master and slave. The slaves can only request parameter values and the master makes the ultimate decision whether to approve or deny a request. The parameters at the L2CAP and LMP level are listed in table 2.

Name	Level	Description
Token Rate	L2CAP	The rate at which an application may send data continuously.
Token Bucket Size	L2CAP	The maximum buffer space available, i.e. the maximum burst size an application is allowed to offer instantaneously.
Peak Bandwidth	L2CAP	The maximum bandwidth at which an application is allowed to offer traffic.
Latency	L2CAP	The maximum delay between transmission of a bit form higher layers to its initial transmission over the air.
Delay variation	L2CAP	Difference between the maximum and minimum possible delay that a packet will experience.
Flush Time Out	L2CAP	Specifies the amount of time the originator 's link manager will attempt to retransmit an L2CAP segment before flushing the packet (i.e. delete it).
T_{poll}	LMP	Specifies the maximum time between subsequent transmissions from the master to a particular slave.

Table 2. The QoS parameters in Bluetooth

Since the scheduler is placed below the L2CAP level, the L2CAP parameters have to be translated in some way. One way is to translate the L2CAP parameters into the LMP parameter $T_{poll.}$. However, this approach causes some problems when using T_{poll} as the only parameter giving guaranteed QoS traffic:

1. T_{poll} only guarantees that the master will attempt to communicate once per T_{poll} slots. The T_{poll} timer is restarted after each POLL even in case of retransmissions. Due to this fact the slave receives less effective bandwidth than negotiated and the T_{poll} interval is shifted.
2. The T_{poll} interval specifies the time between the master's polling of the slave and not the time between the slave's transmission opportunities. When polling the slave the master might use multi-slot packets causing the slave's transfer interval to be different from the T_{poll} interval.

The Figure 12 and Figure 13 shows the T_{poll} deficiencies stated in points 1 and 2 above.

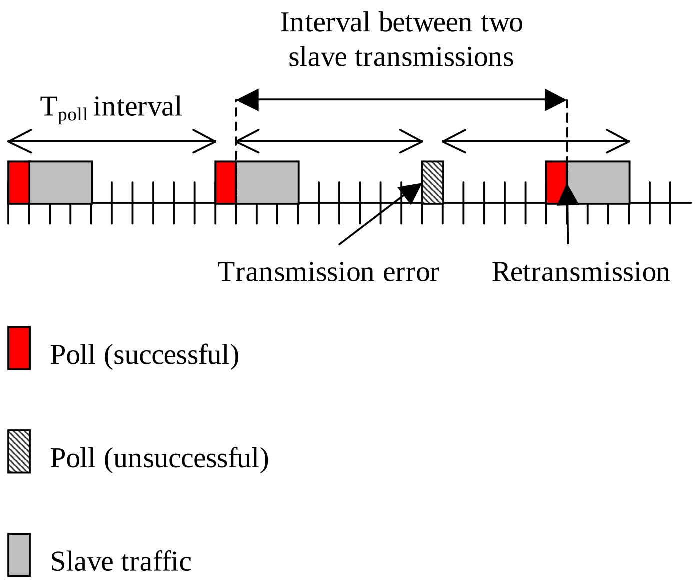


Figure 12. One of the T_{poll} deficiencies (deficiency number 1)

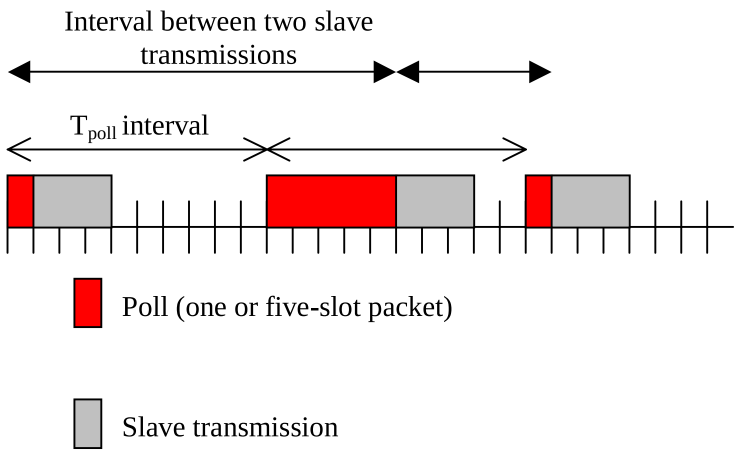

Interval between two slave transmissions
T_poll interval
Poll (one or five-slot packet)
Slave transmission

Figure 13. One of the T_{poll} deficiencies (deficiency number 2)

4.2.1 Pre-scheduling

Pre-scheduling plans a number of tasks in advance, for the scheduling algorithm this means calculating the polling order a number of TDD slots (frames) in the future. This includes scheduling of all communication within the network such as ACL traffic, SCO traffic, and Page scan, but also includes times when the system is idle (possibly due to power saving). The specific number of frames scheduled in advance is called a batch, and the polling order defined in the batch is called the batch schedule. When the batch has been prepared it is fixed, i.e. the polling order in the batch cannot be changed. Since the batch is prepared in advance the calculation performed to prepare a batch and the actual execution of the polling defined in the batch are separated. Once a batch has been prepared the execution of the batch is very simple and straightforward since no new calculations needs to be done.

In the previous section both the major advantages and disadvantages of the pre –scheduling principle are mentioned. The fact that the batch is prepared in advance and cannot be changed after a certain point in time results in a system with reduced responsiveness. If for instance a slave responds to a POLL with a NULL packet (i.e. according the assumption stated in section 4.2 this indicates an empty slave buffer) in the beginning of the batch, the system cannot react to this indication until the next batch. Hence if the slave is scheduled to be polled later in the same batch this may lead to a void poll. Another disadvantage with pre-scheduling is the possibility of a mismatch between the expected packet length (estimated when the batch was prepared) and the actual packet size used by the slave. An example of this is given in Figure 14 where some of the slave transmissions are different (using smaller baseband packets) compared with the pre-scheduled prediction. In this case the slot utilisation is 63 %, and the unused 37 % of the slots could have been utilised if pre-scheduling was not used. The advantages of batched scheduling lies in the possibility for the hardware to only wake up at certain points in time instead of distributing the workload over time, and thus lowering the power consumption.

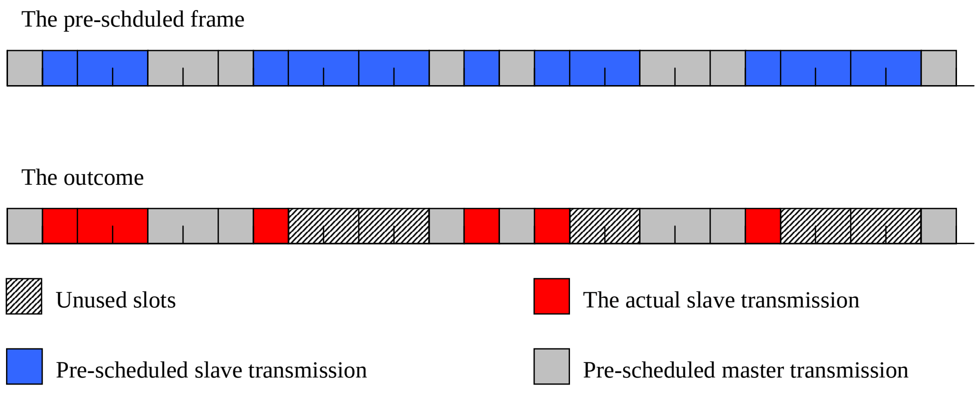

The pre-schduled frame
The outcome
Unused slots
The actual slave transmission
Pre-scheduled slave transmission
Pre-scheduled master transmission

Figure 14. One disadvantage of using pre-scheduling is that pre-scheduled slots might be wasted.

5 Proposed and existing scheduling algorithms

The Bluetooth specification does not specify a scheduling algorithm, however it proposes the Roud Robin (RR) (see section 5.1.1) algorithm as a possible scheduling algorithm. Thus there is no standardised scheduling algorithm used by all manufacturers. The performance of different Bluetooth networks can therefore vary depending on the manufacturer of the master node. Even though not specified by the standard there are several scheduling algorithm proposed for Bluetooth pico- and scatternets. A good scheduling algorithms can estimate the slave's queue status, i.e. if they have something to send or not, which is very important since accurate queue estimations can avoid the bandwidth wasting void polls. There are two different types (for MAC schedulers) of approaches to learn this queue status information: history-based and feedback based. The difference is that the feedback-based approach relies on explicit information carried by the upstream packets (from a slave to the master), while in the history approach the master makes a prediction based upon the previously received packets. The feedback approach often implies a change in the Bluetooth specification or a change in interpretation of transferred data compared to the Bluetooth specification. The feedback approach also introduces some feedback overhead, but the advantage is that the feedback is explicit and therefore always more accurate than a history-based approach where predictions and estimates are made from prior transmissions.

5.1 Intra-piconet schedulers

When comparing different Bluetooth scheduling algorithms there are mainly three criteria: efficiency, fairness, and level of complexity. Apart from these it is also beneficial if the algorithms has QoS capabilities. This section describes some of the proposed scheduling algorithms available today.

5.1.1 The Round Robin algorithm and its extensions

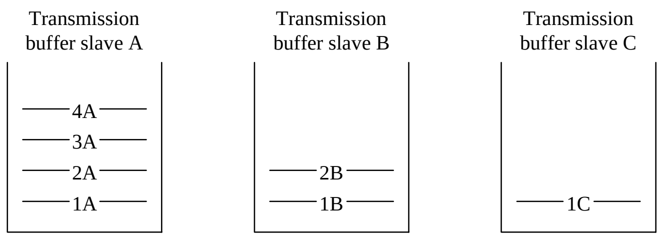
The simplest existing scheduling algorithm is a simple Round Robin (RR) poller, which is widely used as a benchmark when evaluating new scheduling algorithms. The RR algorithm polls the slaves in a cyclic fashion, thus all slaves get an equal chance to share the resource. Hence a solution based purely on RR will not consider the amount of traffic sent by each slave, as a result slaves that do not have anything to send will be polled anyway. Thus the activeness of the slaves (a slave is said to be active if it has data to send, and passive otherwise) is not taken into consideration, which results in void polls.

A possible extension of RR is to introduce an exhaustive service. The exhaustive RR algorithm (ERR) operates as a regular RR algorithm with the difference that the Master-Slave pair currently communicating is allowed to transmit all the packets waiting in their respective buffers before the next slave will be polled. If a master slave pair always had data to send to each other, this could easily lead to starvation of other master slave pairs.

On possible approach to prevent starvation in the ERR scheme is to introduce a gated version with the difference from the exhaustive version being that the number of packets that the slaves are allowed to send before it is the next slave's turn is limited by a fixed number, the LRR limit. This solution, called Limited Round Robin (LRR) scheduling solves the starvation problem. It also produces guaranteed sojourn time between two consecutive transmission possibilities.

The last alternative is called Weighted Round Robin (WRR), which is an RR extension where each slave is assigned a weight. This weight is the relative number of polls each slave is given over a period of time. For example, if we have a 2 to 1 relationship between two slave's respective weights, then one slave will be polled twice as much as the other.

An example of the scheduling outcome using the four mentioned RR polling algorithms is depicted in Figure 15. Slave A has four packets in its transmission queue, B has two, and C has one. It is assumed that no other packet arrives, the LRR limit is two, and that the weights using WRR is set to 4:2:1 (for slaves A:B:C). We can see that for this scenario the RR and ERR each have 3 void polls, while the LRR(2) has 2 void polls. However, if the polling order were reversed, (i.e. starting with C, then B, and then A) the result would be 5 void polls or the RR, and 3 void polls for both ERR and LRR(2). The number of void polls in the WRR case will in always be 0 as long as the weights are ideally assigned.


Transmission buffer slave A
4A
3A
2A
1A
Transmission buffer slave B
2B
1B
Transmission buffer slave C
1C

Polling algorithm	Polling order									
RR	1A	1B	1C	2A	2B	Void Poll	3A	Void Poll	Void Poll	4A
ERR	1A	2A	3A	4A	Void Poll	1B	2B	Void Poll	1C	Void Poll
LRR(2)	1A	2A	1B	2B	1C	Void Poll	4A	Void Poll		
WRR*	1A	1B	2A	1C	3A	2B	4A			

* The polling order may vary depending of the execution algorithm.

Figure 15. Example of RR (and its extension) polling outcome

5.1.2 The Fair Exhaustive Polling algorithm

The history based Fair Exhaustive Polling (FEP) algorithm is based on the assumption that BE traffic is bursty, i.e. if a slave responds with a data packet when polled it is assumed to have more data to send. The scheduler divides all BE nodes on the link into an active and a passive group. The slaves in the active group are polled with a RR scheme and the slaves in the passive group are polled when their T_{poll} timer expires. A slave in the passive group is moved to the active group if, when polled, it has data to send, or the master has data to send to it. If all slaves belongs to the passive group, then they all are moved to the active group. Furthermore it remains in the active group until it responds with a null packet and the master does not have data to send to this slave. The FEP algorithm can be visualised as a two state machine, this is depicted in Figure 16. This algorithm is very promising since it is very easy and simple to implement and has shown good results in simulations [10]. As for its complexity, all the data the master needs to collect and update are which group each slave belongs (one bit), which slave to poll next in the round robin list, and the slaves T_{poll} timers. The Batched – Fair Exhaustive Polling (B-FEP) algorithm is a pre-scheduling version of the FEP algorithm that schedules a number of frames in advanced.

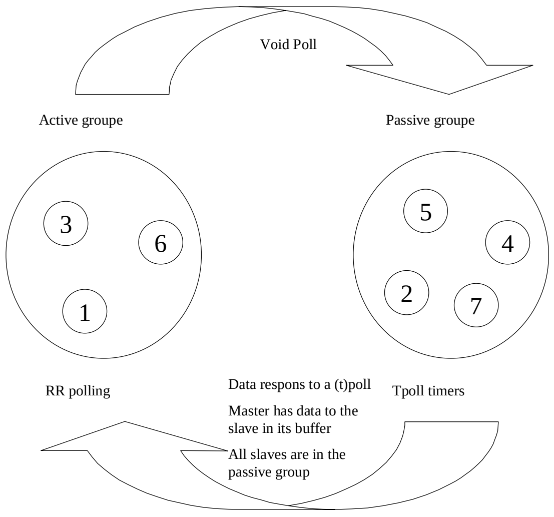

Void Poll
Active groupe
Passive groupe
3
6
1
5
4
2
7
RR polling
Data respons to a (t)poll
Master has data to the slave in its buffer
All slaves are in the passive group
Tpoll timers

Figure 16. The FEP states

5.1.3 The Predictive Fair Poller

The Predictive Fair Poller (PFP) [10] is a scheduling algorithm for Bluetooth BE traffic which was enhanced to also support QoS traffic. The node to be polled is determined based on a predicted probability that a slave will have something the send when polled (based on recent polling events and prediction of the higher layer traffic, i.e. a history based algorithm) and QoS demands.

This algorithm tries to emulate a Generalised Processor Sharing (GPS)**Error! Reference source not found.** system, which in terms of Bluetooth means that on the smallest possible time scale available polls should be divided equally among the slaves that have data to send. To emulate this the following computations are made before each POLL is executed.

For each slave a moving average is calculated on the inter-arrival time between the start of IP packets sent from the slave in response to a POLL. This result, together with the last POLL result, is used to make a prediction whether the slave has data to send or not. The prediction results in a probability that a slave has data to send and is further used to calculate the instantaneous fair share of the resource for each slave. The instantaneous fair share together with the share given to each slave over a period of time is used to calculate the fraction of the fair share for each slave. This fraction together with the probability that each slave has data to send is used to make the decision of which slave to POLL next. Comparing this with the FEP (see section 5.1.2) algorithm, which is a round robin algorithm with one added variable per slave, we conclude that PFP has a higher level of complexity. As the paper [10] shows that the two algorithms has about the same performance regarding throughput, efficiency, and response time, there is no advantage of using FEP for scheduling only BE traffic.

5.1.4 The Fair and efficient Polling algorithm with QoS Support

The Fair and efficient Polling algorithm with QoS Support (FPQ) is an algorithm that supports QoS demands in terms of both bandwidth and delay [26]. This algorithm proposes adding a flow rate parameter to the Bluetooth specification that includes an average Inter arrival time between two consecutive packets and the average packet length. From these flow rate parameters the algorithm makes its next polling choice based on predictions of the queue states both up and down link and the time since slaves were last polled. The authors claims that the next slave to be polled can be calculated using 2 exponential operations, 13 additions, 2 division, 6 multiplications, and 1 comparison, in addition 10 values need to be kept per slave. In terms of complexity the algorithm is good, not much computation is needed, but this is only possible with changes in the Bluetooth specification. Hence this algorithm is ruled out.

5.1.5 The Queue Status based Polling Interval and its successors

In [2], three different scheduling algorithms are proposed with the goal to increase throughput and at the same time reduce power consumption. The idea is to switch the slaves into sniff mode whenever the power overhead of switching the device into sniff mode and back into active mode is less than the power saving achieved by keeping the slave in sniff mode. The MEAN and LIBT (Last Inter-Burst Time) are two algorithms that utilises sniff mode and tries to predict packet bursts by calculating the mean of recent inter-arrival times. This mean value is then used to set proper sniff intervals, which the slave can utilise and thus save some power. A slave returns into active mode, if the master or slave queue consists of more than 2 packets. The difference between the two algorithms lies only in the amount of prior data used in the predictions.

Since Bluetooth is made for devices with limited battery resources it is a good idea to utilise the sniff mode for bursty BE traffic. However, to synchronise all the sniff events with different cycle times without getting too many collisions in the scheduling is very hard. Also, using this scheme throughput may decrease since the device will be polled less frequently (than it would otherwise be polled) and predictions of bursty traffic are very hard to make.

The last of the three proposed algorithms is the Queue Status based Polling Interval (QSPI), which utilises sniff mode and the state of the master’s outgoing buffer associated with each slave. The slaves are put into one of three modes. These three modes are active mode and two sniff modes, with different polling intervals depending on if they has packets to send or not when polled.

The idea of using different levels (similar to an exponential service level where you increase/decrease the service rate in steps) is not advisable. Either the slave has packets to send or it doesn’t, it is as simple as that, you can not have less then “no packets to send”. For the bursty BE traffic it is better to give a slave full capacity when it has traffic to send instead of slowly increasing the rate (see Figure 17).

As for complexity the third algorithm (QSPI) is the most complex of these three (but it is not that complex). The master only needs to keep track of the predicted inter arrival time (i.e. it is history based) and what state each slave is in.

5.1.6 The Limited and Weighted Round Robin algorithm

The limited and weighted round robin (LWRR) polling scheme [2] is not very different from the FEP algorithm, in both the authors try to optimise the performance by reducing the amount of void polls. This optimisation is done by assigning weights to the slaves, which can increase or decrease depending on whether the slave has traffic to send or not. An increased value results in a more frequently polled slave. The major disadvantage with this scheme compared to the FEP algorithm in terms of best effort traffic is that it uses an exponential service reduction/increment. In case of bursty BE traffic in Bluetooth networks it is most often better to poll the slave at full speed when it has packets to send and not making an exponential increase. Doing this, a slave with data to send will be allowed to use the medium at full speed from the beginning, thus enabling it to transfer the data burst over a sorter period of time compared to using an exponential service rate. The same is true for an exponential decrement in the service rate. Figure 17 explains the advantage of using a FEP like polling method instead of an exponential polling (in this case a two step polling) method when bursty traffic is present. One burst of data arrives at the slave some time after the master polls the slave which replies with data. In the FEP approach the slave is given the maximum possible fair share of the resource until it responds to a Poll with a Null packet, after which the master stop polling the slave. The two-step exponential approach starts polling the slave with twice as large polling interval as in the FEP approach. When the slave also (as the first poll) responds the second poll with a data packet its share of the resource is increased to the maximum possible. This maximum share is kept as long as the slave is responding to polls with packets containing data. When the first null packet is received by the master it reduces the service rate by half. The second Null reply results in the master stopping polling of this slave. By looking at the two graphs we can see that the FEP approach has transferred the data burst faster and with less overhead. However, it should be stated, that if an other packet burst would have arrived before the exponential approach stopped polling the slave it could have had a higher responsiveness to that new burst than the FEP approach.

The complexity of this algorithm is low. The master only needs to keep track of the weight assigned of each slave. The computation of the weights is straightforward, a null response will reduce the weight and a data response will increase the weight.

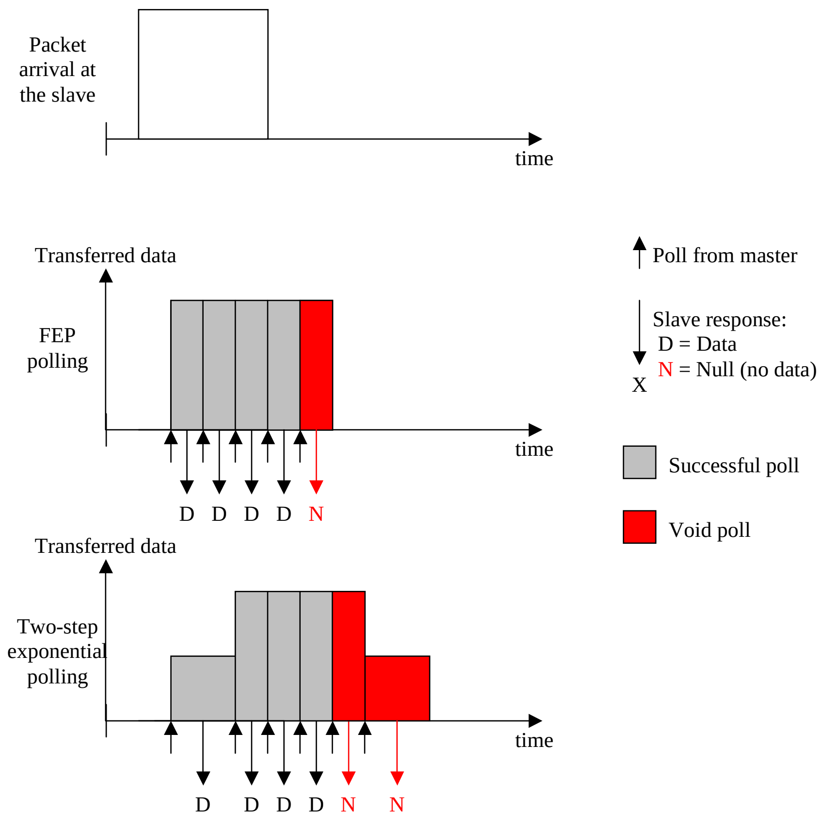


Figure 17. Compassion of different polling approaches when bursty traffic is present

5.1.7 The Adaptive Flow-based Polling algorithm

The ideas regarding the Adaptive Flow-based Polling algorithms introduced in [4] are similar to the ideas in [2] since all use some kind of multi-level increase /decrease in the poll interval. The most promising algorithms from [4] are the Adaptive Flow-based Polling (AFP) algorithm and a variant of that called Sticky AFP, which is a gated version of AFP. This algorithm derives the exponential increase/decrease by a new interpretation of a bit in the Bluetooth specification, i.e. it is a feedback based scheduling algorithm. As mentioned earlier the exponential increase/decrease in the number of polls when BE traffic is present are not the best way to go and as the algorithm relies on changes in the specification, hence it is ruled out of further consideration in this masters thesis.

5.1.8 Data Scheduling and SAR for Bluetooth Medium Access Control

The scheduling algorithms proposed in [21] set priorities by keeping track of and exchanging information about the state of both the master's and the slave's queue. The authors claims that there are free bits in the Bluetooth payload header that can be used for this information exchange from slave to master. Based on this information exchange the master builds up states (1,1), (1,0), etc. for each master slave pair where a (1,0) means that the master has data to send (1) and the corresponding slave has an empty buffer (0). From these states the master decides which slave to poll next while trying to optimise link utilisation, i.e. a (1,1) pair is polled before a (0,1) pair since data is transmitted in both directions. This utilisation based thinking is extended by the authors to include packet sizes when known, e.g. a master slave pair in state (1,0) where the master is using a 5 slot packet (i.e. a 5/6 = 83% link utilisation) has priority over a (1,0) pair where the master is using one slot packets (i.e. a ½ = 50% link utilisation). However, since the algorithm is based on specification changes it is ruled out of further consideration in this thesis.

5.1.9 The floating Threshold Algorithm

The floating threshold algorithm [28] is a feedback-based solution, where the flow bit in the payload header is used to carry the feedback information from the slaves to the master. The flow bit is set when the slave has data exceeding a certain threshold in their outgoing buffer. The master maintains a floating threshold per slave representing the minimum number of backlogged packets at the slave. This information together with the queue status at the master is used to select which slave to poll next. The algorithm gives priority to the master slave pair which has a high utilisation of the link. This utilisation priority is the same as the one in section 5.1.8. A slave from the group with both master and slave queues containing data (i.e. 100% utilisation) will be polled before a pair where only the master or slave has data to send (i.e. 50% utilisation). To avoid starvation of a pair with a low utilisation a timeout is introduced that forces the master to poll slaves that not have been polled for a period of time.

This algorithm imposes changes in the interpretation of the flow bit compared to the specification which implies that this solution is out of the scope of this thesis. As for the complexity the master only has to maintain the floating threshold per master-slave pair. Updating the threshold is easy, it is just a matter of adding or subtracting a value to/from the threshold depending on if the flow bit is set or not.

5.1.10 The Class based Packet Scheduling Policies for Bluetooth

The authors of [22] propose two class based scheduling policies for Bluetooth, which according to them gives end-to-end QoS guarantees. Both differentiate traffic into the two possible traffic classes (classes 1 and 2) based on the different delay requirements of applications. The first proposal, called priority queuing (PQ), has a traffic identifier that sorts the traffic into different input queues at the master (according to their priorities), the queue with the highest priority (common to all slaves) gets exhaustive service before it is time for the other queues (one per each slave) with lower priority traffic. The idea behind this scheduling policy is depicted in Figure 18. The problems with this approach are that it does not give any guarantees apart for giving one traffic class precedence over the other. Assume that we have an application present which has a maximum allowed delay of 10 ms, and at the same time other priority traffic present which is scheduled in the high priority queue, there is no guarantee that the 10ms delay bound will be kept. Furthermore the slave to master (up-link) traffic is not considered at all.

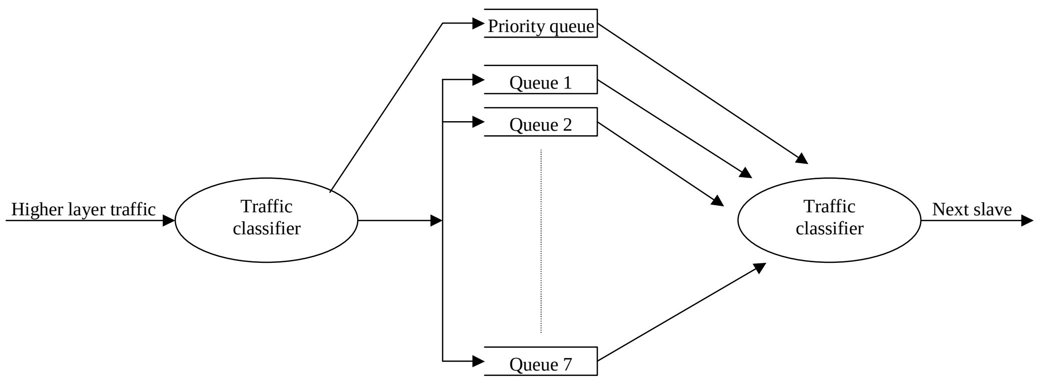


Figure 18. The principal idea behind the priority queuing scheduling algorithm

Their second approach is based on the first and optimised by taking the slave to master traffic into account as well as the master to slave traffic (down-link). The idea is basically the same as with up-link traffic, i.e. divide the traffic into in two priority classes and then poll slaves generating higher priority traffic with an exhaustive service. The slave generating low priority traffic is polled in an RR fashion.. A lower priority slave will receive traffic/be polled if there are no backlogged high priority slaves and no high priority traffic in the master input queue. The problems in this model are: starvation of lower priority traffic (can be helped with admission control), and the separation of the two traffic classes. The separation in the up-link case is done by introducing a bit in the layer 3 header thus requiring extra signalling overhead and thus lying outside the scope of this thesis.

5.1.11 Throughput-Delay Priority Policy

The authors of [37] suggest the Throughput-Delay Priority Policy (T-DPP) scheduling algorithm for a Bluetooth piconet, which the authors claims to be “efficient and QoS aware”. The algorithm is based on Equation 3, which is used to calculate priorities for different Master-slave pairs.

$$P = \alpha T + (1-\alpha) D$$

Equation 3. The basis for the T-DPP algorithm.

This algorithm takes into account both a priority class (T) assigned to each master slave pair and the yielded service slots (D), i.e. the amount of slots given up to higher priority traffic. The priority classes are based on the buffer content of both the master and slave side, if both sides have data to send (the utilisation is 100%) resulting in the highest priority class, the next highest priority class is if either the master or slave has a 5 slots packet to send while the other sides queue is empty (resulting in a utilisation of 83 %), and so on. The scheduler checks all slaves’ priority, P, in a RR fashion. If the pair with the highest priority is checked, it is polled. If another slave is checked it yields its chance to the highest priority pair, and increases its D value by one. The highest priority pair will at the same time decreases its D value by 1. The α parameter is a weight parameter for system performance between throughput (T) and Delay (D). This algorithm only uses one simple equation, however the authors do not describe how the slaves buffer content information is retrieved. They claim to provide QoS by mapping the QoS parameters into T and α without mentioning how this mapping should be done. If we have a very regular source sending packets at a constant rate with strict requirements on the delay (low allowed latency) there is no guarantee that it will be served regularly. The problem is that you adjust the parameters for one pair without regard to the other pairs. It is obvious that the parameters have to be changed depending on whether we have 2 or 7 pairs in the network when trying to maintain a delay bound for a regular source.

5.2 Piconet-schedulers conclusion

The work on QoS scheduling in Bluetooth is quite a new area of research. However, a great deal of research has been for with BE scheduling within the Bluetooth community. A study of existing scheduling algorithms shows that only two history based algorithms meet the requirements (stated in this thesis) as a BE scheduling algorithm. The two algorithms are the Fair Exhaustive Poller (FEP) (see section 5.1.2) and the Predictive Fair Poller (PFP) (see section 5.1.3). Their performances are almost equal [10]. However, the FEP algorithm has a lower complexity and is therefore regarded as the best one (in this thesis).

6 Restrictions/Limitations.

When designing a QoS aware scheduling algorithm several limiting factors must be considered so that the design will be consistent with Bluetooth as it is currently realised based on the existing specification. Some of the major limitations are briefly described here.

As already stated, Bluetooth is a master driven technology, this rules out some of the possible scheduling schemes. Furthermore, radio is an unreliable access medium and the Bluetooth channel is limited in bandwidth. Since the bandwidth is limited the reduction of void polls has to be prioritised.

Portable devices have limited power resources, therefore the Bluetooth standard defines modes for reducing power consumption of the devices. A power effective scheduling algorithm should utilise these modes, by being aware of absent nodes in low power modes. The low cost requirement results in limited computational resources, thus making simplicity a requirement of any scheduling algorithm. The scheduling solution presented in this thesis is more valuable for Ericsson if it is can be integrated in the rest of Ericsson´s Bluetooth solution. The Bluetooth specification can be seen as a limiting factor since an algorithm based on changes in the specification would be useless for Ericsson unless the rest of the Bluetooth SIG accepts these new changes. Today many applications such as streaming audio and video traffic have QoS requirements, hence the algorithm should be able to support these requirements.

6.1 The traffic classes

When evaluating the proposed scheduling algorithm three different traffic classes were identified. The traffic classes and combinations of these were used in the simulations. The traffic classes are Streaming (Audio and Video), Human Interface Devices (HID, e.g. a mouse or a joystick), and guaranteed bandwidth traffic, which could be a FTP file transfer. Furthermore BE traffic was also considered in most of the simulations.

6.1.1 Streaming

Streaming applications over Bluetooth could for example be music encoded with MP3 (Audio) or a streamed (Video) news clip to your laptop. The streaming characteristics assumed for both audio and video are shown in Table 3.

	Audio	Video
Traffic volumes	Asymmetric	Asymmetric
Bandwidth	Constant	Variable
Bit rate (kbps)	64 – 384	32 – 384
Frame size (bytes)	200 –800 (constant)	100 – 1500 (variable)
Inter-arrival (ms.)	20 – 32, regular non bursty	30 – 100, regular non bursty
Latency requirements (ms.)	50 – 500	500
Modelled as a:	Constant Bit rate (CBR) source	Variable Bit rate (VBR) source

Table 3. Streaming traffic characteristics

The streaming video has relatively weak latency demands, which means that when a streamed L2CAP video packet has arrived we have some time before it must be transferred. As shown in Figure 19 the placements of polling instances can be made in various ways, while meeting the latency requirements (staying within the delay bound). Here the streaming packet arrives at the slave and is then transferred to the master (i.e. we are looking at the up-link case). Since the packet is at the slave, the master does not know its size, the only thing the master knows the parameters stated in Table 2 that have been negotiated by the master and slave. The approach with the arrows in black (poll the slave as soon as the L2CAP packet has arrived) has the advantage that if the packet is lost it is possible to attempt retransmissions before the delay bound is reached. This approach results in a higher probability of delivering the L2CAP packet within the promised delay bound. To poll the slave exactly as often as needed (if polled more often the slave buffer would be empty which would lead to bandwidth waste) to transfer a L2CAP packet requires knowledge about the L2CAP packet size.

To approximate the packet size the master needs to have a packet size predictor, which will add more complexity to the scheduling algorithm. The second approach marked with red (solid) pointers is based on calculations from the negotiated bandwidth for the slave. This calculation results in evenly distributed polling instances. This approach has the disadvantage of shorter periods for retransmission of the last packet.

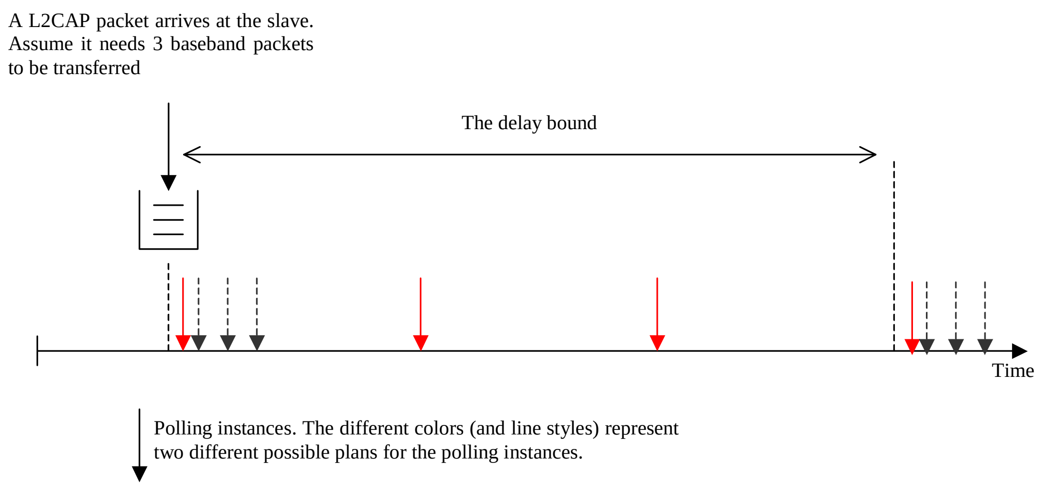

A L2CAP packet arrives at the slave. Assume it needs 3 baseband packets to be transferred
The delay bound
Time
Polling instances. The different colors (and line styles) represent two different possible plans for the polling instances.

Figure 19. Different possible placements of the polling instances for traffic with delay requirements.

6.1.2 HID

Traffic generated by a device similar to a mouse or a Joystick that must be polled often. The characteristics are stated in Table 4.

	HID
Traffic volumes	Symmetric
Bandwidth	Constant
Bit rate (kbps)	Depending on latency requirements
Frame size (bytes)	< 23, i.e. fits in one baseband packet of maximum 3 slots size
Inter-arrival (ms.)	Depending on latency requirements
Latency requirements (ms.)	10 –100

Table 4. HID traffic characteristics

An algorithm using a pre-scheduling method has problems completing retransmission(s) in time. If we have a pre-scheduling frame longer than 8 TDD frames (i.e. 10 ms) the master cannot in all cases make a retransmission within the delay bound for traffic with a 10 ms latency period (or less). A possible solution to this problem is to over provisioning for traffic types with a very short latency period (i.e. a latency period shorter than the length of the pre-scheduling frame).

6.1.3 Guaranteed bandwidth traffic

The guaranteed bandwidth traffic class consists of traffic with reserved bandwidth but no delay requirements. This could for instance be a FTP session. The characteristics are stated in Table 5.

	Audio
Traffic volumes	Asymmetric
Bandwidth	Constant
Bit rate (kbps)	Up to 721
Frame size (bytes)	Irregular
Inter-arrival (ms.)	Bursty
Latency requirements (ms.)	None

Table 5. Guranteed bandwidth traffic traffic characteristics

7 The Algorithm

The proposed scheduling algorithm is a combination of different mechanisms working together as depicted in Figure 20, where the parts within the dotted box belongs to the scheduler. Both QoS and BE traffic are handled by this scheduler, which differentiates traffic into two classes. The first class consists of high priority traffic (for instance traffic with strong requirements on delay and traffic with regular inter-arrival times such as HID traffic). Even if most of the QoS traffic is grouped among the priority traffic not all QoS traffic is, some of it is included in the second group, the ring. The ring consists of QoS traffic with only bandwidth requirements (i.e. without specific delay requirements or with very high delay tolerance) and BE traffic. Slaves in the ring are assigned weights and polled according to a WRR scheme. The scheduler wakes up once per pre-scheduling frame cycle to compute the next pre-scheduling frame. When computing the next pre-scheduling frame the Priority traffic is first considered since its polling placement it important for meeting the relevant delay requirements, then the remaining slots of the pre-scheduling frame are filled with traffic from the ring. Slaves are then polled according to the order stated in the pre-scheduling frame and feedback (such as slave response, NACK/ACK, and amount of data transferred) is collected and used when planning subsequent pre-scheduling frames. The feedback is the basis for retransmissions, setting the activeness of the slaves (using FEP), and new calculations of weights in the ring according to a lead-lag model. Retransmission is only given to priority traffic, and only when retransmissions can be completed within the negotiated delay bounds. The scheduler has adopted FEP ideas in order to avoid void polls and thereby increase the efficiency of the system. The lead-lag model calculates the weights for the QoS traffic in the ring and thereby ensures that each slave receives their negotiated bandwidth. In the remainder of this section several ideas underlying the parts of the algorithm are described and explained.

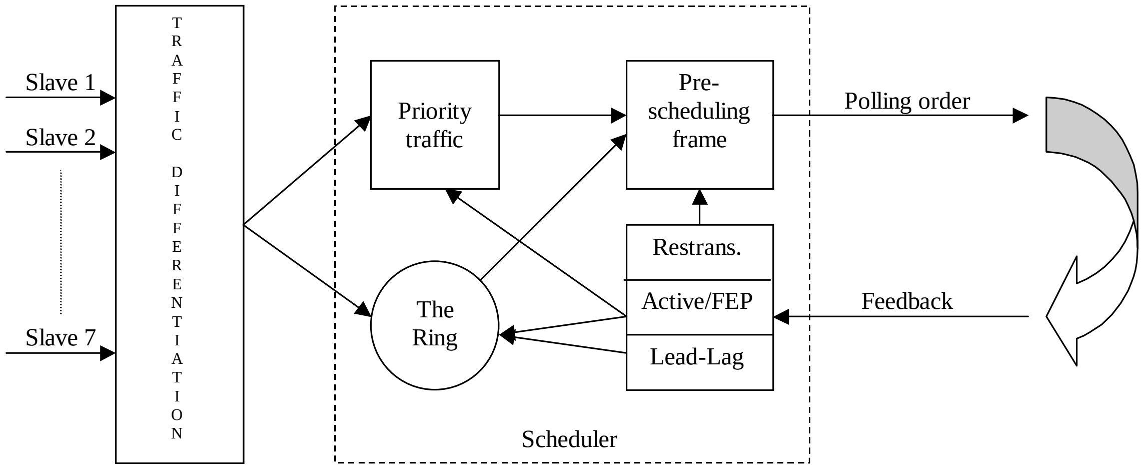


Figure 20. The scheduling basics

7.1.1 Traffic differentiation

Traffic is divided into two groups: Priority traffic and Ring traffic. Priority traffic is QoS traffic with high demands on the latency period (i.e. traffic only allowing short delays between the transmissions) and traffic with very predictable behaviour, i.e. regular inter-arrival times and packet sizes. All other traffic is inserted into the ring, see section 7.4. Thus CBR traffic is regarded as priority traffic and VBR traffic is inserted in the ring unless it requires low delay.

7.1.2 Admission control

It is hard to support QoS traffic demands if there is no admission control mechanism controlling the load of the piconet, i.e. ensuring that it is possible to meet the negotiated QoS demands. This algorithm and the simulations performed assume that an admission control mechanism is implemented and only allow admitting slaves with demands which the scheduler can meet while keep its earlier commitments.

Furthermore the admission control can be used to calculate the initial number of elements each slave associated with the ring should have.

7.2 Pre scheduling

The algorithm supports pre-scheduling as described in section 4.2.1. The length of the pre-scheduling frame (in number of slots) is set to a number *Pre-scheduling_length*. However, the length can be dynamically extended if for instance the master schedules a transmission to start at the end of the pre-scheduling frame and the transmission (in both directions) exceeds the pre-scheduling frame length.

It should also be possible to shorten the length of the pre-scheduling frame, consider the case when only one slave is active, then the need for the scheduling algorithm is reduced. The best way might be to schedule a couple of frames in advance and than schedule some sleep for the relevant devices. Another case where a shortened pre-scheduling frame is needed is if the scheduler is interrupted by a higher priority task while in the middle of scheduling a frame.

In the simulations a pre-scheduling frame is prepared when all the instructions of previous frames has been executed. In reality this is not possible since it takes some time to prepare the next pre-scheduling frame before transferring it to the hardware. Therefore when applying pre-scheduling the pre-scheduling frame must be prepared in advance. It is beneficial to make this preparation as close (in time) to the actual polling execution as possible, since that means that we can use more recent feedback when scheduling the frame. To make this preparation easier (and thereby enabling it to be completed later in time) part of the pre-scheduling frame can be prepared in advance. A frame longer than the pre-scheduling window can be used to facilitate the scheduling of periodic tasks (such as SCO transmissions, Sniff transmissions, and the regular scheduled priority traffic). Part of this longer frame can then be copied to the pre-scheduling frame before adding all the ACL traffic. Figure 21 depicts the usage of a longer pre-scheduling window.

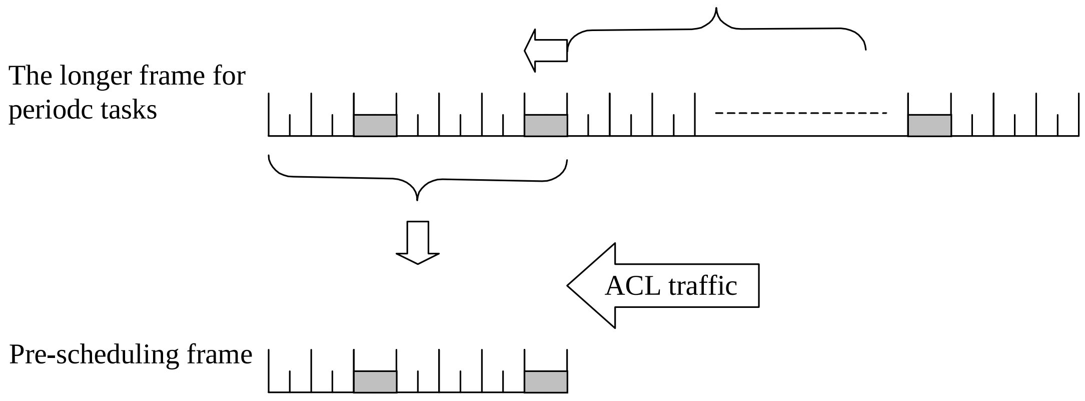


Figure 21. The longer pre-scheduling window

7.3 The priority traffic

The priority traffic require low delay and/or has regular inter-arrival times. For this traffic we have to calculate a polling interval with which we will meet the requirements. This traffic has to be transferred within a certain delay bound, therefore the priority traffic must be given a high priority within the scheduler as compared to other traffic (e.g. PAGE and Inquiry) in order to keep these delay bounds. For instance this traffic should therefore be able (just as SCO traffic is) to interrupt a page scan.

7.3.1 The poll interval

The T_{poll} parameter insures a given minimum bandwidth and bounded maximum interval between poll attempts from the master. This parameter, is kept as it is and is used to detect when passive slaves becomes active again. The first T_{poll} deficiency as stated in section 4.2 is that the T_{poll} interval is shifted if there are retransmissions. Applications that require Quality of Service such as Audio and Video, typically require guranteed periodic transmission opportunities, therefore the poll intervals of priority traffic are specified by a separate parameter (per slave) kept in the scheduler.

Traffic within group1 are sources with high demands for low or predictable latency (i.e. applications allowing only low delay), these traffic sources often generate traffic with fixed inter-arrival times, e.g. HID traffic. To meet their latency demands we calculate the inter-arrival for each flow. There are different possible approaches to do this, I will describe one approach that utilise the token rate and latency (assumed to be equal to the inter-arrival time of L2CAP packets from higher layers) specified in the flow specification, and the assumption that the master knows (must know/guess at the start of every pre-scheduling frame) which baseband packet type the slave will use.

The approach estimates the average L2CAP packet size by Equation 4. The average packet size divided by the size of the baseband packet type used for the transmission yields the number of baseband packets need for transmitting (on average) a single L2CAP packet, this division will be rounded up to nearest integer according to Equation 5.

$$PacketAverageSize[bits] = Tokenrate[bits/s] \bullet Latency[s]$$

Equation 4. Estimation of the average packet size

$$\#BasebandPacketsPerL2\,CAPpacket = \left\lceil \frac{PacketAverageSize[bits]}{BasebandPacketSize[bits]} \right\rceil$$

Equation 5. Number of baseband packets per L2CAP packet

Now that we know the number of estimated baseband packets we need to transmit an average L2CAP packet, these baseband packets have to be transmitted within the agreed maximum latency period (which end at the time for the arrival of the next L2CAP packet). The distribution of the polling instances has been discussed section 6.1, however the distribution must be chosen so that there is time for retransmissions (within the latency bound) after the last baseband packet has been sent. I have chosen to distribute the polling instances evenly in time, to avoid the starvation that might occur if a high priority flow is given the opportunity to transmit all its baseband packets in a burst.

$$PollInterval = \frac{Latency[s]}{\#\,BasebandPacketsPerL2\,CAPpacket} = \frac{Latency[s]}{\left\lceil \frac{Tokenrate[bits/s] \bullet Latency[s]}{BasebandPacketSize[bits]} \right\rceil}$$

Equation 6. The evenly distributed poll interval

Not all polling interval calculations will end up as a multiple of 0.625ms (Bluetooth's time slot length), since the polls will be performed in a slotted time domain it is sometimes necessary to round the calculation to a multiple of 0.625ms. This rounding will be to the closest multiple (of a slot, still keeping rule of the master polling at even numbered slots) with a value lower than the poll interval time.

The poll interval is also adjusted according to the master to slave traffic. For instance if we have calculated a poll interval for a slave to master transmission and the master to slave traffic usually consists of only 1 slot packets and the master changes to a 5 slot packet the polling interval has to be reduced by 4 * 0.625 ms, to avoid the second deficiency described in section 4.2.

7.4 The Ring[7]

When all priority traffic, events such as SCO traffic, PAGE SCAN, and retransmission are pre-scheduled the remaining unused slots are used for the traffic belonging to the Ring, i.e. BE traffic and QoS traffic with little or no delay requirements (i.e. long latency).

[7] The ring concept was first eveloped by Niklas Johansson and Johan Rune at Ericcson.

As stated before the ring is a realisation of the WRR concept. In every round of a WRR scheduler, each node is scheduled for service in proportion to its weight. For example, if node A is assigned weight 1 and node B is assigned weight 2, then node B will be polled twice as often as node A. Hence by assigning weights to different slaves the relative polling frequency of different slaves can be controlled. Thus, weights can be used as a QoS mechanism to differentiate the QoS for different slaves or to give PMP nodes compensation when returning to the piconet. In addition, the weights can be used to ensure that slaves using different baseband packet lengths efficiently receive the same proportion of the bandwidth (i.e. by assigning a greater weight to a slave using shorter packets than a slave using long packets, resulting in a higher polling frequency of the former slave to compensate it for the shorter packets). However, in section 4.1 fairness was defined to be measured in number of polls and not in bandwidth therefore all BE slaves are assigned the same weight, equal to 1. The FEP concept described in section 5.1.2 is utilised in the ring. The ring consists only of elements that are regarded as active based on the feedback from previous transmissions. More details when a slave is regarded as passive are stated in section 7.5.3.

In the implementation used in this thesis the WRR scheme is implemented as a ring using a circular list. However, other implementations are possible, one of them is described in section 7.4.3. While polling active slaves belonging to the ring the scheduler polls one slave after the other going through the ring sequentially round after round. Consequently, if the ring were to contain only one element for each slave in the active group, the resulting polling would be plain RR polling of the slaves in the the ring. In order to reflect the weight of the slaves, each slave is represented by a number of elements that is proportional to its effective weight, W_i. Where W_i is an integer that can take values between [1, *MaxNumberOfElements*], hence each slave can simply be represented in the ring by a number of elements equal to the slave's W_i parameter. To keep track of the current position in the ring the scheduler keeps track of a *ringElementPointer*. This pointer is increased one step in the ring after each poll of a slave corresponding to an element in the ring. The number of elements each QoS and PMP slave has and the length of the ring are kept by the scheduler to make compensation calculations for the QoS and PMP slaves. Figure 22 illustrates the ring, each number in the ring represents the corresponding slave, this ring has a total of 16 elements, slave number 1 has 4 elements (i.e. weight $W_1 = 4$) and slave number 2 has 2 elements (i.e. weight $W_2 = 2$), etc. The next element is one, which means that the next slave to be polled is slave number one, after it has been polled the *ringElementPointer* will be updated, and point on the next element, in this case an element containing a six.

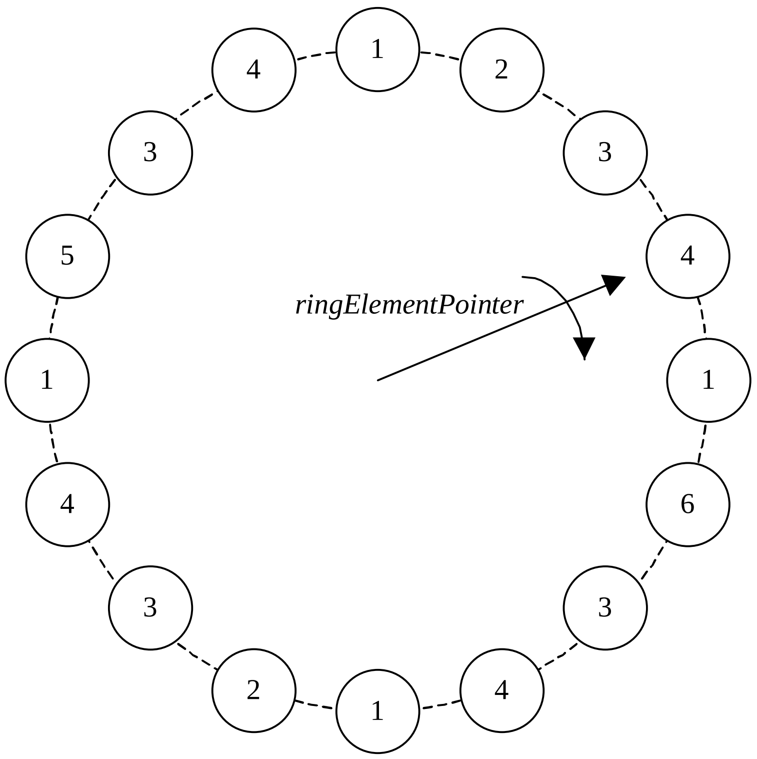


Figure 22. The ring used for polling low priority traffic. Each element in the ring has a number, which represents the slave corresponding to this number. This ring has elements from 6 different slaves.

Using this scheme, when we have completed a round in the ring each slave in the active group is polled a number of times that is proportional to its weight, W_i. Thus WRR is achieved. Assuming that the elements of the ring do not change, the scheme will always achieve fairness (among the slaves in the ring) during an interval equal to the time it takes for the scheduler to complete one circuit of the ring. If the ring is unchanged depends on how often you would like to update the ring (which is a trade-off between complexity and responsiveness of the system). Furthermore, if a slave still scheduled in the ring leaves the active group, its elements in the ring are consequently removed, the remaining BE slaves in the active group will share the polling resources given up by the slave that left the active group. The share that each of the remaining slaves in the ring will receive will be proportional to its weight, W_i. Since all BE slaves have the same weight they will receive an equal share of these new polls. Thus this scheme will achieve fairness defined as in section 4.1. This fairness will be upheld irrespective of how the elements of the different slaves are distributed in the ring. However, it is preferable that the elements of a slave are evenly distributed along the ring, as this makes the polling smooth, resulting in low (or at least slowly changing) delay variation.

7.4.1 Inclusion of elements in the ring

The algorithm for including new elements in the ring is vital for the system's performance. Inserting many new elements may cause large delays for BE slaves and slaves with a small number of elements in the ring. Another problem that might cause large delays is if the inclusion of new elements in the ring results in a poor distribution of the slave's elements. An example of poor distribution of the slave's elements is depicted in Figure 23, which has the same amount of elements as Figure 22, but with another distribution. A distribution as shown in Figure 23 will result in a bursty capacity allocation for the slaves in the ring, which not is good since the probability of full buffers increases.

After all elements that should be removed from the ring have been removed, the BE slaves that become active are included in the ring. They are inserted just before the element to be polled next, i.e. where the *ringElementPointer* points. The reason for putting it here is that all slaves that have an element in the ring have already waited some time to be polled, the new element to insert should not bypass this waiting "queue", so additions are placed at the end of the queue.

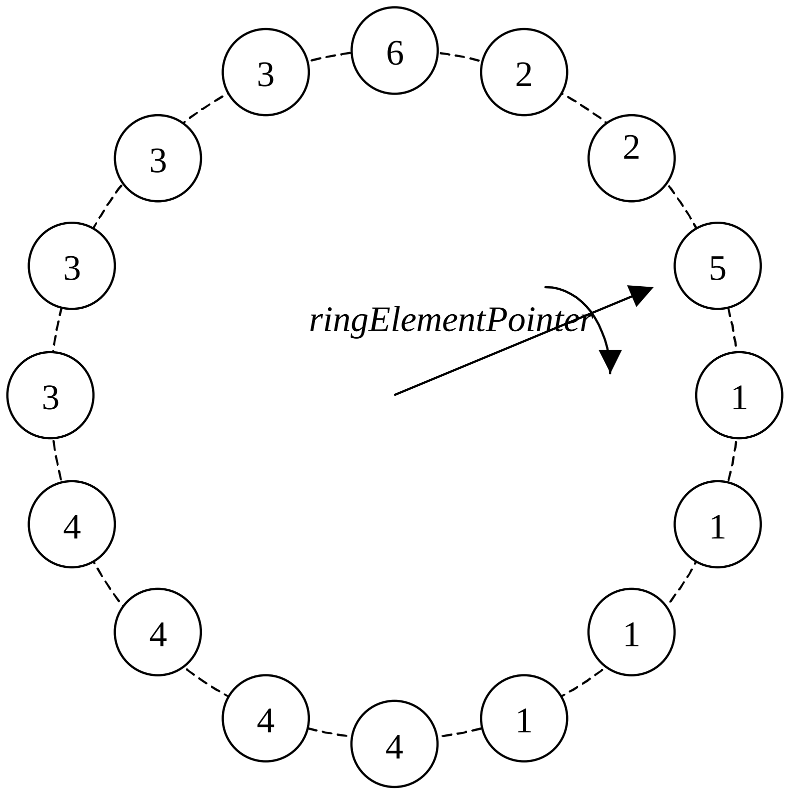


Figure 23. Example of a poor distribution of elements in the ring.

For the QoS slaves, their inclusion can be achieved in many different ways, but only in ways that also satisfy the low-complexity and computation-efficiency requirements. A good and simple distribution algorithm is to go through the ring backwards, sequentially (starting at the position indicated by the *ringElementPointer*) and insert the W_i new elements with L // W_I +1 old elements in-between every two consecutive new elements (L is the length of the ring before insertion). Inserting slaves backwards (from the *ringElementPointer*s polling direction) is done based on the same reasoning as insertion of BE slaves, the other elements in the ring have already waited some time. The reason for adding the one to the L // W_I can easily be seen in the following example: assume that 20 new elements are to be inserted in a ring of length 19, 19 // 20 = 0 which would imply that the 20 elements would be inserted in a sequence starting from the *ringElementPointer*. Adding the one overcomes this problem. If, after going through a complete circuit of the ring, there are still new elements to insert, the insertion algorithm just continues the insertion (considering the newly inserted elements as old elements) until finished. This approach can easily be extended by checking at each insertion the elements in-between which we will insert the new element: If one of the two elements has the same value as our element we will try to insert it one step forward in the ring. If it is not possible to insert the element with other elements in-between, we will discover this after one circuit. This extension leads to more wider distribution of the elements in the ring and was used in the simulations.

7.4.2 Removal of elements from the ring

Before any insertions are made in the ring the slaves that no longer belong to the active group are removed. If some (but not all) of the elements belonging to a QoS slave are removed, this is done in a similar way as the inclusion of elements. That is, if W_r elements are to be removed for a slave with W_{old} elements in the ring, every W_{old}// W_r+1 elements is removed. An additional check as in the case of inserting elements can be implemented to check if the elements before and after of the element to be removed are the same. If they are then the next element according to W_{old}// W_r+1, is tried until you have completed one circuit of the ring, at which time the original element is removed.

7.4.3 A token bucket model, an alternative visualisation of the ring

The ring can be visualised and implemented in many various ways, for example as a linked list, a vector, or we could use a token bucket model. On such token bucket model that has a central token bucket and a bucket per slave is depicted in Figure 24. The buckets contain tokens, where a token represents a poll instance. Once a slave has been polled it is charged a token from its bucket, which is then returned to the system's central bucket. When the central system bucket is full (i.e. contains all the tokens in the system) all tokens are redistributed to the slaves. The scheduler checks the buckets in a RR order and if the bucket contains a token, that slave is polled and charged a token, i.e. the full central token bucket represents one circuit of the ring.

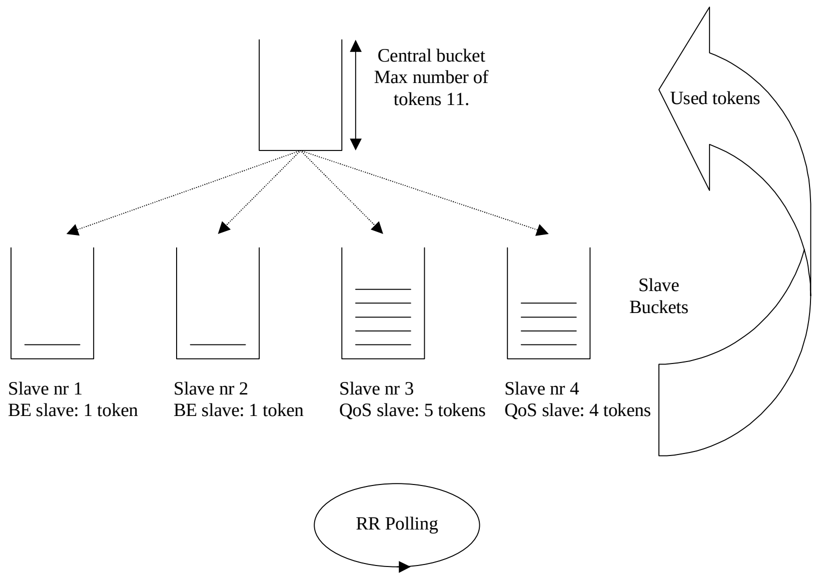


Figure 24. The token bucket model

The number of tokens for each slave is fixed as long as no slave leaves or enters the system. However, they can become passive and active according to FEP without the need for a new token calculation. The best effort slaves will receive one token each at a redistribution from the central token bucket, the QoS slaves receives a number of token ensuring them of their negotiated bandwidth.

When a slave becomes passive its share has to be redistributed to the other slaves. To emulate the fairness as defined in this thesis the tokens of a passive slave are shared equally among the BE slaves. Thereby the QoS slaves will receive their share and all BE slaves will share the remaining bandwidth.

7.5 Feedback

Every time the scheduler wakes up (i.e. when it is time to prepare a new pre-scheduling frame) it processes the results from the last pre-scheduling frame. Thus the responsiveness of the system will depend on the length of the re-scheduling frame. Feedback indicates whether retransmissions are needed, if the QoS slaves in the ring need to change their respective weight, and if slaves have become passive. Figure 25 shows the different feedback information gathered from the last pre-scheduling window.

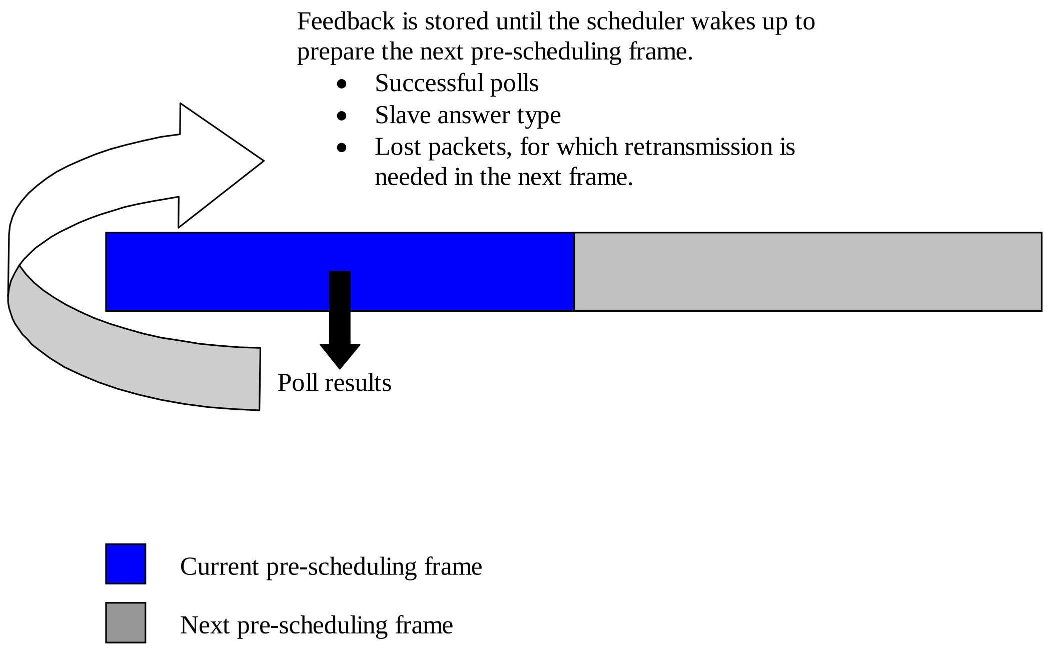

Feedback is stored until the scheduler wakes up to prepare the next pre-scheduling frame.
• Successful polls
• Slave answer type
• Lost packets, for which retransmission is needed in the next frame.
Poll results
Current pre-scheduling frame
Next pre-scheduling frame

Figure 25. Feedback from previous pre-scheduling frame

7.5.1 Retransmissions

A priority traffic flow receives a retransmission opportunity in the pre-scheduling frame following the one where the flow lost a packet or packets. This retransmission is only done if it is possible to do so within the slave’s negotiated delay bound and is placed in the beginning of the pre-scheduling frame.

7.5.2 Lead-Lag compensation

The lead-lad concept discussed in [27], is a key component in many wireless fair queuing algorithms. Leading means that a flow has received more capacity than requested over a period of time relative other flows. Lagging flows are flows that over a period of time have received less capacity than negotiated. The compensation model determines how leading flows give up their lead and how lagging flows make up for their lag.

In Bluetooth flows might be bursty, and this can be exploited. A flow will be allocated more capacity when it has a lot to send and less (or none) in when there is no data to transmit or when other flows are lagging and need to transmit more. In Figure 26 we can see two flows with different negotiated bandwidth (the dotted lines), their lead and lag (the zigzag lines). When the bandwidth given after a period of time is below the dotted line they are lagging and when the opposite occures then they are leading. However, in the long run flows should follow the dotted lines. When leading the alternatives are to continue to lead (keep the same slope), decrease the slope until lagging, or stop transmitting (horizontal slope) the opposite holds for a lagging flow.

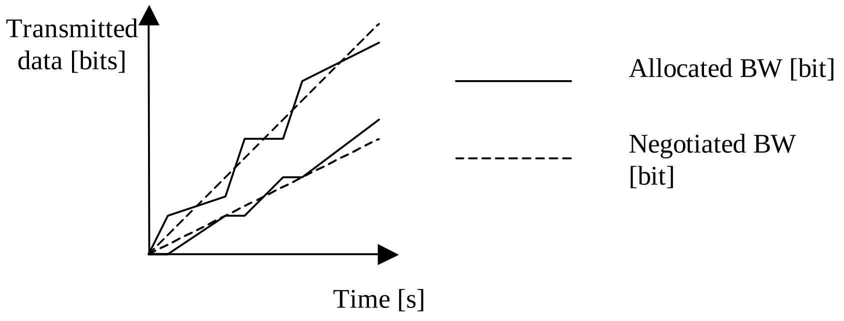


Figure 26. The Lead-Lag concept

All traffic associated with the QoS slaves is analysed and the number of successful transmissions over a period of time is kept. Two different approaches to how this information should be interpreted are suggested depending on exactly what you are guaranteeing when giving the slaves a certain bandwidth guarantee. Either you guarantee the slaves a certain raw bandwidth, i.e. assuming that if a slave has been given 10 polls over a time period Δt and is expected to use DH5 packets, then regardless of whether the slave answered with a null or data packet it has received an **opportunity** to use a bandwidth of 339 * 10/ Δt Kbytes/s (since 339 bytes is the maximum payload of a DH5 packet). An other approach takes the null answers and lost packets into account and calculates the amount of **successfully sent data** packets over a time period Δt (i.e. a null packets does not contain any data and is therefore not considered as a successfully sent data packet). The third approach is a further refinement of the second approach. It takes into account how much data the packet sent from a slave contained. Which approach to use depends on what the scheduler guarantees, however, the approach and calculations are the same, the only differences is what is stored in the memory.

The number of elements to insert for each slave is calculated for every new pre-scheduling frame. The simplest solution is to just calculate the compensation considering only with the number of elements that each slave currently has. This calculation is done using a memory, which keeps track of the bandwidth allocated (or given depending on which scheme that is used) (g_bw_i) to each slave (according to one of the three approaches above), and compares this value with the negotiated value (n_bw_i). Equation 7 gives the change that needs to be made.

$$\Delta bw_i = \frac{(n\_bw_i - g\_bw_i)}{g\_bw_i} + 1$$

Equation 7. Computing the necessary change in bandwidth

This change in number of elements for each flow has to be computed relative to the current number of elements in the ring. The corresponding number of elements that each slave is given can be calculated according to Equation 8.

$$L = \text{ring.size(), the number of elements in the ring}$$
$$Lx = L$$
$$for(All\ \ active\ \ guranteed\ \ bandwidth\ \ slaves\ \ i)$$
$$if\left(\frac{E_i \cdot \Delta bw_i}{Lx} > 1\right),\ \ \Delta E_i = max\_number\_of\_elements$$
$$else\ \ \Delta E_i = \frac{E_i \cdot (\Delta bw_i - 1)}{1 - \frac{E_i \cdot \Delta bw_i}{Lx}}, \text{ given by manipulating Equation 9}$$
$$\Delta E_i = round(\Delta E_i)$$
$$update\ \ Lx,\ \ Lx = Lx + \Delta E_i$$

Equation 8. The change in number of elements

$$\frac{E_i}{Lx} \cdot \Delta bw_i = \frac{E_i + \Delta E_i}{Lx + \Delta E_i}$$

Equation 9.

Equation 9 simply states that the current fraction in the ring E_i/L times the needed change in bandwidth (• bw_i) should be equal to the fraction of the ring after the change (E_i + • E_i) / (L + • E_i) for slave i. Where (E_i + • E_i) is the number of elements for slave i after the change and consequently (L + • E_i) is the total number of elements in the ring after the change.

Using this approach the last slave to be updated will get the most accurate update. There are different possibilities for the continuation. The first is to stop here and when it is time for the next update start with another slave, i.e. going round in a RR fashion. This approach will lead to fairness over time. An other approach (which still can start updating slaves in a RR fashion) is to check the slaves to see if the update was ok with repect to the new Lx value, i.e. a two-step iteration (which could be extended with more iterations) towards the ideal value. Equation 10 gives the iteration check; if the computed new weight differs from the weight the slave is entitled to by more than 0.5, i.e. by more than one element, a change is made.

$$Z = \left(\frac{E_i + \Delta E_i}{Lx} - \frac{E_i \cdot \Delta bw_i}{L}\right) \cdot Lx$$
$$if\left(|Z| > 0.5\right), \text{ the calculated change is accepted}$$
$$else\ \ \Delta E_i = \Delta E_i + round(Z), Lx = Lx + round(Z)$$

Equation 10. The iteration check

This iteration can be repeated a number of times if needed. However, since we are dealing with discrete values many iteration will probably not be needed.

However, using only this scheme in an environment with bad radio conditions causing so many errors that the system cannot meet the negotiated QoS guarantees will not lead to a even degradation in service level. This is because the calculations do not take the other flows leading or lagging into consideration. The flows with a higher negotiated bandwidth will be penalised and degrade more than flows with a lower negotiated level. To overcome this problem some additional rules are necessary. These rules only apply if there is more than one active QoS slave present in the ring.

1. If all active QoS flows are lagging:
   Remove all BE slaves in the ring

Find the QoS flow with the largest lag,

$$\text{i.e.}\forall \text{Active QoS flows find} : \max\left(\frac{Negotiated\ \ Bandwidth - Given\ \ Bandwidth}{Given\ \ Bandwidth}\right) =$$

$$= \max(\Delta bw_i - 1) = (\Delta bw_m - 1)_{\max}$$

Give the flow, m, with the maximum lag *MaxNumberOfElements* elements in the ring.
Give the rest of the active flows a number of elements according to:

$$MaxNumberOfElements \cdot \frac{(\Delta bw_i - 1)}{(\Delta bw_m - 1)_{\max}} \cdot \frac{Negotiated\ \ Bandwidth_i}{Negotiated\ \ Bandwidth_m}$$

2. If an active flow is leading and we are about to reduce its amount of elements by x; then check if there are any active flows lagging, if there are then reduce the number of elements to one.

Furthermore if a QoS flow is leading and the calculations indicates that we should remove all elements its this will only be done if there are lagging flows.

In the simulations each slave starts with one element in the ring which it will also be given when becoming active after a passive period; however, a better approach is to assign each slave (associated with the ring) a *Starting_weight*, which is only recalculated if a slave enters or leaves the network (not simply becoming passive or active) and thereby alters the load of the network. The *Total_starting_weight* value is kept as a variable according to Equation 11. All these parameters have to be computed by the admission control algorithm.

$$Total\_starting\_weight = \sum_{\forall \text{ present slaves i}} (Starting\_weight)_i$$

Equation 11.

When a passive QoS flow becomes active again after a passive period it receives a weight according to Equation 12, which weigh the slave's starting weight based on the current number of elements in the ring in relation to the starting number of elements in the ring. A further extension of Equation 12 would weight the calculated value based on the lead or lag the current slave is experiencing, and thereby instantly compensate for this lead or lag.

$$Elements\_to\_insert = (Starting\_weight)_i \cdot \frac{Current\_ring\_length}{Total\_starting\_weight}$$

Equation 12.

7.5.3 Queue status estimations -FEP

The active and passive group concept from the FEP algorithm is utilised. A slave in the passive group is polled when its T_{poll} timer expires, the master has data to send to it (when it is moved to the active group), or when the active group is empty. When the active group is empty all the slaves in the passive group are polled according to a simple RR scheme

7.6 Time windows

In some of the simulations time windows are used. These time windows are associated with the priority traffic. A time window means that already scheduled traffic can be moved within this time window in order to make the insertion of the traffic from the ring into the pre-scheduling window easier. The advantage of using time windows can be seen in Figure 27. Two scenarios are depicted: one using time windows and one without. The grey slots are high priority traffic scheduled before scheduling the traffic in the ring. The ring pointer is updated clockwise and we can see that the utilisation of this pre-scheduling frame is higher when using time windows. However, keeping track of the time windows and moving traffic around in the pre-scheduling frame increases the complexity of the algorithm as will be described below.

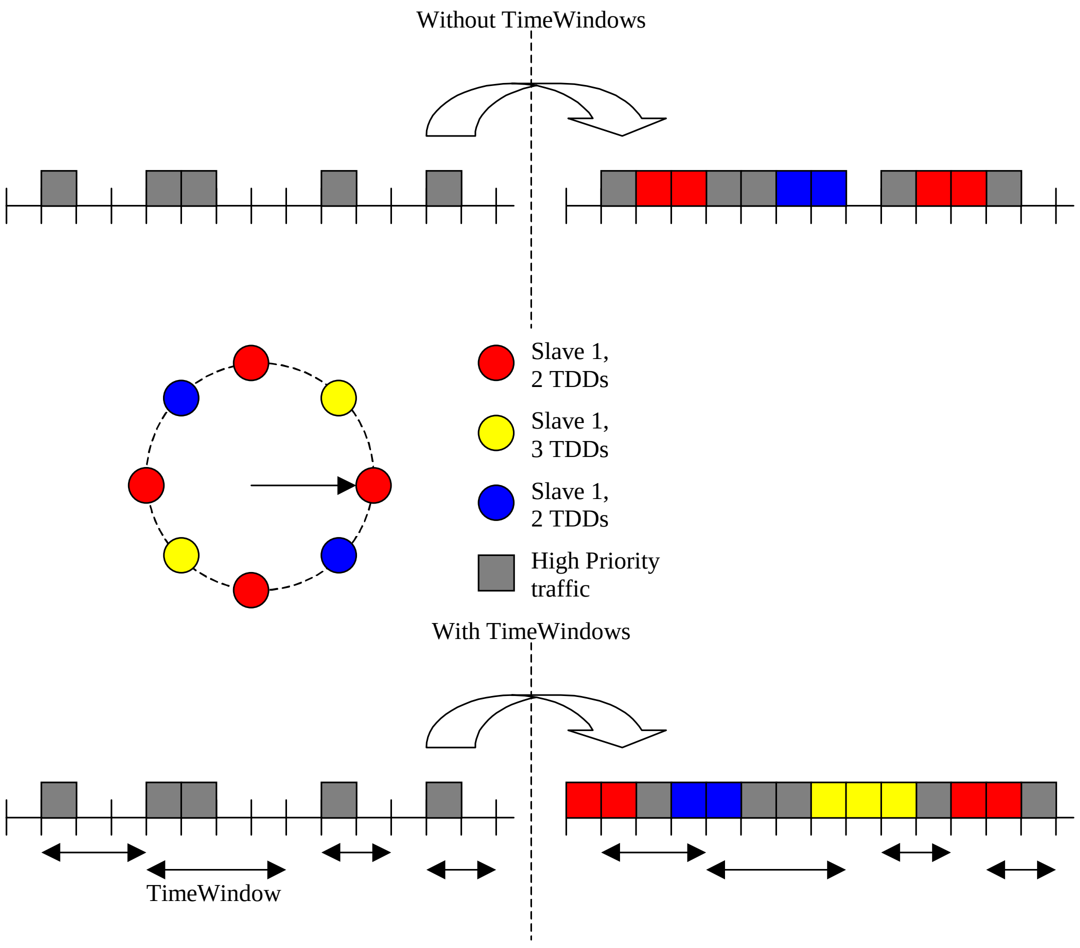


Figure 27. The advantage of using Time windows

7.6.1 Placement of the time windows

The placement and size of the time windows will have impact on the delay of the application and complexity of the algorithm. One possible way of placing the time windows is to have the window end at the position where the slave would have been polled without using time windows. Then when other traffic is scheduled the high priority traffic can be moved forward in the time window until a timer goes of and indicates that we have reach the end of the time window.

7.7 Insertion of slaves into the pre-scheduling window

The priority traffic is inserted into the longer pre-scheduling frame (see section 7.2). When it is time to prepare the next pre-scheduling window the retransmissions are then first added. The retransmissions are placed in the beginning of the pre-scheduling frame, pushing the Priority traffic forward within their time windows if necessary. What is than left of the pre-scheduling frame is given to the slaves in the ring. They are inserted sequentially until the pre-scheduling frame is filled. Pseudo code for the insertion process of ring slaves is given below.

```
Insert (slave at first empty position in the pre-scheduling frame)
If (insertion ok)
        return
else position occupied
        if (blocking traffic has time windows && is possible to move) insert and return
        else try to insert before
        insert afterwards if possible within the pre-scheduling frame
```

7.8 Master to slave traffic

If the master has traffic to send (to the slaves) that cannot be included in a one slot packet and sent instead of the polling messages this requiren an extension to the algorithm. If the down stream traffic is asymmetric and most of the traffic is going in the master to slave direction, this could be scheduled in very similar to the up stream traffic: BE traffic could be placed in the ring and QoS traffic could be sent when the packets arrives (the master has full buffer knowledge) using an exhaustive service. However, problems arises when there is traffic in both directions, thus there needs to be a duplex scheme where the flow with the most strict QoS requirements will determine the poll instances, in order to avoid as many void polls as possible. However, all of these would require extensions to the algorithm. Hence they are part of future work.

8 Simulations

A number of simulations evaluating the capabilities of the algorithm are described in this section. The simulations mainly evaluate traffic in the ring since the priority traffic is predictable and receives service at regular times and therefore there is not much to evaluate with respect to this traffic.

Fairness is a fundamental requirement of a scheduling algorithm. The definition of what is fair and what is not can vary, the viewpoint of fairness used in this masters thesis was described in section 4.1. For QoS traffic the algorithm is fair if nodes receive the agreed service (in terms of throughput and delay) negotiated by master and slave. Fairness for BE slaves are measured in the number of polls, thus the system is fair if active BE slaves each receive the same number of polls over time.

The error model used in these simulations is very simple, it is just random baseband packet loss induced by failed CRC check. The pre-scheduling frame is set to 28 slots in all simulations.

L2CAP delay and throughput are measured from when a packet is sent to L2CAP at one end and fully reassembled at the other end. As an example, the delay time for a L2CAP packet (slave to master transmission) that is divided into three baseband packets is depicted in Figure 28. As for the L2CAP throughput, the number of bits transferred is the payload data plus the headers appended by the diffeent layers as the data passes, i.e. due to UDP, TCP and L2CAP.

Detailed slave settings, load calculations, and information about the ns2 traffic generators used are given in the simulation descriptions found in Appendix B.

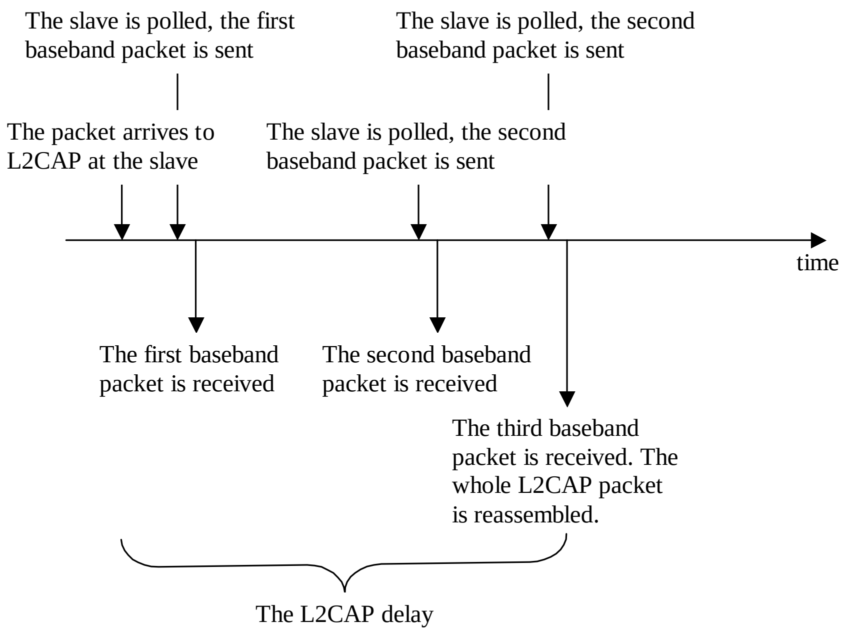

The slave is polled, the first baseband packet is sent
The slave is polled, the second baseband packet is sent
The packet arrives to L2CAP at the slave
The slave is polled, the second baseband packet is sent
time
The first baseband packet is received
The second baseband packet is received
The third baseband packet is received. The whole L2CAP packet is reassembled.
The L2CAP delay

Figure 28. The L2CAP delay

8.1 The simulator

The simulations were made in an ns2-based environment [35]. Ns2 , “Network Simulator version 2” has originally been developed at University of California at Berkeley. Ns is a discrete, event-driven simulator used to simulate different kinds of communication networks. Ericsson has built a Bluetooth simulator based on ns2. This simulator implements most of the Bluetooth specification, regarding packet exchange and network structure. The simulations are performed using C++ code with MIT’s Object Tool Command Language (OTcl) as interface to the user. Each simulation produces a trace-file describing when the packets where sent, to whom, the payload size, when a packet reached the upper layers, packet type, etc. By process these trace files we can calculate throughput and delay for flows.

8.2 Fairness between BE slaves

Considering a piconet with only BE slaves present, the algorithm is said to be fair if Equation 2 holds, i.e. if each BE slave has received the same amount of polls from the master looking over a period of time (t1,t2) where both are active. In this case since all BE slaves are assigned the same weight (equal to 1) the system will be reduced to a simple RR algorithm, thus each slave will receive an equal number of polls. A simulation to verify this is depicted in Figure 29 with corresponding results shown in Table 6. Figure 29 shows the throughput graph of 5 BE flows (flows 2 to 6). Each of the flows is modelled as CBR traffic the first three are using 5 slot packets and the other two 3 slot packets. The total piconet load is 0.9 and the simulations are performed in an error free environment. As the figure shows the slaves using the same packet size receive an equal share of the resource. The slaves using three slot packets each receive the same number of polls as the ones using 5 slot packets, as can be verified by normalising the average throughput with the payload size of the baseband packet used, which is shown in Table 6.

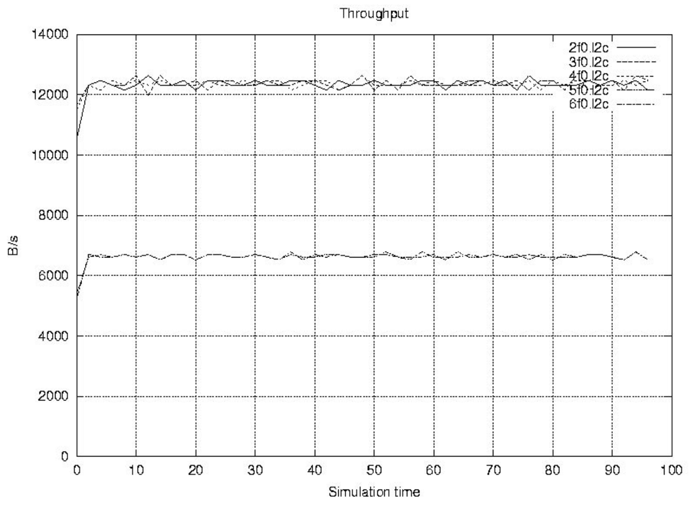


Figure 29. Fairness considering only BE slaves

Slave number	Avg. L2CAP throughput [Bytes/s]	Baseband packet type/Size[Bytes]	Avg. polls per second [s^{-1}]
2	12331	DH5/339	36.4
3	12354	DH5/339	36.4
4	12346	DH5/339	36.4
5	6617	DH3/183	36.2[8]
6	6616	DH3/183	36.2[8]

Table 6. The result of BE polling

[8] The difference of 0.2 in number of polls per second between the flows using 5 slot packets and 3 slot packets corresponds to the difference in utilisation of the baseband packet, see appendix B.3 for slave settings.

8.3 Fairness between guaranteed bandwidth traffic and BE slaves

When the piconet and the ring contain both BE slaves and QoS slaves fairness is upheld if the QoS slaves receive the bandwidth they have negotiated and the BE slaves equally share the remaining polls.

This simulation only considers slaves in the ring. There are two QoS slaves and three BE slaves. The first QoS slave, denoted slave number 2, has negotiated a guaranteed bandwidth of 155 kbit/s (i.e. 19.37 Kbytes/s), the second QoS slave, denoted slave number 3, has negotiated an agreed bandwidth of 128 kbit/s (i.e. 16 Kbytes/s). All BE slaves are set to always have data to send when polled as long as their connections exist. As Figure 30 shows the two QoS connections end before the end of the simulation this was done in order to see how the ring and the lead-lag algorithm (re-)distributes the bandwidth among the remaining slaves. The simulation presented in Figure 30 shows that flows 2 and 3 (the QoS flows) receive steady service near their negotiated guaranteed level, the lead-lag compensator upholds these service levels. When flow number 2 is disconnected after around 43 seconds the BE slaves receives the excess bandwidth. This occurs because slave number 3 reduces its number of elements in the ring in order to have the same share of the total number of elements in the ring as it had before (i.e. keeping the same level of bandwidth). This shows that the ring together with the lead-lag compensator results in a fair distribution of bandwidth. The reason for slave 6 receiving less bandwidth then slave 4 and 5 is because it uses a smaller baseband packet type DH3 compared to DH5 as slave 4 and 5 are using. It can be shown with the same method as in section 8.2 that the BE slaves have received the same number of polls.

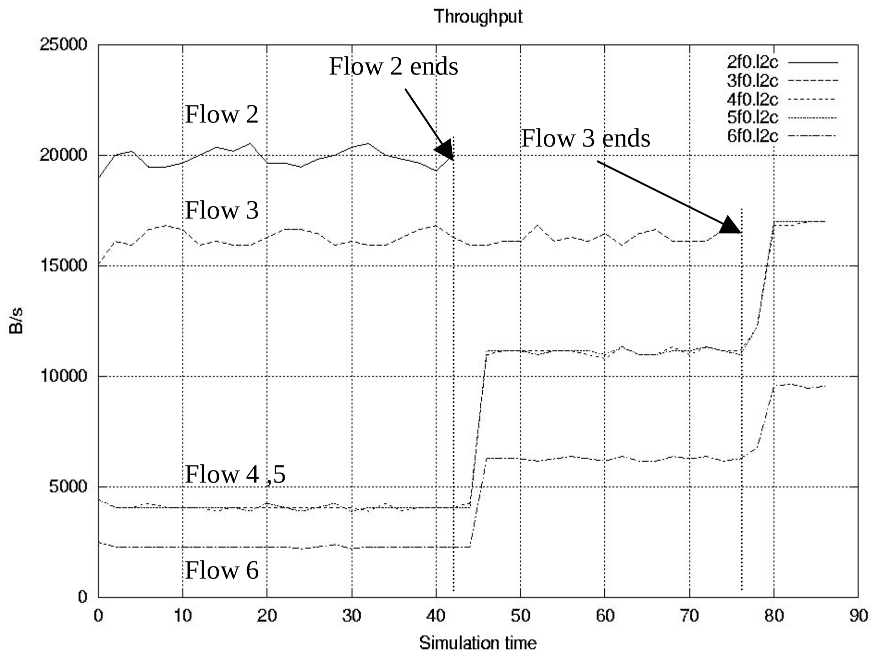


Figure 30. The fairness of the ring

8.4 Fairness in an error prone environment

QoS slaves have a higher priority compared to BE slaves which must share the remaining capacity when the QoS slaves have had their negotiated share of the resource. The same must hold true in an error prone environment. Therefore the best effort slaves will have to give up capacity when the amount of errors (causing retransmissions) and piconet load would exceed 100% of the total system capacity.

In this scenario we have one QoS slave and one to four BE slaves in the ring. The QoS slave has a negotiated bandwidth of 128 kbit/s and the BE slaves traffic is varied so that the load of the network ranges from 0.3 up to 0.9 of the total piconet capacity. All slaves are modelled as CBR sources since the idea of this scenario was simply to test the lead-lag compensators operation in an error prone environment, hence the burstiness of the slaves does not matter. In both Figure 31 and Figure 32 the simulation results are shown when there is no errors (dotted lines) and when on average 40% of the CRC checks fails (lines). The black lines represent the system total throughput/utilisation and the red line the QoS flows throughput/utilisation.

Figure 31 shows that the throughout for the QoS stream is kept at the negotiated level at the cost of BE traffic. The BE traffic starts to deviate from the error free curve when the load plus error level reaches 100% of the system capacity. The reason for the deviation occurring at around 0.55 (+0.4 = 0.95) instead of 0.6 is because we are using pre-scheduling. Pre-scheduling has the disadvantage that on occasion there will be unutilised slots.

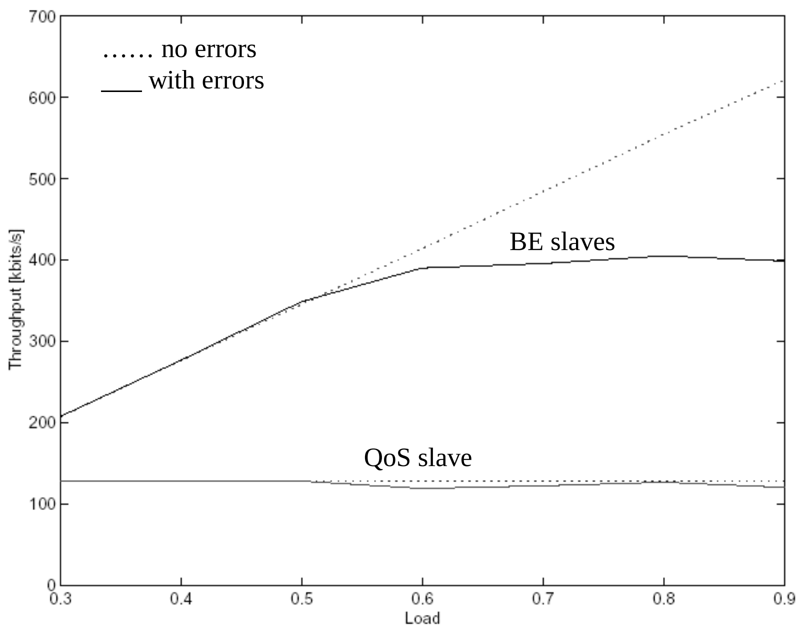


Figure 31. Lead lag behaviour in an error prone environment, throughput.

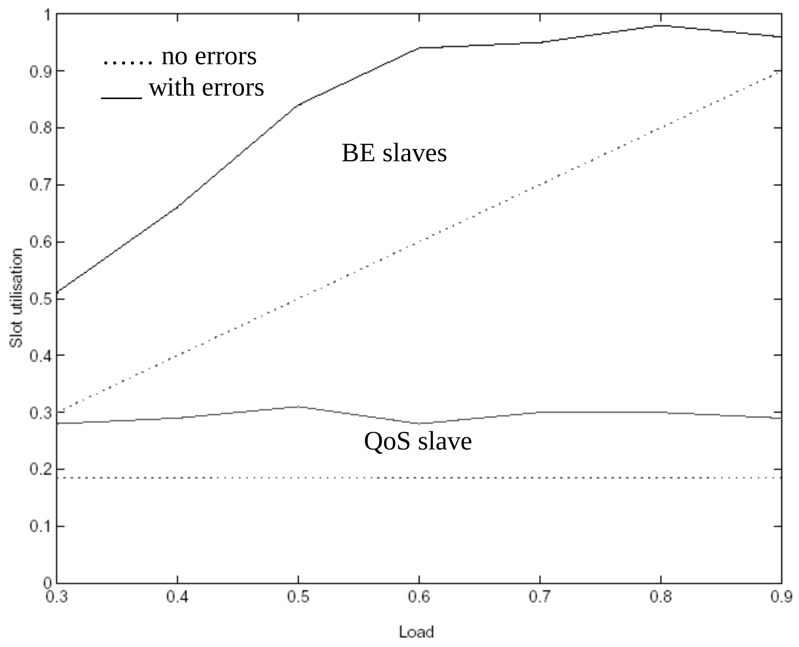


Figure 32. Lead lag behaviour in an error prone environment, slot utilisation.

8.5 Fairness among QoS slaves in the ring – test of the lead lag model

In this scenario five QoS slaves are present in the ring, all of them are modelled as VBR sources by using an exponential on off model (further described in appendix B.6) with different slave settings. The slaves had a negotiated bandwidth ranging from 60 kbit/s to 180 kbit/s as stated in Table 7; causing a mean network load of 93 %. All CRC checks were set to randomly fail in 20 % of all checks and thereby causing retransmissions of 20 % of all L2CAP data packets.

As we can see in Figure 33 all the slaves received a service that is lower then their negotiated level because offered network load is over 100% (20 % caused by retransmissions and 93 % caused by the original slave traffic). However the lead-lag model ensures that the all the QoS slaves service level degrade with to the same level, 9%, as shown in Table 7. Table 7 further states the slave's mean packet delay, the distribution in delay times is bursty since the sources are bursty and when passive the slaves have to wait 50 ms before receiving a new chance to send a packet (t_{poll} = 50 ms). The reason for an increasing packet delay for slaves with a lower negotiated bandwidth is that sending at a lower rate will be considered as passive more often and thus has to wait the 50 ms more often during which the slave's applications might produce new packets.

Slave	Negotiated bandwidth [kbit/s]	Given avg. bandwidth (L2CAP) [kbit/s],	Difference in % from negotiated value	Mean packet delay/ (0.05 confidence limits) [ms]
1	180	164.1	8.8 %	5.760 / (5.515, 6.005)
2	180	164.1	8.8 %	5.793 / (5.526, 6.060)
3	120	109.6	8.7 %	7.344 / (6.964, 7.725)
4	60	54.9	8.5 %	1.126 / (10,422, 12.089)
5	60	54.9	8.5 %	1.138 / (10.502, 12.215)

Table 7. Simulation data

As we can see the lead-lag model works reasonably well (however, not perfectly) under these conditions giving an evenly degraded level of service to the slaves.

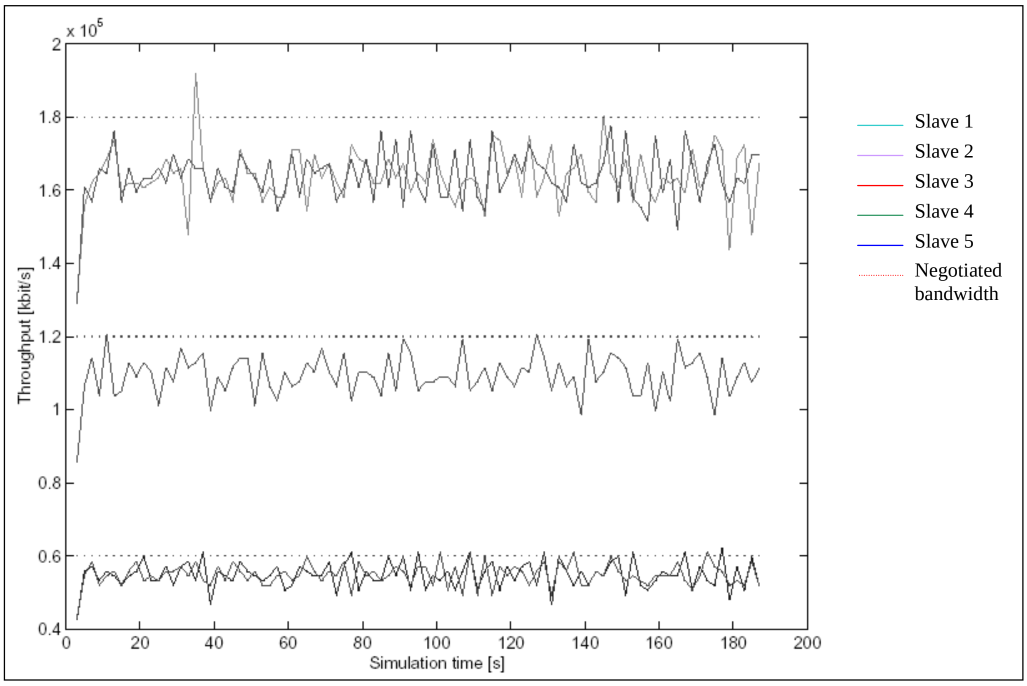


Figure 33. A sample simulation, the slaves throughput is depicted and also their negotiated bandwidth.

8.6 Delay using time Windows

This simulation shows the advantage of using time windows by plotting the cumulative density function (cdf) over the delay of BE slaves. This simulation consists of a piconet with 6 slaves, 2 priority traffic slaves and 4 bursty BE slaves. The detailed information is found in Table 8 and appendix B7. The priority slaves are a mouse and a MP3 player, both modelled as CBR sources. The BE slaves are bursty and modelled using an exponential on-off model. A number of simulations with and without using time windows where made (with different ns2 seed settings). From those simulations the L2CAP delay of the BE slaves were analysed. Figure 34 shows the analysis of this data in form of two cdf curves one where the slaves are using time windows and one without. Further the mean delay with corresponding confidence intervals from the two sample populations are given in Table 9. The result shows as expected an increased delay when the time windows not are used, this is because the pre-scheduling frame is not utilised as efficiently when using time windows. However, it should be noted that the are scenarios where the advantage of time windows is less than in this scenario, if there is less priority traffic then the advantage of time windows will be reduced but on the other hand if there is less traffic to handle with time windows, then the computation needed for time windows is also decreased. The reason for the slave's mean delay to be as large (40 –50 ms) is because we are transferring 3 baseband packets/L2CAP packet with a t_{poll} interval of 50 ms.

Slave nr	Interarrivl time [ms]	Packet size [bytes]	Baseband packet type /(# of base band packet /L2CAP packet)
1	10	4	DH1 / 1
2	50	834	DH5 / 3
3	34.2 (during on period)	820	DH5 / 3
4	34.4 (during on period)	820	DH5 / 3
5	34.2 (during on period)	820	DH5 / 3
6	34.2 (during on period)	820	DH5 / 3

Table 8. Some slave details

Time windows	Mean L2CAP delay [ms]	0.05 Confidence interval [ms]
Yes	0.0397	(0.03837, 0.04102)
No	0.0523	(0.05097, 0.05398)

Table 9. The mean delay with and without using time windows

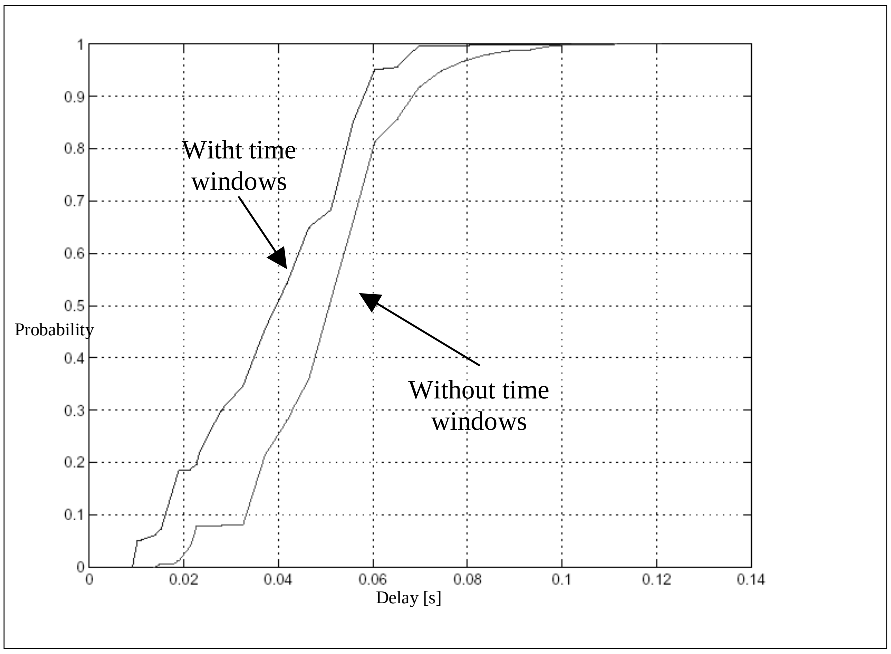


Figure 34. The cdf of the delays with (blue line) and without (green line) time windows

8.7 Simulation with the three different traffic categories present

This scenario considers a piconet with a mix of traffic types. The piconet contains one QoS slave is a MP3 player with regular inter-arrival times and fixed packet sizes modelled as a CBR source; this slave is scheduled as Priority traffic and given slots for regular transmissions. The ring consists of four slaves: two guaranteed bandwidth slaves and two BE slaves. All of these are modelled as bursty using an on off model, however the BE slaves are burstier (i.e. more active and passive periods) The guaranteed bandwidth slaves have a negotiated bandwidth of 140 kbit/s and 84 kbit/s respectively. More details about the different simulation settings are given in appendix B.9.

This simulation was is conducted in an error free environment which shows that the complete algorithm works when there are several different traffic types present. In Figure 35 we can see all of the slaves. It is important to note (indicated with an arrow in the figure) that the QoS group 1 slave receives a stead service level around 17 kbit/s, the L2CAP delay for this slave has a mean of 0.031 with 5% confidence limits of (Lower) 0.023 and (upper) 0.040. The reason for the 0.31 mean delay is that the polling calculations in section 7.3.1 yields that the slave should be polled every 13 slot pair and three based band packets should be transfered at slot pair 0, 13, 26 from the L2CAP packets arrival will yield the result. Furthermore the variance causing the 5% confidence limits is a result of the use of time windows set to 8 TDDs per poll, i.e. each poll instance can be moved 10 ms causing the variance.

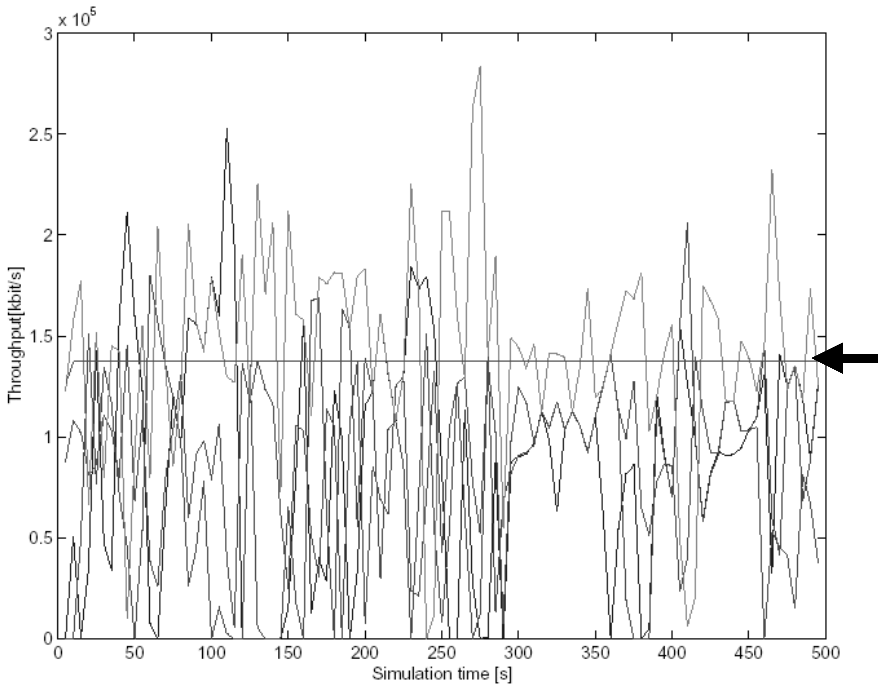


Figure 35. Simulation with a mix of traffic types

The average throughput (at the L2AP level) for all slaves is stated in Table 10. The reason that slave number one has a higher bandwidth than negotiated is that the slave has negotiated a poll interval calculated in section 7.3.1, which results in some overprovisioning. As the table shows the slaves receives a service around their negotiated value even if they are very bursty.

Slave/Type/scheduled as	Avg. L2CAP throughput [kbit/s]	Negotiated bandwidth [kbit/s]
1/QoS/Priority	137	133
2/QoS/Guranteed bandwidth	143	140
3/QoS/Guranteed bandwidth	86	84
4/BE/BE slave in the ring	79	na
5/BE/Be slave in the ring	87	na

Table 10. Throughput results for the simulation with mixed trffic types.

Figure 36 shows the total QoS traffics throughput vs. the BE slaves total throughput during the simulation. We can see that the two curves are mirror images of each other. When the QoS slaves (the curve with a higher throughput level, blue in colour) needs more bandwidth it is taken from the BE slaves. When the lead lag model reduces the poll level for the QoS slaves the bandwidth is given to the BE slaves yielding the mirror curve.

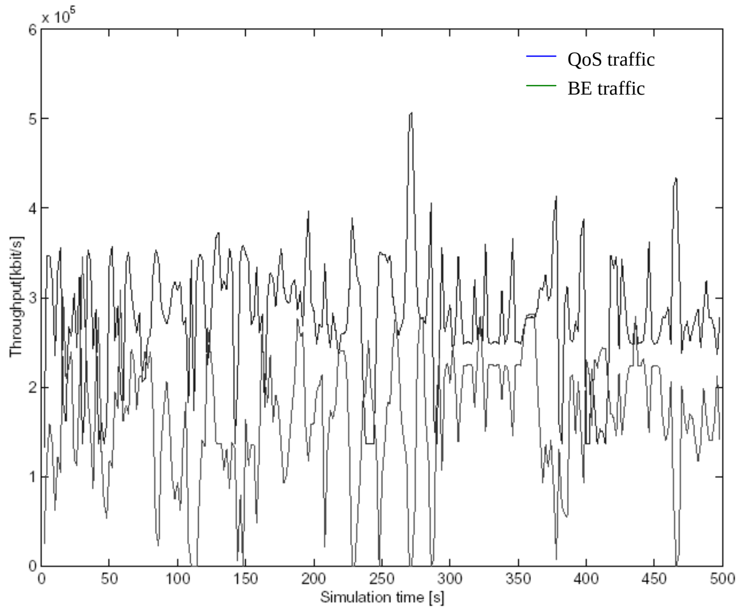


Figure 36. When active the BE traffic total throughput curve is a mirror image of the QoS traffic total throughput curve.

9 Conclusions

During the work of this master thesis an intra-piconet scheduling solution for Bluetooth (version 1.1) has been developed, presented, and tested. The algorithm is capable of maintaining QoS reservations at the same being fair and efficient. The simulations show that the theories work in practise, i.e. that the QoS reservations are upheld at the same time as fairness is achieved among the slaves in the system.

The simulations showed that Fairness is achieved among BE slaves, among QoS slaves, and when both BE and QoS slaves are considered. Furthermore fairness is also achieved when the link is degraded due to the occurrence of link errors.

In the care of a high load, as in the simulation in section 8.5, the slaves algorithm works and yields short delays and a high throughput. One efficient way of reducing the delays is to use time windows, simulations show significant advantage in terms of delays for the slaves using time windows since it increases the slot utilisation.

The algorithm is straight forward to implement and should be able to meet the low complexity requirement that concerns all parts in a Bluetooth system.

It should be noted that the evaluation of the algorithm did not handle all possible traffic types, rather a few that are likely to be frequently used in a Bluetooth network.

10 Future work

All parameters (*Max_number_of_elements*, T_{poll} values, pre-scheduling frame length etc.) associated with the algorithm needs to be evaluated in order to get an optimal or more optimal performance.

An optimal packet selection algorithm should be implemented that selects the packet type to be used the baseband transmissions. The packet selector should optimise the packet type used (single or multislotpackets, the level of error correction and detection) using delay and throughput as criterias.

Making the pre-scheduling more adaptive to changes, i.e. making it an semi pre-scheduling approach would lead to a system with more adaptivity. If a slave is using packets other than predicted or responds with a null packet should be utilised better. One possible approach

The fact that a Bluetooth node, participating in scatternets, is absent from one piconet while communicating in another increases the complexity of the scheduling algorithm. The increase of complexity is mainly due to two reasons, first: slaves should not be polled when absent from the piconet, secondly overall performance of the system is improved if PMP nodes can be compensated for the time that they are absent, i.e. while they are present in the piconet they should get an increased share of the bandwidth. This compensation could be included in the lead lag model, however the number of elements to include (the level of compensation) for absent PMP nodes returning to the Piconet must be calculated according to some as yet unknown algorithm. Compensation in relation to the absent time might be a starting point.

There is, as stated in section 2, ongoing work within the SIG to improve Bluetooths QoS capabilities. It is likely that we will see changes in the specification in future Bluetooth versions that have an enhanced QoS mechanism. If this is done and more parameters such as average inter-arrival times and average packet sizes can be negotiated between master and slave a more accurate algorithm can be devloped. Consequently making QoS schedulers will be ongoing and need updates as the SIG enhances the Bluetooth specification with more QoS capabilities.

11 References

[1] "***Specification of the Bluetooth System***", 22 February 2001, accessed 10 June 2002, Available from http://www.bluetooth.com/.

[2] Antonio Capone, Mario Gerla and Rohit Kapoor. "***Efficient Polling Schemes for Bluetooth picocells***". In proceedings of IEEE ICC 01, Helsinki, Finland, June 2001.

[3] Indraneel Chakaborty, Abhishek Kashyap, Anupam Rastogi, Huzur Saran, Rajeev Shorey and Apurva Kumar, "***Policies for Increasing Throughput and Decreasing Power Consumption in Bluetooth MAC***". In proceedings of ICPWC´2000, Hyderabad, India, December 2000.

[4] Abhishek Das, Abhishek Ghose, Ashu Razdan, Huzur Saran and Rajeev Shorey. "***Enhancing Performance of Asynchronous Data traffic over the Bluetooth Wireless As-hoc Network***". In proceedings of IEEE ICCCN 2002, Miami, Florida, October, 2002

[5] Jaap Haartsen. "***Bluetooth – The universal radio interface for ad hoc, Wireless connectivity***". Ericsson Review, No.3, 1998.

[6] Jaap C. Haartsen. "***The Bluetooth Radio System***". IEEE Personal Communications, Vol. 7, No 1, February 2000, pp. 28 –36.

[7] Jaap C. Haartsen. "***Bluetooth: a new radio interface providing ubiquitous connectivity***". In precedings of IEEE Vehicular Technology Conference (VTC), Tokyo, Japan, MAY 2000.

[8] Thomas M. Siep, Iac C. Gifford, Richard C. Braley and Robert F. Heile. *Paving the Way for Personal Area Networkstandars:* ***"An Overview of the IEEE P802.15 Working Group for Wireless Personal Area Networks"***. IEEE Personal Communications, February 2000

[9] Gerald Q. Maguire Jr. Lecture notes "***2G1330 Mobile and Wireless Network Architectures Bluetooth***" *14* March 2002, accessed 10 June 2002 from http://www.it.kth.se/edu/gru/NetArch/Lectures-2002/P4-Lecture7and8-2002.pdf;

[10] Rachid Ait Yaiz and Geert Heijenk ."***Polling Best Effort Traffic in Bluetooth***". In proceedings of 4th International Symposium on Wireless Personal Multimedia Communications (WPMC'01), Aalborg, Denmark, September, 2001.

[11] Martin van der Zee and Geert Heijenk. "***Quality of Service in Bluetooth Networking***". 03/01/2001. Accessed 29/10/2002. Available from: http://ing.ctit.utwente.nl/WU4/Documents/Bluetooth_QoS_ING_A_part_I.pdf

[12] Srinivasan Keshav. "***An engineering approach to computer networking, ATM networks, the Internet and, the Telephone network***", Addison-Wesley , 1st edition January 15, 1997. ISBN 0201634422

[13] Martin van der Zee. "***Quality of Service in Bluetooth Networking***". Ericsson report, Doc No. 13/0362-FCP NB 102 88 Uen 200-05-23.

[14] Lars Ahlin and Jens Zander. "***Principles of Wireless Communications*".** Second edition, Studentlitteratur 1997. ISBN 91-44-00762-0.

[15] Leander Kempenaars ."***Evaluation of rendezvous point inter-piconet scheduling***". Masters Thesis from Department of Telecommunication Thechnology and Electronics, Eindhoven University of Tecnology, the Netherlands. February 2002.

[16] Simon Baatz, Matthias Frank, Carmen Kuhl, Peter Martini and Christoph Scholz. "***Adaptive Scatternet Support For Bluetooth using Sniff Mode***" In proceedings of LCN 2001 IEEE, Tampa, Florida, November 2001.

[17] Songwu Lu, Vaduvur Bharghavan and R. Srikant. "***Fair Scheduling in Wireless Packet Networks***". In Proceedings of ACM SIGCOMM, Cannes, France. September 1997

[18] Reuters. "***Bluetooth—putting the bite on Wi-Fi?***". June 14, 2002.

[19] David Coursey. "***Bluetooth vs. WiFi: Why it´s NOT a death match***". AnchorDesk. May 29, 2002.

[20] CSR. "***Why Bluetooth?*".** http://www.csr.com/applications/bt-why.htm. Date accessed 29/10 –2002.

[21] Manish Kalia, Deepak Bansal and Rajeev Shorey. "***Data Scheduling and SAR for Bluetooth MAC***".IEEE MoMuC '99

[22] Vishwanath Sinha and D. Raveendra Babu. "***Class-based Packet Scheduling Policies for Bluetooth***". National Conference on Communications, Jan. 26 - 27, 2002.

[23] William Stallings. "***Data and Computer communications***". Fifth Edition. Prentice-Hall. 1997. ISBN 0-02-415425-3.

[24] Leandros Tassiulas and Theodoros Salonidis."***Distributed On-Line Schedule Adaptation for Balanced Slot Allocation in Bluetooth Scatternets and other Wireless Ad-Hoc Architectures***". Technical Research Report. CSHCN 2002-14. University of Maryland.

[25] Silicon Wave. White paper "***Bluetooth and 802.11 Compared***". November 26, 2001. www.siliconwave.com/pdf/ 73_0005_R00A_Bluetooth_802_11.pdf. Accessed 29/10/2002

[26] Jean-Baptiste Lapeyire and Thierry Turletti. "***Adding QoS Support for Bluetooth Piconet"***. Institut national de recherche en informatique en automatique. ISBN 0249-6399. July 2002.

[27] Vaduvur Bharghavan, Songwu Lu, and Thyagarajan Nandagopoal. "***Fair Queuing in Wireless Networks: Issues and Approaches***". IEEE Personal Communications, February 1999

[28] Changlei Liu, Kwan L. Yeung and Victor O.K. Li***. "A Novel MAC Scheduling Algorithm for Bluetooth System***". Department of electrical and Electronic Engineering, The University of Hong Kong. Submitted to INFOCOM 2003. Available from http://www.eee.hku.hk/~kyeung/piconet_ft.pdf Accessed 2003- 01 – 05.

[29] Infrared specifications. http://www.irda.org/standards/specifications.asp Accessed 2002 - 10 – 30.

[30] WLAN specifications: http://standards.ieee.org/catalog/olis/lanman.html Accessed 2002 – 10 –30.

[31] Siliconwaves "BLUETOOTH + WI-FI COEXISTENCE SOLUTIONS" http://www.siliconwave.com/coexistence.html. Accessed 2002 –10 -30

[32] Allied world. http://www.alliedworld.com/servlets/Home. Accessed 2002 –10 – 30.

[33] K. Pawlikowski, H-D. Joshua Jeong and J.-S. Ruth Lee. "***On credibility of Simulations studies of telecommunication networks***". Revision of the paper in IEEE Communications Magazine, April 2000.

[34] Niklas Johansson, Fredrik Alriksson, and Ulf Jönsson ***" JUMP Mode -A Dynamic Window-based Scheduling Framework for Bluetooth Scatternets***". SwitchLab, Ericsson Research. MobiHoc 2001.

[35] Ns2. http://www.isi.edu/nsnam/ns/. Date accessed 2002-11-03.

[36] Songwu Lu, Vaduvur Bharghavan, and R. Srikant. "***Fair Scheduling in Wireless Packet Networks***". Proceedings of ACM SIGCOMM 1997.

[37] Yang-Ick Joo, Jong Soo Oh, Oh-Seok Kwon, Youngsuk Kim, Tae-Jin Lee, and Kyun Hyon Tchah. "*An Efficient and QoS-aware Scheduling Policy for Bluetooth*". In proceedings of VTC fall 2002, Vancouver, British Columbia, September 2002.

[38] Yonghe Liu, Stefan Gruhl, and Edward W. Knightly. "***WCFQ: an Opportunistic Wireless Scheduler with Statistical Fairness Bounds***". To appear in IEEE Transactions on Wireless Communication. Last accessed from http://www-ece.rice.edu/networks/publications.html 2003-01-02.

Appendices

Appendix

A List of abbreviations

Abbreviation	Complete word	Short description
ACL	Asynchronous Connection-Less link.	Bluetooth data link type
AM_ADDR	Active member Address	The 3-bit address used for identifying Bluetooth slaves within a piconet.
B-FEP	Batched Fair Exhaustive Polling Scheduler	Scheduling algorithm in Bluetooth
BD_ADDR	Bluetooth Device Address	The unique address for a Bluetooth device.
BE	Best Effort	Traffic type.
CDF	Cumulative density function	Describes the probability that a trial X takes on a value less than or equal to a number x.
CRC	Cyclic Redundancy Check	For detecting transmission errors.
DHx	Data High rate	Bluetooth packet carrying only data, at a high rate, x = the number of slots for the packet duration.
DMx	Data Medium rate	Bluetooth packet carrying only data, at a medium rate, x = the number of slots for the packet duration.
DV	Data Voice	Bluetooth packet carrying both data and voice.
ERR	Exhaustive Round Robin	Polling scheme where the polled pair is allowed to transmit all their buffered packets in one sequence.
FEC	Forward Error Correcting code	Error correcting Code.
FEP	The Fair Exhaustive Polling Scheduler	Scheduling algorithm in Bluetooth
FHS	Frequency hop synchronisation	Bluetooth control packet used for exchanging information vital in among other things the page, and inquiry processes.
GFSK	Gaussian Frequency Shift Keying	The modulation scheme used for the transmitted radio signal in Bluetooth.
HEC	Header-error-check	The header checksum, to check the correctness of the received header.
HVx	High quality Voice	Bluetooth packet carrying voice, x = the number of slots for the packet duration.
ISM	Industrial-Scientific-Medical band	The 2.4 GHz band where the Bluetooth transceiver operates.
L2CAP	Logical Link Control and Adaptation Protocol	Provides connectionless and conection-oriented data services to upper layer protocols.
MTU	Maximum Transfer Unit	The maximum allowed packet size on a specific path.
NULL	Null	A Bluetooth control packet used to return link information of the source (sent from slave to master).

PAN	Personal Area Networks	A network carried by one person, e.g. a mobile phone and its connected Bluetooth headset.
PFP	The Predictive Fair Poller	Scheduling algorithm in Bluetooth.
PMP	Participating in Multiple Piconets	Name for a node participating in more then one piconet.
PS	Polling Scheduler	Entity in the PMP scheduler
PSM	PMP Slave Manager	Entity in the PMP scheduler
RR	Round Robin	Simple scheduling algorithm.
SCO	Synchronous Connection-orientated link	Circuit switched data link between a master and a slave in Bluetooth.
SIG	Bluetooth Special Interest Group	The standardization group formed to develop the Bluetooth standard.
TDD	Time Division Duplex	The Bluetooth channels are divided into slots using a TDD scheme in order to get a full duplex transmission. A TDD equals two slots (2 * 0.625 ms) which could be a Poll/null sequence.
WRR	Weighted Round Robin	A basic scheduling algorithm.

B Simulation data

B1 Used ns2 traffic support

Traffic over UDP:

Exponential:	Generates traffic according to an Exponential On/Off distribution. Packets are sent at a fixed rate during on periods, and no packets are sent during off periods. Both on and off periods are taken from an exponential distribution. Packets are constant size. Used to simulate bursty BE slaves and VBR traffic.
CBR:	Generates traffic according to a deterministic rate. Packets are constant size. Optionally, some randomising dither can be enabled on the interpacket departure intervals. Used to simulate CBR traffic such as HID or MP3 players**.**

Traffic over TCP:

FTP:	Simulates bulk data transfer. Used to simulate FTP traffic.

B2 Scenario description and slave settings

NOTE: if ns2 slave settings are not stated here they are the ns2 default setting, which can be found in the ns2 manual [35] and ns2 code.

For all scenarios the pre-scheduling frame length is set to 14 TDDs, this can however be dynamically extended as describe in the thesis.

If not stated the memory for the QoS slaves is set to 0.5 s.

B3 Fairness between BE slaves slave settings

```
set cbr_(2) [new Application/Traffic/CBR]
$cbr_(2) set packetSize_ 300
$cbr_(2) set interval_ 0.0262

set cbr_(3) [new Application/Traffic/CBR]
$cbr_(3) set packetSize_ 300
$cbr_(3) set interval_ 0.0262

set cbr_(4) [new Application/Traffic/CBR]
$cbr_(4) set packetSize_ 300
$cbr_(4) set interval_ 0.0262

set cbr_(5) [new Application/Traffic/CBR]
$cbr_(5) set packetSize_ 150
$cbr_(5) set interval_ 0.0262

set cbr_(6) [new Application/Traffic/CBR]
$cbr_(6) set packetSize_ 150
$cbr_(6) set interval_ 0.0262
```

B4 Fairness between both QoS and Be slaves

Guaranteed bandwidth traffic:

1 CBR source	176 kbit/s	Using:	DH5 packets
1 CBR source	64 kbit/s	Using:	DH5 packets

BE traffic

2 BE slave	Using:	DH5 packets
1 BE slave	Using:	DH3 packets

Tpoll interval: 50 ms

```
# QoS 128 kbit/
set cbr_(2) [new Application/Traffic/CBR]
$cbr_(2) set packetSize_ 330
$cbr_(2) set interval_ 0.017
$cbr_(2) set random_ 0
$cbr_(2) set maxpkts_ 1000000000
```

```
# BE - QoS 64 kbit/s
set cbr_(3) [new Application/Traffic/CBR]
$cbr_(3) set packetSize_ 330
$cbr_(3) set interval_ 0.020625
$cbr_(3) set random_ 0
$cbr_(3) set maxpkts_ 1000000000
```

The three BE slaves are set to always have data to send.

B5 Fairness in an error prone environment

The packet size was set to 300 bytes for all slaves, and all slaves are of type CBR, $cbr_(X) set packetSize_ 300, set cbr_(x) [new Application/Traffic/CBR], x = (2, 3, 4 ,5 6). All the slaves are using DH5 packets.

The load is calculated as the number of (theoretically) utilised TDD frames/total amount of available TDD frames.
Ex. Load = 0.3
Over 1 s are there 800 TDDs (1/1.25 ms = 800 per second)
cbr_(2):1 / 0.0205 = 49.34 packets/s => 49.4 * 3 = 148 TDDs /s
cbr_(3):1/0.03265 = 30.6 packets/s => 30.6 * 3 = 92 TDDs /s

utilisation: (148 + 92) / 800 = 0.3.

Load = 0.3
```
$cbr_(2) set interval_ 0.02025
$cbr_(3) set interval_ 0.03265
```

Load = 0.4
```
$cbr_(2) set interval_ 0.02025
$cbr_(3) set interval_ 0.0349
$cbr_(4) set interval_ 0.0349
```
Load = 0.5
```
$cbr_(2) set interval_ 0.02025
$cbr_(3) set interval_ 0.035735
$cbr_(4) set interval_ 0.035735
$cbr_(5) set interval_ 0.035735
```

Load = 0.6
```
$cbr_(2) set interval_ 0.02025
$cbr_(3) set interval_ 0.0362
$cbr_(4) set interval_ 0.0362
$cbr_(5) set interval_ 0.0362
$cbr_(6) set interval_ 0.0362
```

Load = 0.7
```
$cbr_(2) set interval_ 0.02025
```

```
$cbr_(3) set interval_ 0.0291
$cbr_(4) set interval_ 0.0291
$cbr_(5) set interval_ 0.0291
$cbr_(6) set interval_ 0.0291
```

Load = 0.8

```
$cbr_(2) set interval_ 0.02025
$cbr_(3) set interval_ 0.0243
$cbr_(4) set interval_ 0.0243
$cbr_(5) set interval_ 0.0243
$cbr_(6) set interval_ 0.0243
```

Load = 0.9

```
$cbr_(2) set interval_ 0.02025
$cbr_(3) set interval_ 0.021
$cbr_(4) set interval_ 0.021
$cbr_(5) set interval_ 0.021
$cbr_(6) set interval_ 0.021
```

B6 Fairness among QoS slaves in the ring – test of the lead lag model

Network load

Over 1 s there are 800 TDDs available (1/1.25 ms = 800 per second)
vbr_(1): 180kibt/s and 300*8 bit/packet => 75packets/s => 75 * 3 (DH5 packets with a poll takes 3 TDDs) = 225 TDDs/s.
vbr_(2): 225 TDDs/s
vbr_(3): 150 TDDs/s
vbr_(4): 75 TDDs/s
vbr_(5): 75 TDDs/s
(225 +225+150+75+75)/800 = 93%

Guaranteed bandwidth traffic:

2 VBR sources 180kbit/s Using: DH5 packets (1 baseband packet / L2CAP packet)
2 VBR sources 180kbit/s Using: DH5 packets (1 baseband packet / L2CAP packet)
2 VBR sources 180kbit/s Using: DH5 packets (1 baseband packet / L2CAP packet)

Tpoll interval: 50 ms
The simulator was seeded with seed = 5

Bandwidth memory = 0.5 s
Max_number_of_elements (in the ring) =15
20 % of the CRC checks failed randomly
All slaves were using DH5 packets

```
# QoS 180 kbit/s
set vbr_(1) [new Application/Traffic/Exponential]
$vbr_(1) set packetSize_ 300
$vbr_(1) set burst_time_ 1
$vbr_(1) set idle_time_ 0.001
$vbr_(1) set rate_ 181800


# QoS 180 kbit/s
set vbr_(2) [new Application/Traffic/Exponential]
$vbr_(2) set packetSize_ 300
```

```
$vbr_(2) set burst_time_ 1
$vbr_(2) set idle_time_ 0.001
$vbr_(2) set rate_ 181800


# BE  120 kibt/s
set vbr_(3) [new Application/Traffic/Exponential]
$vbr_(3) set packetSize_ 300
$vbr_(3) set burst_time_ 1
$vbr_(3) set idle_time_ 0.001
$vbr_(3) set rate_ 121200

# BE  60 kibt/s
set vbr_(4) [new Application/Traffic/Exponential]
$vbr_(4) set packetSize_ 300
$vbr_(4) set burst_time_ 1
$vbr_(4) set idle_time_ 0.001
$vbr_(4) set rate_ 60060


# BE 60 kibt/s
set vbr_(5) [new Application/Traffic/Exponential]
$vbr_(5) set packetSize_ 300
$vbr_(5) set burst_time_ 1
$vbr_(5) set idle_time_ 0.001
$vbr_(5) set rate_ 60060
```

B7 Delay using time windows

Priority traffic:

1 HID (mouse) pollinterval: 8 TDDs Using : DH1 packet
1 MP3 player pollinterval: 13 TDDs Using: DH5 packets (3 baseband packets / L2CAP packet)
Time window size: 6 TDDs (6 * 1.25ms).

BE traffic

4 Bursty slaves Using: DH5 packets (3 baseband packets / L2CAP packet)

Tpoll interval: 50ms

The simulations where conducted using different seeds {0, 5, 10, 15, 20, 25, 30, 35} with a duration of 200 s each.

```
# HID
set cbr_(1) [new Application/Traffic/CBR]
$cbr_(1) set packetSize_ 1
$cbr_(1) set interval_ 0.01
$cbr_(1) set random_ 0
$cbr_(1) set maxpkts_ 10000000

# MP3 - player DH5
set cbr_(2) [new Application/Traffic/CBR]
$cbr_(2) set packetSize_ 834
$cbr_(2) set interval_ 0.05
$cbr_(2) set random_ 0
$cbr_(2) set maxpkts_ 10000000000

#BE
set cbr_(3) [new Application/Traffic/Exponential]
$cbr_(3) set packetSize_ 820
$cbr_(3) set burst_time_ 3
```

```
$cbr_(3) set idle_time_ 9
$cbr_(3) set rate_ 192000

# BE
set cbr_(4) [new Application/Traffic/Exponential]
$cbr_(4) set packetSize_ 820
$cbr_(4) set burst_time_ 4
$cbr_(4) set idle_time_ 14
$cbr_(4) set rate_ 192000

# BE
set cbr_(5) [new Application/Traffic/Exponential]
$cbr_(5) set packetSize_ 820
$cbr_(5) set burst_time_ 5
$cbr_(5) set idle_time_ 18
$cbr_(5) set rate_ 192000


# BE
set cbr_(6) [new Application/Traffic/Exponential]
$cbr_(6) set packetSize_ 820
$cbr_(6) set burst_time_ 3
$cbr_(6) set idle_time_ 8
$cbr_(6) set rate_ 19200
```

B 9 The traffic mix

Priority traffic:

1 MP3 player pollinterval: 13 TDDs Using: DH5 packets (3 baseband packets / L2CAP packet)

Time window size: 8 TDDs (6 * 1.25ms).

Guaranteed bandwidth traffic:

1 VBR source 140kbit/s Using: DH5 packets (1 baseband packet / L2CAP packet)

1 VBR source 84 kbit/s

BE traffic

2 Bursty slaves Using: DH5 packets (1 baseband packets / L2CAP packet)

Tpoll interval: 100 ms

The simulator was seeded with seed = 5

```
# MP3 player
set cbr_(1) [new Application/Traffic/CBR]
$cbr_(1) set packetSize_ 834
$cbr_(1) set interval_ 0.05
$cbr_(1) set random_ 0
$cbr_(1) set maxpkts_ 10000000000

# VBR QoS slave - 140 kibt/s
set vbr_(2) [new Application/Traffic/Exponential]
$vbr_(2) set packetSize_ 300
$vbr_(2) set burst_time_ 5
$vbr_(2) set idle_time_ 2
$vbr_(2) set rate_ 196000

# VBR QoS slave – 84 kbit/s
set vbr_(3) [new Application/Traffic/Exponential]
$vbr_(3) set packetSize_ 300
$vbr_(3) set burst_time_ 4
```

```
$vbr_(3) set idle_time_ 2
$vbr_(3) set rate_ 126000

# BE
set be_(4) [new Application/Traffic/Exponential]
$be_(4) set packetSize_ 300
$be_(4) set burst_time_ 8
$be_(4) set idle_time_ 8
$be_(4) set rate_ 158400

# BE
set be_(5) [new Application/Traffic/Exponential]
$be_(5) set packetSize_ 300
$be_(5) set burst_time_ 8
$be_(5) set idle_time_ 8
$be_(5) set rate_ 158400
```