Adding QoS Support for Bluetooth Piconet

Jean-Baptiste Lapeyrie, Thierry Turletti

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Adding QoS Support for Bluetooth Piconet

Jean-Baptiste Lapeyrie — Thierry Turletti

N° 4514

Juillet 2002

THÈME 1

ISSN 0249-6399 ISRN INRIA/RR--4514--FR+ENG

Adding QoS Support for Bluetooth Piconet

Jean-Baptiste Lapeyrie [*] , Thierry Turletti [†]

Thème 1 — Réseaux et systèmes
Projet Planete

Rapport de recherche n° 4514 — Juillet 2002 — 52 pages

Abstract: Bluetooth is an emerging standard for short range, low cost, low power wireless access technology. The Bluetooth technology is just starting to appear on the market and there is an urgent need to enable new applications with real time constraints to run on top of Bluetooth devices. The Bluetooth Specification proposes a Round Robin scheduler as possible solution for scheduling the transmissions in a Bluetooth Piconet. However, this basic scheme performs badly under asymmetric traffic conditions. Recently, several polling schemes have been proposed to improve performance on asymmetric transmissions and to support bandwidth guarantee. However, there is no solution available to support both delay and bandwidth guarantees required by real time applications. In this paper, we present FPQ, (Fair and efficient Polling algorithm with QoS support) a new polling algorithm for Bluetooth Piconet that supports both delay and bandwidth guarantees and aims to remain fair and efficient with asymmetric flow rates. We present an extensive set of simulations and provide performance comparisons with other polling algorithms. Our performance study indicates that FPQ, while supporting flow rate and maximum delay QoS requests, outperforms Deficit Round Robin [6] in term of delays by at least 10% in all cases, sometimes by more than 30% to 50%. Moreover, FPQ was designed to take the specifics of Bluetooth into consideration, in particular the low complexity required for cheap implementation.

Key-words: Bluetooth Piconet, Data Scheduling, Fairness, Medium Access Control (MAC), Polling algorithm, QoS, Time Division Duplex (TDD).

[*] jean-baptiste.lapeyrie@polytechnique.org
[†] turletti@sophia.inria.fr

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : +33 4 92 38 77 77 — Télécopie : +33 4 92 38 77 65

Support de Qualité de Services pour les réseaux Piconet de Bluetooth

Résumé : Bluetooth est un standard émergeant en tant que technologie d'accès sans fil à courte portée, bas coût et faible puissance. Alors que la technologie Bluetooth commence juste à apparaître sur le marché, il y a un besoin urgent de permettre à de nouvelles applications avec des contraintes temps-réel de fonctionner sur des périphériques Bluetooth. La norme Bluetooth propose d'utiliser l'algorithme Round Robin comme solution d'ordonnancement des transmissions dans un Piconet. Comme cet algorithme a des performances médiocres avec un trafic asymétrique; plusieurs algorithmes de polling ont été proposés dans la littérature afin d'améliorer le support de transmissions asymétriques et d'apporter une certaine garantie de bande passante. Dans ce document, nous décrivons FPQ, un nouvel algorithme pour les réseaux Piconet de Bluetooth, qui offre des garanties de service en bande passante et délai maximum. Ces nouveaux services sont nécessaires pour les applications temps-réel et visent à rester efficaces et équitables même avec des débits asymétriques. Nous présentons un grand nombre de simulations et fournissons une comparaison de performance avec d'autres algorithmes. L'étude de performance montre que FPQ, tout en supportant des requêtes de QoS, améliore les résultats de l'algorithme Deficit Round Robin ([6]) en terme de délai d'au moins 10% et parfois de plus de 30% à 50%. De plus, FPQ a été conçu en considérant les spécificités de Bluetooth, en particulier la faible complexité requise pour une implémentation bon marché.

Mots-clés : Piconet de Bluetooth, Ordonnancement des données, Equité, Contrôle d'Accès au Medium, Algorithme de polling, QoS , Time Division Duplex (TDD)

Contents

List of Figures

Chapter 1

Introduction

1.1 Bluetooth

1.1.1 Overview

Formed in February 1998 by mobile telephony and computing leaders Ericsson, IBM, Intel, Nokia, and Toshiba, the Bluetooth Special Interest Group (SIG) is designing a royalty-free technology specification, with the aim of enabling seemless voice and data communication via short-range links and allowing users to connect a large range of devices easily and quickly, without the need for cables. Thus, Bluetooth is an emerging standard for short range, low cost, low power wireless access technology and today, the Bluetooth interface offers a full wireless networking solution for ad-hoc networks. Figure 1.1 illustrates the Bluetooth stack, as defined in the Bluetooth Specification [8] designed by the SIG.

The Radio layer, operating in the 2.4 GHz Industrial, Scientific and Medical (ISM) band, provides the physical channel among Bluetooth devices, with a gross bit rate of 1 Mb/s, and features low energy consumption for use in battery operated devices. This physical layer, which uses 79 different Radio Frequency (RF) channels, utilizes as technique of transmission a Frequency Hopping Spead Spectrum (FHSS), where the pseudo-random hopping sequence is unique for each ad-hoc network. The Baseband layer manages the transport service of packets on this physical link and the synchronization of devices, while the Link Manager Protocol (LMP) performs the connection set-up and management of physical links. Higher level protocols transmit and receive data through the Logical Link Control and Adaptation Protocol (L2CAP), which implements features like protocol multiplexing, Segmentation And Reassembly (SAR). A Host Controller Interface (HCI) provides a command interface to the Baseband, the LMP, and access to hardware.

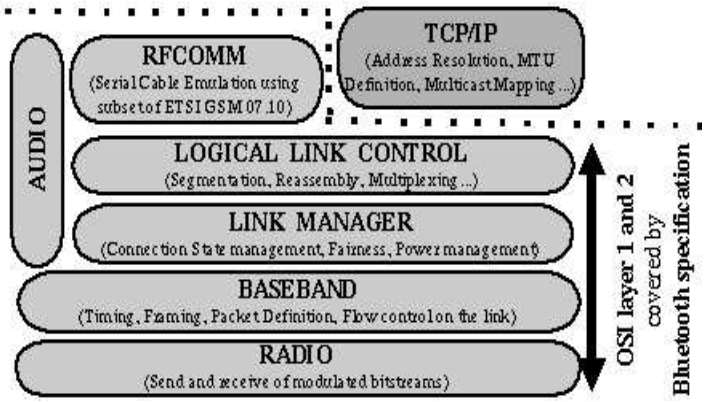

Figure 1.1: Bluetooth Stack

1.1.2 MAC Scheduling

At the start of a connection, the initializing unit is assigned as a master. A Piconet is the network formed by a master and one or more slaves, but only up to 7 active slaves (i.e., take part in the data exchange). Transmissions can take place from master to slave (*downstream traffic*) or from slave to master (*upstream traffic*).

In order to support full duplex transmissions, a Time Division Duplex (TDD), which divides each second into 1600 time slots, is adopted in the MAC layer located in the Baseband. The transmission of a Baseband packet usually covers a single slot but it may last up to five consecutive time slots. Therefore, the master fully controls the traffic in the Piconet; indeed, a slave is allowed to transmit a Baseband packet only if in the previous time slot the master has sent him a Baseband packet. Hence, when the master sends data to a slave, it gives the slave the opportunity to transmit data back as well. When the master has no data to send, it may poll the slave with a packet without payload (called a *POLL* packet). Then, the slave has to respond by sending back data, if available, or by sending a packet with no payload (called a *NULL* packet). This mechanism is illustrated in Figure 1.2. A critical point in the efficiency of Bluetooth Piconet is the management of the limited available bandwidth of Bluetooth. Consequently, the MAC protocol scheduling algorithm (i.e., the polling scheme) is a very important feature for determining the Piconet's efficiency, because it decides the order in which Bluetooth units are polled, and therefore the order in which Bluetooth units send or receive data.

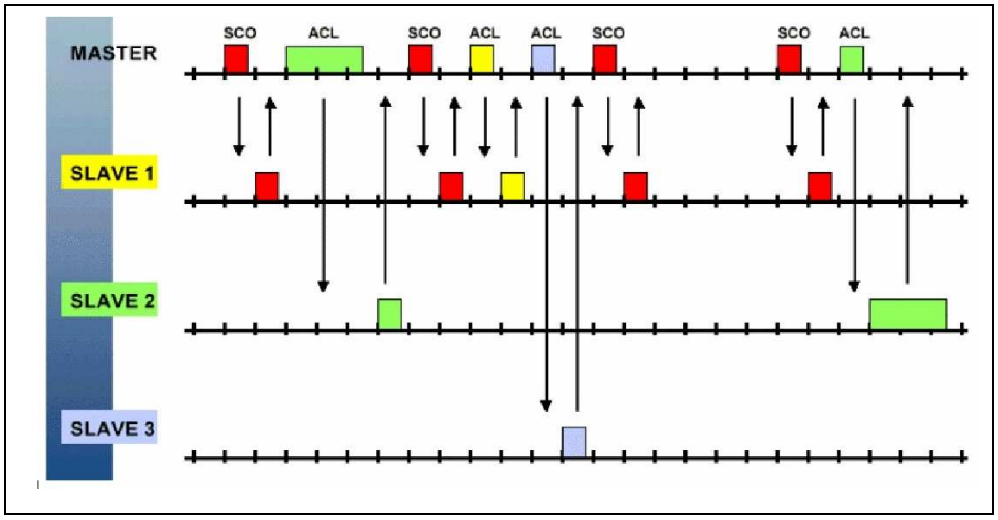

Figure 1.2: Bluetooth MAC Scheme : Polling

1.2 Issues to support QoS

Bluetooth can support simultaneous voice and data communications with only limited Quality of Service (QoS) support. Two types of concurrent services are supported: Synchronous Connection Oriented links (SCO) and Asynchronous Connectionless Links (ACL) that have characteristics common with respectively circuit switched and packet switched services. The SCO link aims to carry real-time traffic such as voice traffic. It uses a slot reservation mechanism which allows the periodic transmission of SCO data with guaranteed delay and bandwidth. However, it is not flexible and can only provide a fixed symmetric bandwidth (at most 64 kb/s). Such a service is not efficient for real time and delay sensitive applications like streaming audio and video that may require variable and asymmetric bandwidth. Moreover, the SCO has limited error detection and correction capabilities, and no retransmission mechanism, which makes it inefficient when bit errors occur in bursts. On the other hand, the ACL link was originally designed to support data applications. It is based on a polling algorithm between a master and up to seven active slaves. The ACL bandwidth is determined by the ACL packet type and the frequency with which the device is polled. It can provide both symmetric (up to 432 kb/s) and asymmetric (up to 721 kb/s) bandwidth. Moreover, it can offer reliability in the presence of interference (even when bit errors occur in bursts) using both Forward Error Control (FEC) and retransmission mechanisms. Ac-

tually, since the delay involved with retransmissions is short [1] in this type of network, the ACL link could also be used to transmit real time applications. For all these reasons, we will consider only the ACL link in the remainder of the document.

1.3 Aim of the work

The aim of this paper is to enable Bluetooth to support new services with asymmetric bandwidth and delay guarantees. Given the characteristics of wireless networks, quantitative QoS guarantees (i.e. hard guarantees) can not be provided through the Bluetooth layer. However, new MAC scheduling algorithms can be designed to support *relative guarantees* [7]. Indeed, upcoming Bluetooth applications such as audio/video streaming require more QoS support. As we have said above, no algorithm for polling has been specified by the Bluetooth Special Interest Group.

Through this report, we propose and evaluate a Fair Poller with QoS support (FPQ) algorithm, which aims to remain fair and efficient in presence of asymmetric flow rates. We add two new QoS requests: a Flow Rate (FR) request and a Maximum Delay (MD) request, since these are important QoS features not supported by Bluetooth.

The rest of the paper is organized as follows. Chapter 2 overviews the polling algorithms designed for Bluetooth. Chapter 3 describes in details our FPQ polling algorithm. Then, Chapters 4, 5, and 6 introduce the three simulation models used to study and compare FPQ with other scheduling algorithms, and evaluate the performance of FPQ through several simulations. Concluding remarks are summarized in Section VI.

[1] In Bluetooth Piconet, an acknowledgment can be received within 1.25ms.

Chapter 2

Related Works

We introduce different polling algorithms which are already used in some Bluetooth implementations, such as the Round Robin, or which have been designed especially for Bluetooth.

2.1 The Round Robin algorithm

As mentioned previously, the Bluetooth Specification proposes a simple Round Robin (RR) policy as a potential polling algorithm. Such a basic scheduling policy supports a one-limited service discipline that equally divides the total number of polls among the slaves without taking into account the different requirements of each slave. As a consequence, some slaves are polled much more than necessary, while high-traffic slaves may be polled less. If the overall traffic is low, or if the flow rates are similar, this kind of poller handles traffic well. But, in any other situation, the RR poller is very inefficient [2].

2.2 The Defi cit Round Robin algorithm

The Deficit Round Robin (DRR) algorithm [6] behaves almost like RR, but gives higher priority to active slaves with backlogged data (in downstream traffic as well as in upstream traffic) and therefore improves the behavior of the RR algorithm. For example, with only upstream traffic, the DRR algorithm polls each slave until it has no more data, which corresponds to reception of a NULL packet from the slave. In this case, DRR stops polling a slave as soon as this slave sends back a NULL packet. Compared to RR, the improvements of the DRR are that it takes into account downstream as well as upstream traffic and that it can handle asymmetric flow rates among the slaves. However, the behavior of

the DRR algorithm reveals some weakness. For example, if a lot of packets are waiting in a slave's queue, DRR becomes unfair on short time scale since it polls this slave until its queue becomes empty. Meanwhile, the other slaves of the Piconet wait with their data, which increases the delays of the packets in their queues.

2.3 The Predictive Fair Poller

The Predictive Fair Poller (PFP) [5] allows handling of the FR QoS requests. Furthermore, this poller implements interesting ideas about methods for handling upstream traffic and asymmetric flow rates, e.g., a the probability that a slave has data packets waiting in its queue, knowing the flow rates of upstream traffic. Nevertheless, this algorithm is especially designed for upstream traffic and does not handle MD requests.

2.4 The Sticky-Adaptive Flow-based Polling (Sticky-AFP) algorithm

The Sticky-AFP [3] enables high traffic rate and low delays in the Piconet but is designed only for downstream traffic. it does not support any QoS request. Nevertheless, in our second simulation model, the results obtained by the Sticky-AFP [3] constitute a reference for the evaluation of FPQ.

2.5 The Effi cient Double-Cycle polling algorithm

The Efficient Double-Cycle (EDC) polling algorithm [2] outperforms RR by dynamically adapting the polling frequency to traffic conditions. However, EDC differentiates upstream traffic from downstream traffic and does not take into account any QoS request.

Chapter 3

The FPQ scheme

When comparing different polling algorithms, the main criteria to take into account as required by Bluetooth are efficiency, fairness, and low complexity. Efficiency implies transmission of the maximum amount of data: the maximum possible number of time slots should be used for data transmission. As a consequence, time slots should not be wasted by using too many POLL and NULL packets. On the other hand, fairness requires fair share of bandwidth and fair distribution of delays among the different slaves, independently of the flows' characteristics. Furthermore, polling algorithms have to be as independent as possible of other layers in order to easily fit into the Bluetooth stack.

We first describe and discuss the ideas and the principles of FPQ. Then, we analyze in details its features and discuss its complexity.

3.1 Role of HCI

First of all, as the connection requests are made through the Host Controller Interface (HCI), the QoS requests have to be accepted by the HCI, in the master side, before establishing connections. Applications should provide their Flow Rate (FR) requests. In fact, a FR request includes an average Interval of Time between two consecutive Packets (ITP) and an average Packet Length (PL). Applications may provide different parameters so that the HCI can compute the couple of parameters (ITP,PL). On the other hand, PL should represent the number of time slots required by the transmission of a packet. Thus, PL takes into account the Segment And Reassembly (SAR) policy of the L2CAP for the segmentation of the initial packet in Baseband packets. The MD request corresponds to the maximum end-to-end delay the application can support. We note $(ITP_{s,i}, PL_{s,i}, MD_{s,i})$ the parameters

for flows from slave i to master, and $(ITP_{m,i}, PL_{m,i}, MD_{m,i})$ for flows from master to slave i.

The HCI has to decide if the Piconet is able to handle all the requests, with 1600 time slots per second. We chose to compute the number of slots required by the QoS requests in the worst case, i.e., in the case which requires the maximum number of time slots for the satisfaction of the QoS requests. We assume that between two Baseband packets, the master (the corresponding slave) sends a POLL (NULL) packet. For example, if an IP packet requires a 5-slot and a 3-slot Baseband packet, the HCI can decide that it will use 8 time slots for data transmission and 2 time slots for acknowledgement traffic, taking consequently 10 time slots for its entire transmission. Then, the HCI computes the total number of slots required for the satisfaction of the FR requests. As we will see in Section 3.3.2, the MD requests imply supporting, in the worst case, a POLL and a NULL packet every $MD_{s,i}$ seconds [1]. Finally, the HCI calculates the total number of time slots necessary for supporting all QoS requests in the worst case. If the Piconet can handle all traffic requests, the HCI forwards the parameters to FPQ, which will try to satisfy these requests, as independently as possible of other layers. In other cases, we could imagine some QoS negotiation between the HCI and applications in order to establish communications with lower QoS requests.

3.2 General Description

3.2.1 Objectives and issues

FPQ tries to reconcile both efficiency and fairness objectives. Knowledge of the different flow rates of applications should permit a better distribution of the polls among the slaves. This should help to limit the number of NULL and POLL packets used, which improves the efficiency of the polling algorithm. On the other hand, FPQ has to pay attention to the differences of delays among similar flows, i.e. flows with the same MD request. From a fairness point of view, all similar flows should have almost the same distribution of delays.

Moreover, if we decide (as in [2]) to separate both types of traffic (i.e. upstream and downstream traffic), this may increase the probability of having a POLL or a NULL packet sent for each master-slave transmission. As a result, some time slots will be wasted with Baseband packets without data. On the other hand, if we mix both types of traffic, the delays may increase. For example, if slave i is polled with 5-slot Baseband packets rather than POLL packets (i.e. 1-slot packets), its end-to-end delay will be larger. Therefore, to avoid a waste of time slots, FPQ mixes upstream and downstream traffic together.

[1] MD requests for downstream traffi c do not inevitably imply a waste of time slots.

A packet sent by an application on top of a Bluetooth unit will be denoted by an AP packet, independently of the type of this packet. When the number of sources is high, triggering several transmissions of AP packets in the same time can become an issue. The more AP packets are transmitted in the same time, the longer will be the transmission time. Consequently, an important issue is to complete the transmission of an AP packet before beginning the transmission of another AP packet.

3.2.2 Principles and description

We note $Q_{s,i}$ the queue of slave i and $Q_{m,i}$ the queue of the master for packets towards slave i. We remind that in the Baseband layer of the master, there is a queue for each active slave.

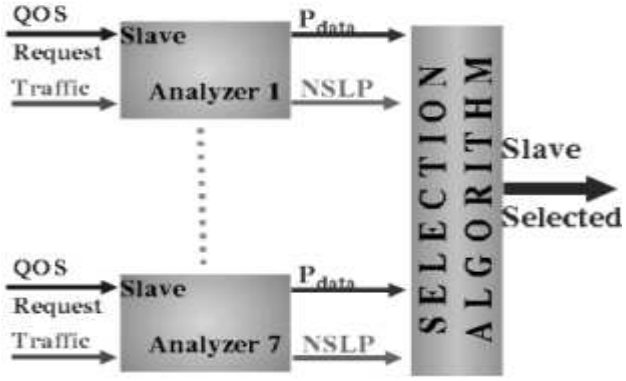

Figure 3.1: Polling scheme at the master side

Using the ITP from the QoS requests, the state of $Q_{m,i}$ and the characteristics of the traffic through the Baseband layer, a Slave Analyzer computes:

1. An instantaneous probability of having data in $Q_{s,i}$ ($P_{data\ s,i}$).
2. An instantaneous probability of having data in $Q_{m,i}$ ($P_{data\ m,i}$).
3. A Number of Slots since Last Poll ($NSLP$) for upstream traffic ($N_{s,i}$).
4. And for downstream traffic ($N_{m,i}$).

Then, each Slave Analyzer provides to the Selection Algorithm two parameters P_{data_i} and N_i, defined as follows:

$$P_{data_i} = P_{data\ s,i} + P_{data\ m,i} \tag{3.1a}$$

$$N_i = N_{s,i} + N_{m,i} \tag{3.1b}$$

Selecting the slave with the highest P_{data_i} increases the chance of having data packets transmitted; thus, P_{data_i} is directly linked to efficiency. N_i should permit to limit the delays for each slave, and is linked to fairness. Using all P_{data_i} and N_i, the Selection Algorithm decides which slave has the highest priority.

Since $0 \leq P_{data_i} \leq 1$ and $N_i \geq 1$, the Selector calculates first $\sum_{Active\ Slaves} P_{data,i}$ (which is greater than 0 as long as there is more than one active flow) and $\sum_{Active\ Slaves} N_i$, in order to obtain the following normalized parameters:

$$p_{data_i} = \frac{P_{data_i}}{\sum_{Active\ Slaves} P_{data_i}} \quad n_i = \frac{N_i}{\sum_{Active\ Slaves} N_i} \tag{3.2}$$

Now, we have $0 \leq p_{data_i} \leq 1$ and $0 < n_i < 1$. As we have pointed out above, efficiency is represented by p_{data_i}, whereas fairness is represented by n_i. In order to control both efficiency and fairness, we have defined the priority Pr_i as follows:

$$Pr_i = \alpha * p_{data_i} + (1 - \alpha) * n_i \tag{3.3}$$

The parameter α controls the tradeoff between fairness and efficiency. To be efficient, α must be close to 1. To increase fairness, and if the traffic of Piconet does not require too much efficiency, α must be close to 0. We believe that the parameter α should be set by the HCI, and forwarded to the MAC Layer. It would be then possible to design algorithms in the HCI to decide the value of α with respect to the traffic and requests.

Once the selector has decided which slave has the highest priority, the master will send a data (or a POLL) packet to the selected slave, which in turn will send back a data (or a NULL) packet. By selecting slave i, we allow traffic to flow towards slave i and then to flow from slave i.

The following section details the computation of the different values $P_{data\ s,i}$, $P_{data\ m,i}$, $N_{s,i}$ and $N_{m,i}$.

3.3 Computation of P_{data_i} and N_i

3.3.1 $P_{data\ s,i}$ calculation with only FR request

Each Slave Analyzer assumes that the arrival times of AP packets from the application follows a Poisson process. We believe this is a realistic assumption since the initial distribution is unknown, and the Poisson process remains one of the most unpredictable processes with respect to the arrival times of the packets.

When a slave answers with a NULL packet at time T_0, this means that there is no more AP packets waiting in its queue. According to the Poisson process, it becomes quite easy to determine the probability of having data at time $T_0 + \Delta T$. However, it is more difficult to determine this probability when the slave responds with a data packet at time T_0, because we do not know the state of its queue. Before describing our algorithm for determining these probabilities, we introduce the following notation. T_0 represents the last time at which the slave responded with a NULL packet. T_1,T_2,...,T_N, represent the beginning of the transmission of each AP packet, with $T_0 < T_1 < ... < T_N$. We note that $P_{\geq 1,N}$ is the probability of having one or more AP packets in the queue of the slave at time T_N. $P_{>1,N}$ stands for the probability of having strictly more than one AP packet in the queue at time T_N, and $P_{=1,N}$ for the probability of having exactly one AP packet in the queue, at time T_N. For all these probabilities, the only assumption is to have no AP packet at time T_0.

Additionally, we denote by $P(=1, T_N, T_{N+1})$ the probability that one AP packet arrives between T_N and T_{N+1}, $P(\geq 1, T_N, T_{N+1})$ the probability of having one or more than one AP packet between T_N and T_{N+1}, and $P(=0, T_N, T_{N+1})$ the probability of having no AP packet between T_N and T_{N+1}.

Initializing the number of AP packets in the queue to exactly one or zero at time T_0 yields the same probabilities from a Poisson process, so we choose the following initial values:

$$P_{=1,0} = 1 \quad P_{\geq 1,0} = 1 \quad P_{>1,0} = 0 \tag{3.4}$$

We have the following recurrence formulas:

$$P_{=1,N+1} = P(=1, T_N, T_{N+1}) * P_{=1,N} + P(=0, T_N, T_{N+1}) * P_{=2,N} \tag{3.5a}$$

$$P_{\geq 1,N+1} = P(\geq 1, T_N, T_{N+1}) * P_{=1,N} + P_{>1,N} \tag{3.5b}$$

$$P_{>1,N+1} = P_{\geq 1,N+1} - P_{=1,N+1} \tag{3.5c}$$

The term $P_{=2,N}$ prevents us from having a regular recurrence. In order to get rid of this term, me make the approximation that the ratio between $P_{=2,N}$ and $P_{=1,N}$ is equal to the ratio between the probabilities of having $N+1$ and N packets arrived in the time interval $[T_0, T_N]$:

$$\frac{P_{=2,N}}{P_{=1,N}} = \frac{P(\text{N+1 packets in } [T_0, T_N])}{P(\text{N packets in } [T_0, T_N])} = \frac{(T_N - T_0) * \frac{1}{ITP}}{N+1} \tag{3.6}$$

By substituting (3.6) in (3.5a) and computing $P(=1, T_N, T_{N+1})$, $P(=0, T_N, T_{N+1})$ and $P(\geq 1, T_N, T_{N+1})$ under the assumption that the arrival times of AP packets follow a Poisson process, we obtain the recurrence formulas, with $\Delta T = T_{N+1} - T_N$:

$$P_{=1,N+1} = \exp\left(-\frac{\Delta T}{ITP}\right) * \frac{P_{=1,N}}{ITP} * \left(\Delta t + \frac{T_N - T_0}{N+1}\right) \tag{3.7a}$$

$$P_{\geq 1,N+1} = \left(1 - \exp\left(-\frac{\Delta T}{ITP}\right)\right) * P_{=1,N} + P_{>1,N} \tag{3.7b}$$

$$P_{>1,N+1} = P_{\geq 1,N+1} - P_{=1,N+1} \tag{3.7c}$$

Now, we want to compute the probability of having one or more packets at time $t = T_{N+1} > T_N$, given that there were one or more AP packets at times T_1,T_2...T_N

$$\begin{aligned} P_{data\,s,i} &= P\left(\{\geq 1, N+1\} \,/\, (\{\geq 1, N\} \ldots \{\geq 1, 1\})\right) \\ P_{data\,s,i} &= \frac{P(\{\geq 1, N+1\} \ldots \{\geq 1, 1\})}{P\left(\{\geq 1, N\} \ldots \{\geq 1, 1\}\right)} \\ P_{data\,s,i} &= \frac{P_{\geq 1,N+1}}{P_{\geq 1,N}} \end{aligned} \tag{3.8}$$

By reporting (3.7b) in (3.8),we finally obtain :

$$\begin{aligned} P_{data\,s,i} &= \frac{\left(1 - \exp\left(-\frac{T_{N+1} - T_N}{ITP}\right)\right) * P_{=1,N} + P_{>1,N}}{P_{\geq 1,N}} \\ P_{data\,s,i} &= 1 - \exp\left(-\frac{t - T_N}{ITP}\right) * \frac{P_{=1,N}}{P_{\geq 1,N}} \end{aligned} \tag{3.9}$$

However, an AP packet, after its segmentation by the L2CAP, may require several Baseband packets to be fully transmitted. If the master of the Piconet started but has not finished receiving all Baseband packets of an AP packet from slave i, that means that Baseband packets of this AP packet are waiting in $Q_{s,i}$. Consequently, the master is sure that the next time it polls slave i, it will receive a Baseband packet with data. In that case, as the presence of data in $Q_{s,i}$ is sure, the probability must be:

$$P_{data\ s,i} = 1 \tag{3.10}$$

The poller must now determine when it has received the first segments of an AP packet but not the whole AP packet. A simple way is to compare the length of the Baseband packet and its maximum possible length. If they are different, the end of the transmission of the AP packet is then reached, otherwise the transmission of the AP packet will likely require at least another Baseband packet.

3.3.2 $P_{data\ s,i}$ calculation with full QoS request

As the efficiency of the poller is directly influenced by $P_{data\ s,i}$, we will use this parameter to improve the behavior of the polling algorithm.

We first analyze how to support the Maximum Delay request, transmitted through the parameter $MD_{s,i}$. As we do not know the arrival times of the AP packets in the queues of the slaves, one way to satisfy the new request is to make sure that the $Q_{s,i}$ is empty every $MD_{s,i}$ seconds. The poller knows that $Q_{s,i}$ is empty when it receives a NULL packet. Assume that at time T_0, slave i sends back a NULL packet. To fulfill the MD request, the poller must then receive a NULL packet from slave i at time $T_1 < T_0 + MD_{s,i}$. Then, it must receive another NULL packet before $T_1 + MD_{s,i}$. Else, there may be two different reasons: either slave i has always sent back data packets or it has not been polled in the meantime. In both cases, the poller has to poll slave i as soon as possible until it finally sends back a NULL packet. Let $T_1^{'}$ be the time this NULL packet is received by the poller, Slave i must now receive the next NULL packet before $T_1^{'} + MD_{s,i}$.

Thus, we introduce a new variable for each slave: the Interval of Time since Last Null packet received ($ITLN_i$).

When we are sure that a slave has data to send (i.e. an AP packet transmission is pending), $P_{Data,i}$ is set to 1, its maximum value. The MD request asks for higher priority for the corresponding slave. After experimenting with different approaches, we decided to define the new value for $P_{Data,i}$ as follows:

$$\hat{P}_{Data,i} = P_{Data,i} + 1_{\{ITLN_i \geq MD_{s,i}\}} \tag{3.11}$$

A potential problem appears when $\hat{P}_{Data,i} \geq 1$; but, as the Selection Algorithm normalizes each parameter, this is not a problem.

3.3.3 $P_{data\ m,i}$ calculation with only FR request

Looking at the states of $Q_{m,i}$ and following the previous scheme of $P_{data\ s,i}$, we may determine $P_{data\ m,i}$ as follows:

$$P_{Data\ m,i} = \begin{cases} 0 & \text{,when } Q_{m,i} \text{ is empty} \\ 1 & \text{,when } Q_{m,i} \text{ is not empty} \end{cases} \tag{3.12}$$

However, this method has some drawbacks. For example, assume that a packet arrives in $Q_{m,i}$ while this queue is empty. The following situation occurs:

	$P_{data\ s}$	$P_{data\ m}$	P_{data}
Slave i	1	0	1
Slave j	0.05	1	1.05

Whereas slave i has started transmission of an AP packet towards the master (because of $P_{data\ s,i} = 1$), the FPQ algorithm automatically sets the highest priority to slave j for transmission of an AP packet towards slave j. Indeed, the choice of slave j increases the probability of having data packets in both directions. The point is that as soon as a new packet arrives in $Q_{m,j}$, it receives the higher priority for transmission, interrupting the transmission of an AP packet from slave i. It would be more efficient to wait for the end of the transmission of the AP packet of slave i before polling slave j. Moreover, with this approach, we do not differentiate Baseband packets which correspond to the first segment of an AP packet from the other Baseband packets: consequently, this method is not able to provide any priority to transmissions of AP packets which are in progress.

We note $HoLP_{m,i}$ the Head-of-Line packet of $Q_{m,i}$ (i.e. the first Baseband packet of $Q_{m,i}$), and FSA, a packet which is the First Segment of an AP packet (i.e. the first Baseband packet of the transmission of an AP packet). We distinguish three possibilities, depending on the state of $Q_{m,i}$:

1. $Q_{m,i}$ is empty.
2. The $HoLP_{m,i}$ is not a FSA.
3. The $HoLP_{m,i}$ is a FSA.

We set $P_{data\ m,i}$ to 0 in the first case and to 1 in the second case. In the third case, since there are data remaining in $Q_{m,i}$, $P_{data\ m,i}$ should be greater than 0. Moreover, as we want FPQ to establish a distinction between FSA packets and the other Baseband packets, $P_{data\ m,i}$ should be lower than 1 in this third case. So, this value should influence the behavior of the poller without giving a higher priority to the data in $Q_{m,i}$. We have empirically set $P_{data\ m,i}$ to 0.5 in the third case:

$$P_{data\ m,i} = \left\{ \begin{array}{ll} 0 & \text{,when no packet is in } Q_{m,i} \\ 0.5 & \text{,when } HoLP_{m,i} \text{ is a } FSA \\ 1 & \text{,when } HoLP_{m,i} \text{ is not a } FSA \end{array} \right\} \tag{3.13}$$

3.3.4 $P_{data\ m,i}$ calculation with all QoS requests

We have already described and explained how to handle the MD request for upstream traffic. The principle for fulfilling this request is to ensure that $Q_{m,i}$ is empty every $MD_{m,i}$ seconds. The same scheme as above will be used to compute $P_{data\ m,i}$: FPQ calculates a new variable, the Interval of Time since last time $Q_{m,i}$ was Empty ($ITQE_i$). As for upstream traffic, we compute the new value of $P_{data\ m,i}$ as follows, as so to give the highest priority to this flow:

$$\hat{P}_{Data,i} = P_{Data,i} + 1_{\{ITQE_i \geq MD_{m,i}\}} \tag{3.14}$$

3.3.5 $N_{s,i}$ calculation

The $N_{s,i}$ parameter is used to estimate the amount of time between each opportunity the slave has to begin the transmission of an AP packet. Thus, this parameter allows harmonization of the delays among the active slaves of the Piconet. We therefore distinguish two types of Polls. The first type corresponds to the case where P_{data} is set to 1, meaning that the slave is transmitting an AP packet through several Baseband packets. However, these Polls do not bring any information about AP traffic, and do not allow the slave to transmit another AP packet, in case AP packets remain in the queue. On the other hand, the second type of poll, corresponding to P_{data} different from 1, is very useful since it provides information about AP traffic. Indeed, the state of $Q_{s,i}$ is advertised to the poller by the existence, or by the absence, of AP packets waiting for transmission. Only in this case has the slave an opportunity to begin transmission of an AP packet. Consequently, $N_{s,i}$ is set to 0 only in this second case and is always incremented at each slot.

3.3.6 $N_{m,i}$ calculation

We will use a similar scheme to $N_{m,i}$ to compute $N_{m,i}$. As soon as $Q_{m,i}$ is empty, $N_{m,i}$ is set to 0, since there are no data in $Q_{m,i}$. When there are data in $Q_{m,i}$, $N_{m,i}$ is incremented at each slot and set to 0 when slave i is selected and in the meantime, the $HoLP_{m,i}$ is the first segment of an AP packet.

3.4 Evaluation of Complexity

To enable cheap implementation of Bluetooth, the complexity of polling algorithms should be kept low. Our FPQ implementation requires some registers (about ten per slaves) used for the Slave Analyzers. The exponential expressions are the most greedy of CPU. However, we do not need many operations per active slave, as can be seen from describing the process in detail. For computation of Pr_i, each slave obtains $P_{data\ m,i}$, $N_{m,i}$ and $N_{s,i}$ by reading registers or flags. The calculation of $P_{data\ s,i}$ require at most 1 exponential calculation, 2 additions, 2 divisions and 1 multiplication. However, if $Q_{s,i}$ is not empty, $P_{data_{s,i}}$ is set to 1, without any calculation. After that, 2 additions are necessary to compute P_{data_i} and N_i. We can modify the expression of the probabilities to obtain:

$$Pr_i = P_{data_i} + \left(\frac{(1-\alpha)}{\alpha} * \frac{\sum_{ActiveSlaves} P_{data_i}}{\sum_{ActiveSlaves} N_i} \right) * N_i \tag{3.15}$$

where $\left(\frac{(1-\alpha)}{\alpha} * \frac{\sum_{ActiveSlaves} P_{data_i}}{\sum_{ActiveSlaves} N_i} \right) = Constant.$ This constant can be computed once for all slaves. Then, 2 operations for computation of Pr_i and one comparison for finding the slave with highest priority are required for each slave.

After selecting a slave, and receiving back the slave's Baseband packet, FPQ has to update some registers. FPQ only resets or increments some registers (for example $N_{s,i}$), except for the probabilities needed by $P_{data\ s,i}$ of the selected slave, which may require up to 1 exponential, 3 divisions, 3 multiplications and 6 additions. If the selected slave sends back a NULL packet, the probabilities are set to initial values. So, the number of operations to perform per slave is kept low. Moreover, as we have a maximum of 800 polls per second, we believe that current low cost chips are able to perform these operations.

3.5 Overview of simulation models and conditions

We use three different models of traffic to evaluate performance of FPQ. The three following sections first describe the simulation model used, then show and analyze the behavior of FPQ in these particular situations. The first model, called Upstream Traffic Model (UTM), simulates polling algorithms with only upstream traffic, asymmetric flow rates and high overall throughput. The second model, called Downstream Traffic Model (DTM) analyzes FPQ with only downstream traffic (i.e. with a reverse traffic compared to the previous simulation model) and with FTP traffic. The third model, called Mixed Traffic Model (MTM), mixes both upstream and downstream traffic, to evaluate the ability of FPQ to simultaneously handle these types of traffic.

We use the Network Simulator (NS2) [10] with Bluehoc extensions [9] from IBM to perform the simulations. We added some changes to Bluehoc and NS2 to allow traffic in both directions. In addition to the implementation of FPQ, we use the Bluehoc implementation of DRR, and our own implementation of PFP. Our simulations are available in [11].

For all the polling algorithms, we use the same SAR policy in the L2CAP layer as described in [3] (i.e. SAR-Optimum Slot Utilization (OSU), with *slot_limit*=5), because this policy performs well in terms of throughput, link utilization and end-to-end delay [3]. Moreover, in our simulations, all the buffers of the Bluetooth stack are considered as infinite. Nevertheless, if FPQ obtains small delays for packets in the simulations, it implies that small buffers are sufficient. We do not use any loss model for the transmission channel because we want to study our algorithm independently from the constraints and effects of other layers. A loss model would naturally introduce larger delays and lower throughput. However, thanks to the Bluetooth QoS parameters, we have the possibility of limiting the number of retransmission of Baseband packets. Moreover, this loss model does not have any additional negative effect on the behavior of FPQ. In the simulations, we assume that only one application per slave sends data. If there are many applications on top of one Bluetooth unit and after each application has forwarded its QoS requests, the HCI should summarize all these QoS requests as if there were only one application on this slave: the FR request would be equal to the sum of all FR requests, and the MD request would be equal to the lowest MD request.

PFP and FPQ have both a tuning parameter α. We choose to present each poller with a particular value, $\alpha = 0.8$, which achieves a good tradeoff between efficiency and fairness with both polling algorithms. All the results presented in this section are the average values obtained from four different simulations.

Chapter 4

Simulations with Upstream traffic Model (UTM)

4.1 Description of the UTM

DRR and PFP have been used for comparison in this model of traffic. We use the same simulation conditions as in [5]. There is only traffic from the slaves to the master, and only one application per slave that generates IP packets. Arrivals of IP packets follow a Poisson process. The Bluetooth Baseband packet types used are DH1, DH3, DH5 which respectively take one, three and five slots, and have payloads of 27 bytes, 183 bytes and 339 bytes. We use a trimodal distribution of IP packets size [4]. 30% of IP packets have a size of 40 bytes, 12% a size of 1500 bytes and 58% a size uniformly distributed between 300 bytes and 600 bytes.

Let $\bar{d}$ be the average number of data slots of IP packets, and $\bar{p}$ the average number of Poll packets required per IP packet, we have :

$$\bar{d} = 8.30 \quad \bar{p} = 1.99 \tag{4.1}$$

As in [5], the rates of the Poisson arrival process are as follows:

$$\lambda_3 = \lambda_4 = \lambda_5 = \lambda_6 = \lambda_7 = \lambda_b \tag{4.2a}$$

$$\lambda_1 = \lambda_2 = \lambda_a \geq \lambda_b \tag{4.2b}$$

and we have as well:

$$COV = \frac{\sqrt{\frac{\sum_i \lambda_i^2 - \frac{1}{7}\left(\sum_i \lambda_i\right)^2}{6}}}{\frac{1}{7}\sum_i \lambda_i} \quad with\ i \in \{Active\ Slaves\} \tag{4.3a}$$

$$\rho = \frac{\sum_i \lambda_i \bar{d}}{1600} \quad with\ i \in \{Active\ Slaves\} \tag{4.3b}$$

ρ represents the percentage of slots used only by data packets. All the simulations are made with $\rho = 0.7$. COV represents the coefficient of variation of the seven flow rates. $COV = 0$ means that all flow rates are the same, and, the higher the COV is, the larger is the difference between λ_a and λ_b. Due to equations (4.2) and (4.3), each combination of ρ and COV gives a unique solution for the arrival rates $(\lambda_1, ..., \lambda_7)$. Thus, with $\rho = 0.7$, it is possible to use COV as input for the simulations.

We introduce MD requests for each slave, and all QoS requests are summarized in Table 4.1

Table 4.1: QoS Requests summary

SLAVES	S_1	S_2	S_3	S_4	S_5	S_6	S_7
λ used	λ_a	λ_a	λ_b	λ_b	λ_b	λ_b	λ_b
MD (ms)	100	200	200	300	300	300	500

COV	0	0.2	0.4	0.6	0.8	1.0	1.2
$\lambda_a\ (s^{-1})$	19.29	24.93	30.58	36.22	41.37	47.51	53.16
$\lambda_b\ (s^{-1})$	19.29	17.03	14.77	12.51	10.25	7.99	5.73
$ITP_a(in\ ms)$	51.85	40.11	32.70	27.61	23.88	21.05	18.81
$ITP_b(in\ ms)$	51.85	58.73	67.71	79.93	97.54	125.10	174.38

According to these QoS requests, using the same calculation as the HCI, defined in section 3.1, 91% of the slots are used for data transmission and satisfaction of MD requests as well. Thus, the Piconet is highly loaded. Finally, the simulations, as in the MTM model, have a duration of 1200 seconds.

4.2 Results with the UTM

4.2.1 Analysis of Throughputs

We first analyze the overall throughput of the three polling algorithms to ensure that they handle all the traffic. As we can observe on Figure 4.1, the three pollers are able to handle the high traffic of this model with very low variations (less than 1%), certainly introduced by the random model of the simulations. We now analyze more precisely how each polling algorithm distributes the polls among the slaves.

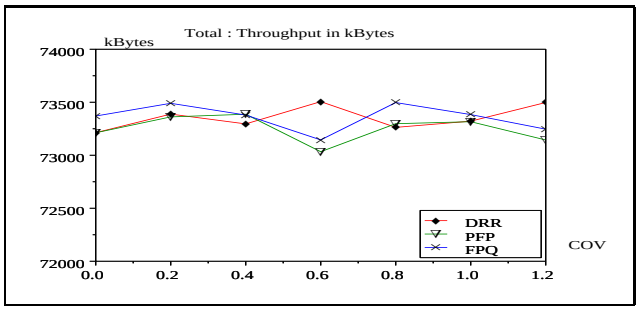

Figure 4.1: Overall throughput versus COV

4.2.2 Analysis of Polls

The way each poller divides the number of polls helps in understanding the behavior and the priorities of polling algorithms. Since slaves with high throughput have to be polled frequently, we analyze the ratio of good polls for each slave, i.e. the number of Polls with remaining data packet in the following slot, divided by the total number of Polls.

Efficiency leads to a total number of polls for each slave proportional to its flow rate and therefore to the same ratio of good polls for each slave. As a consequence, slaves with low flow rates are polled less often than other slaves, so their packets may wait a longer time before being transmitted. Therefore, in order to increase fairness, it is necessary to send more polls to these slaves, which will obtain lower ratio of good polls.

Figure 4.2 confirms that slaves are differentiated by the polling algorithms through their flow rates. All the schemes show continuous increases of their values with respect to COV for slaves 1 and 2, and continuous decreases for slaves 3,5 and 7. One may notice that PFP and FPQ try to be more efficient, and thus perhaps less fair than DRR, because they poll more often high rate slaves (i.e. slaves 1 and 2), and consequently less often low rate slaves.

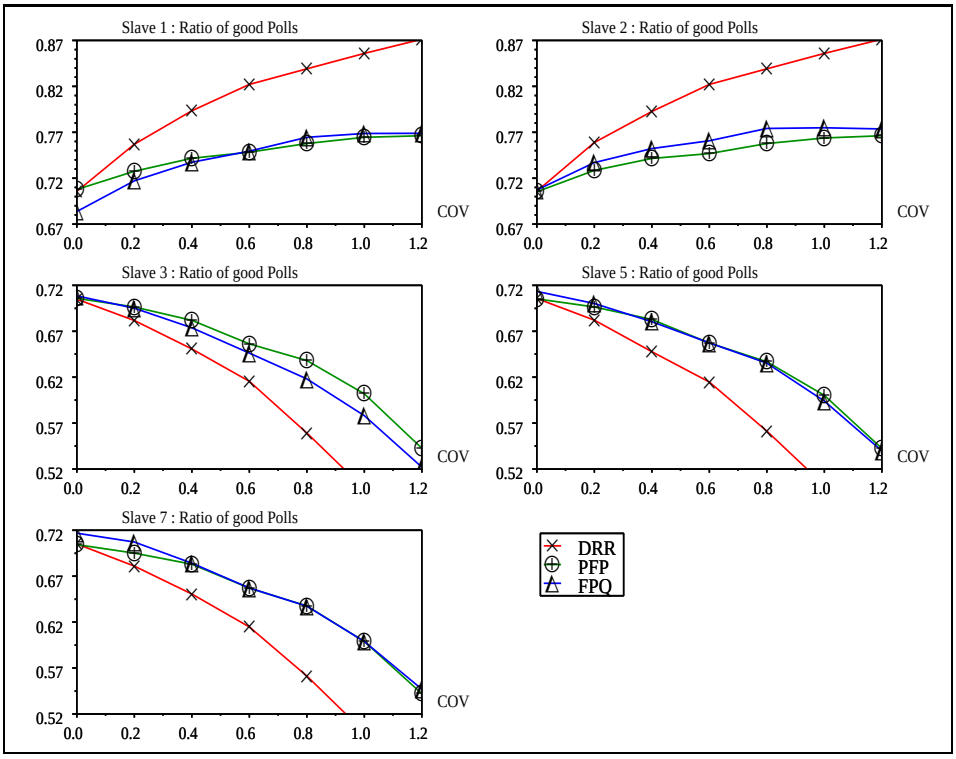

Figure 4.2: Distribution of good polls versus COV

As we expected, slaves with low $MD_{s,i}$ have been favored by FPQ, principally Slave 1 and to some extent Slave 3, since they have been polled more often than other slaves. FPQ has consequently given high priority to the slaves requesting a low $MD_{s,i}$. Nevertheless, the other slaves have not been too much penalized; they have only lost some priority.

4.2.3 Analysis of Delays

We only consider end-to-end delays of IP packets because they are the most relevant delays for applications on top of Bluetooth units. We analyze the evolution of the average delays for all packets, with respect to the COV value, and then for Slaves 1,2,3 and 7, which have all different QoS requests. As each poller handles correctly all the traffic, efficiency is evaluated through the average delay of all packets. Then, comparison of delays between the slaves provides an estimation of the fairness of each poller: a low MDs, i should naturally lead to a low average delay for slave i.

Although FPQ supports QoS requests, Figure 4.3 reveals that FPQ performs better than the other polling schemes, improving the overall average delay with a gain of around 10%. By looking at this overall average delay we notice that PFP and FPQ are more sensitive to the evolution of COV: consequently, FPQ adapts itself better to the changes of traffic conditions. As the distribution of polls has shown, PFP and FPQ are less fair than DRR. Nevertheless, FPQ remains quite fair, and provides the same enhancement for the overall average delay. Moreover, we can clearly observe the effects of $MD_{s,i}$ requests on the average delays. For example, if we consider Slaves 1 and 2, they have the same FR request but different MD requests: Slave 1, with a low $MD_{s,1}$, has been consequently favored. In the same way, Slave 3 has been favored compared to Slave 7; yet, Slave 7, which has the highest $MD_{s,i}$, has not been too much disadvantaged compared to the other slaves.

Although average delays characterize well the behavior and the efficiency of each poller, the distribution of delays brings other important information such as the percentage of packets which fulfill the MD requests. So, we present distributions of delays for several slaves on Figure 4.4, where each curve represents, with respect to time t, the number of IP packets sent by the slave with a delay less than t, divided by the total number of IP packets sent by this slave. For lack of space, we have decided to present the results with the mean value of COV, i.e. $COV = 0.6$. However, the results remain very similar with different values of COV.

Figures 4.4 and 4.5 confirms the previous results reported on efficiency and fairness for each polling algorithm. We observe how FPQ differentiates the slaves in order to satisfy the QoS requests. One may notice the efforts made to obtain high values for each slave at the critical time $t = MD_{s,i}$.

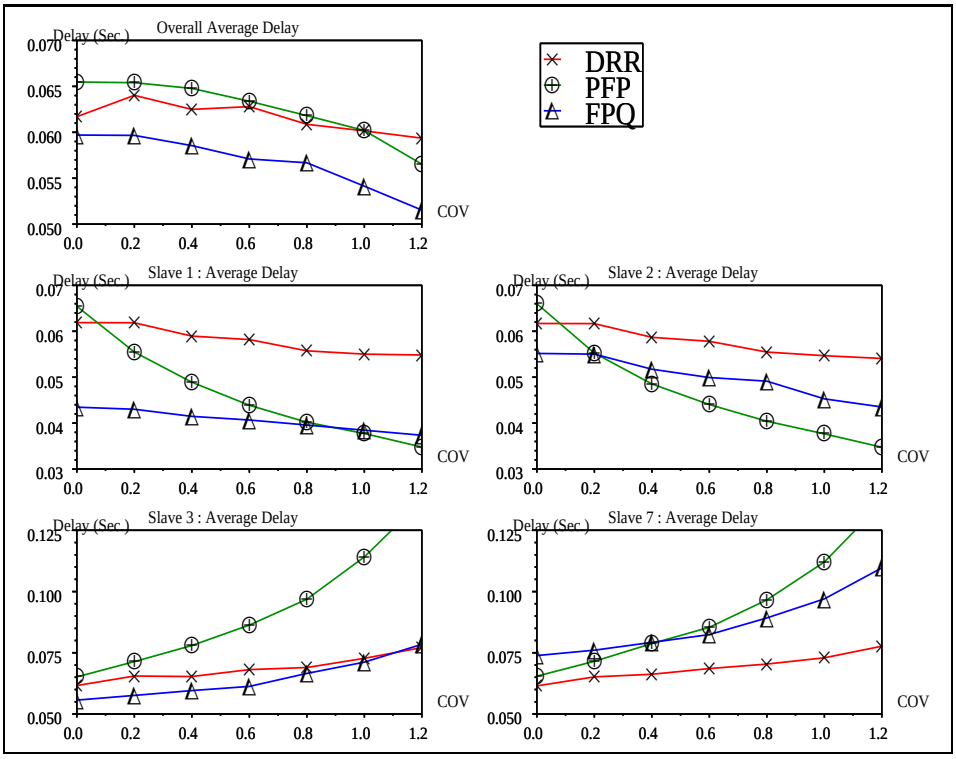

Figure 4.3: Average end-to-end delays versus COV

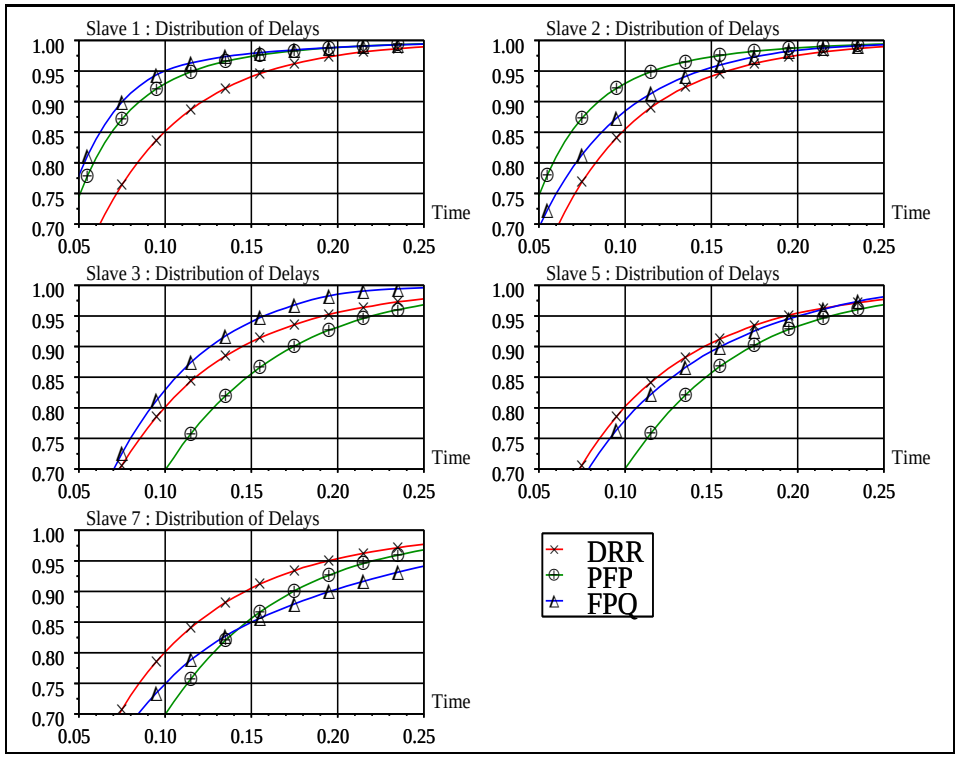

Figure 4.4: Distributions of delays per slave versus time

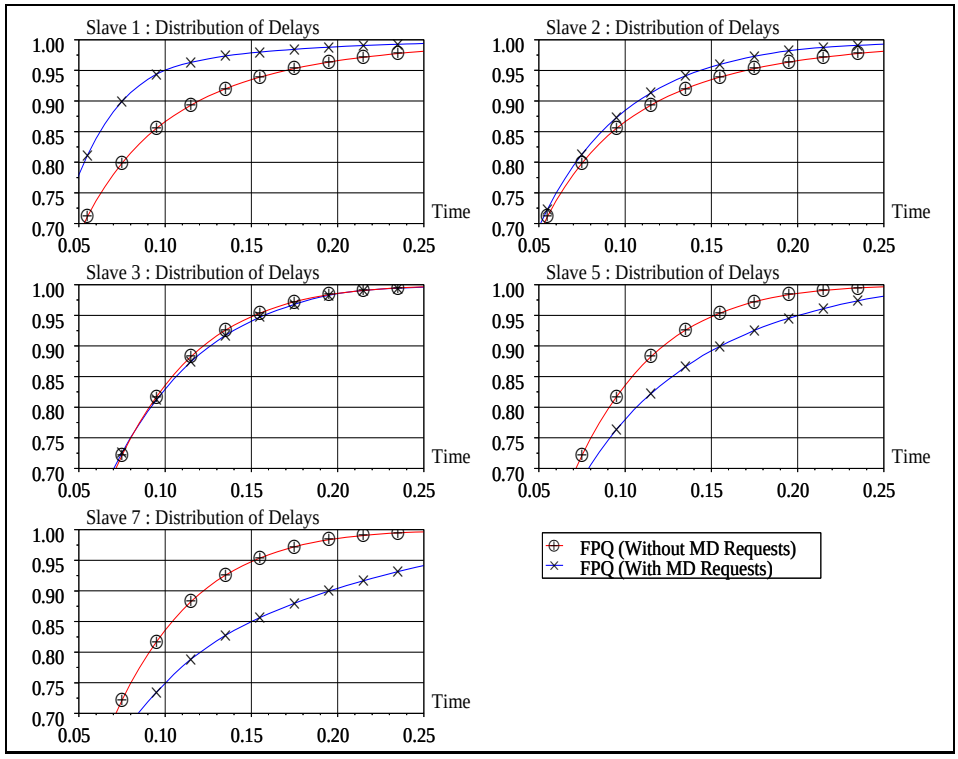

Figure 4.5: Distributions of delays per slave versus time - Effects of the MD Requests on the behavior of FPQ

4.2.4 Analysis of Jitter

We now analyze the jitter of the end-to-end delays, as it is an important parameter to analyze for real-time transmissions. We show both the average jitter per slave (Figure 4.6) and the variance of jitter (Figure 4.7). The average jitter allows us to estimate the importance of differences between consecutive delays. On the other hand, the variance of jitter indicates how the different values are distributed. Note that low variance corresponds to low scattered values.

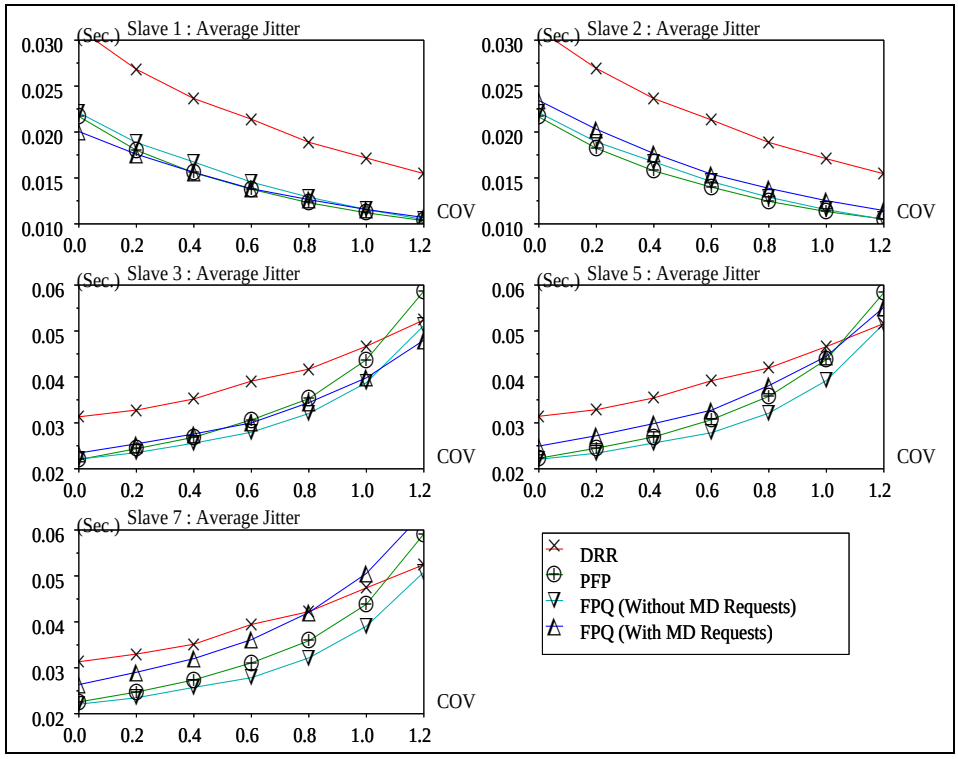

Figure 4.6: Average Jitter per slave versus COV

We observe that FPQ performs well and outperforms DRR. We present the results of FPQ handling the MD requests $\big($FPQ (With MD requests)$\big)$ and FPQ without handling any MD requests $\big($FPQ (Without MD requests)$\big)$ to evaluate the impact of the MD requests on the jitter. Without MD requests, we can say that FPQ performs at best. Yet, the MD requests introduce significant changes in the behavior of FPQ. Principally, slaves with high $MD_{s,i}$,

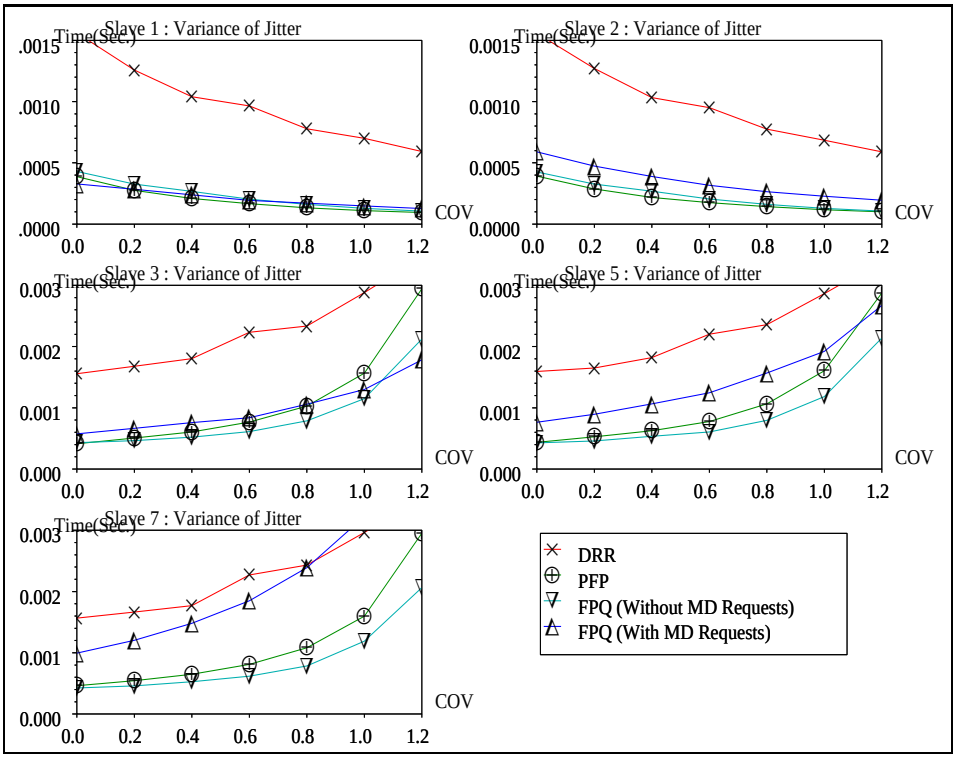

Figure 4.7: Variance of Jitter per slave versus COV

such as slave 7, are disadvantaged: they often loose priority in order to satisfy the MD requests of other slaves. Consequently, slave 7 obtains higher differences between consecutive delays. This could explain the evolution observed between FPQ without MD requests and FPQ with MD requests.

4.2.5 Effects of the parameter α on the behavior of FPQ

Finally, we present the effects of the parameter α on FPQ through the evolution of the average delay (Figure 4.8) and the distribution of delays per slave with $COV = 0.6$ (Figure 4.9).

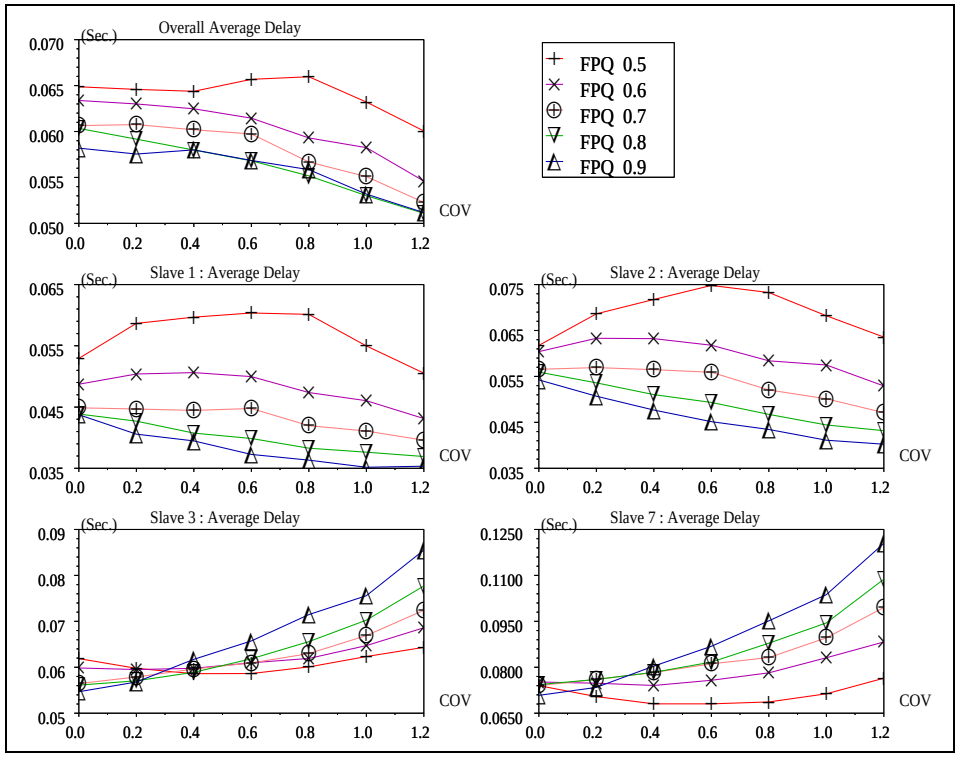

Figure 4.8: Average delay per slave versus COV

Both figures confirm the foreseen impact of α; high values give priority to efficiency and satisfaction of QoS requests, whereas low values give priority to fairness. Consequently, it is possible to adapt the α parameter according to traffic conditions and QoS requests. For

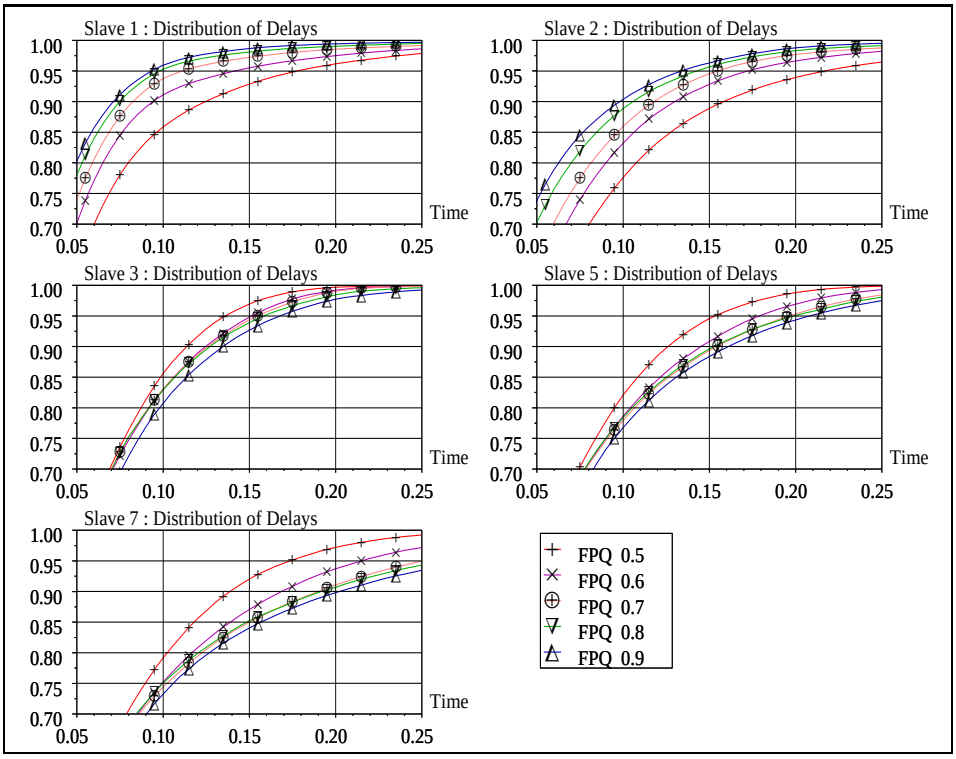

Figure 4.9: Distribution of delays per slave versus time

high flow rates and low $MD_{s,i}$, we can choose a high value of α. On the other hand, for low flow rates and high $MD_{s,i}$, it is better to select a lower value of α, that provides QoS support and more fairness in the Piconet. When looking at Figure 4.8, the asymmetry of QoS requests must be taken into account as well.

FPQ has been specially designed for supporting upstream traffic with QoS requests. These results above show the efficiency of our polling algorithm and its ability to adapt itself to various traffic conditions. Moreover, the simulations show that, with only upstream traffic, FPQ successfully provides QoS support for Bluetooth Piconet.

Chapter 5

Simulations with the Downstream Traffi c Model (DTM)

5.1 Description of the DTM

The DTM model is used to study performance of FPQ with only downstream traffic and using some FTP traffic [3].

The results which will be presented in the next section can be compared to the results of the AFP and StickyAFP algorithms [3], which were designed especially for downstream traffic. In table 5.1, we describe the protocols, applications, flow rates, start and stop times of applications used for traffic towards each slave.

Table 5.1: QoS Requests summary

Slave	S_1	S_2	S_3	S_4	S_5	S_6	S_7
Protocol	TCP	TCP	UDP	UDP	UDP	UDP	UDP
Applications	FTP	FTP	CBR	CBR	CBR	CBR	CBR
Rates (kbps)	—	—	17.6	35.3	17.6	5.5	5.5
Start(sec.)	0	10	5	20	0	0	10
Stop (sec.)	60	20	25	50	60	40	60

TCP Reno is used in the simulations, since it is one of the most common reference implementation for TCP.

5.2 Results with DTM

We now study FPQ with only downstream traffic and some FTP traffic. In this model of simulation, there is only downstream traffic, with two FTP sources. As in subsection 4.2, we present the results of FPQ with the parameter $\alpha = 0.8$. In order to obtain significant values, independently of punctual events, the throughputs shown on Figure 5.1 are averaged every 0.5 seconds for Slaves 1 and 2, and every 2 seconds for the other slaves. The most important point is the evolution of TCP throughput of Slave1; compared to the results obtained in [3], we obtain about the same throughput for Slave 1. Furthermore, we may notice the perfect share of bandwidth between the two FTP streams, represented by slaves 1 and 2, when they are both active. On the other hand, CBR traffic is completely handled by FPQ.

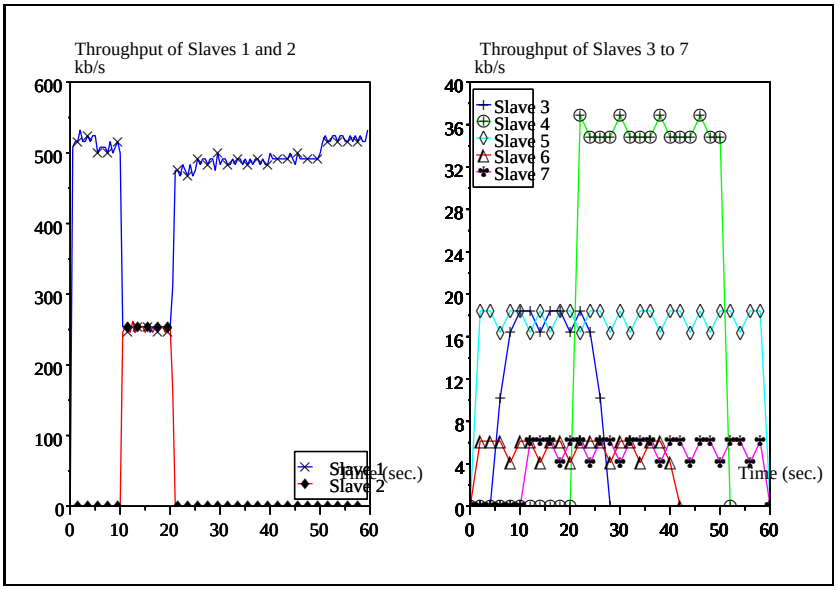

Figure 5.1: Throughput per slave versus time

The average delays shown on Figure 5.2 reveal higher delays for FTP traffic, compared to CBR traffic. These high delays are certainly caused by the fact that FTP traffic generates some “bursts” of IP packets, which consequently increases queuing delays. Yet, the most important point for this type of traffic is not delays, but the overall throughput. On the contrary, CBR traffic with low flow rates obtains very low delays compared to [3]. Although

FPQ was originally designed for upstream traffic, our results show that FPQ is suitable for handling downstream traffic and FTP traffic as well.

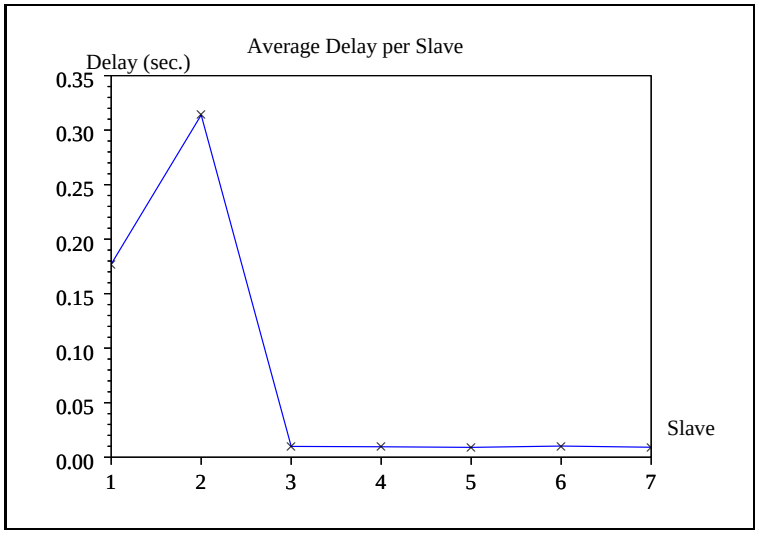

Figure 5.2: Average delays versus slave number

Chapter 6

Simulations with Mixed Traffic Model (MTM)

6.1 Description of the MTM

DRR has been used for comparison in this model of traffic. Arrival times of IP packets from all applications follow a Poisson process. All QoS requests are summarized in Table 6.1.

Table 6.1: QoS Requests Summary

Slave	S_1	S_2	S_3	S_4	S_5	S_6	S_7
Upstream : FR	$3.\lambda_u$	$3.\lambda_u$	λ_u	λ_u	λ_u	λ_u	λ_u
Upstream : MD(ms)	100	200	150	150	200	200	500
Downstream : FR	0	$2.\lambda_d$	λ_d	0	$2.\lambda_d$	$4.\lambda_d$	0
Downstream : MD(ms)	—	200	200	—	100	150	—

Each application generates IP packets with the same distribution of packet size than in the UTM. Each second, about 140 packets are sent by the applications to Bluetooth units:

$$\sum_{Active\ Slaves} \lambda_{u,i} + \lambda_{d,i} = 11.\lambda_u + 9.\lambda_d = 140$$

This implies that 73% of time slots should be used only for data transmission, without taking into account POLL or NULL packets. In the worst case, these QoS requests use 95% of all time slots. Thus, the Piconet is highly loaded.

We introduce the parameter μ that represents the distribution of flows among upstream and downstream traffic.

$$\mu = \frac{\sum_{Active\ Slaves} \lambda_{d,i}}{140} \tag{6.1}$$

With $\mu = 0$, there is only upstream traffic, with $\mu = 0.5$, upstream and downstream traffic are equal, and $\mu = 1$ implies downstream traffic only.

6.2 Results with MTM

We now study performance of FPQ with both upstream and downstream traffic, and analyze the influence of streams on each other. As in the previous cases, we have decided to present the results with $\alpha = 0.8$ for FPQ. As for upstream traffic, the simulations are made for a duration of 1200 seconds. Figures 6.1 and 6.2 correspond to simulations without any MD request.

We recall that when $\mu = 0$, there is only upstream traffic, when $\mu = 0.5$ upstream and downstream traffic are equal and when $\mu = 1$ there is only downstream traffic.

Figure 6.1 represents the overall delays for upstream and downstream traffic. We notice that FPQ performs well for both streams and decreases delays from 10 to 30% for upstream traffic and far more than 50% for downstream traffic compared to DRR. With FPQ, average delays for downstream traffic (upstream traffic) decrease when downstream traffic (upstream traffic) decreases. Moreover, in comparison with previous simulations, upstream traffic seems not to be penalized by downstream traffic. However, this traffic is better handled than upstream traffic, since we have further information about the $Q_{m,i}$.

Figure 6.2 shows the distribution of delays for each flow, with μ set to 0.5 and UT standing for upstream traffic and DT for downstream traffic. Basically, on this Figure, we compare slaves with same flow rates for upstream or downstream traffic, but with different flow rates in reverse traffic. With this scenario, we can observe the effects of the streams on each other. As expected, flow rates of reverse traffic interacts with the delays of each stream: for example, if we have a significant flow rate towards Slave 2, this means that Slave 2 will often receive data, and thus, it will be often polled and will obtain low delays for its upstream traffic. In spite of these differences, the main points are the overall efficiency of FPQ and the fact that no slave seems to be penalized.

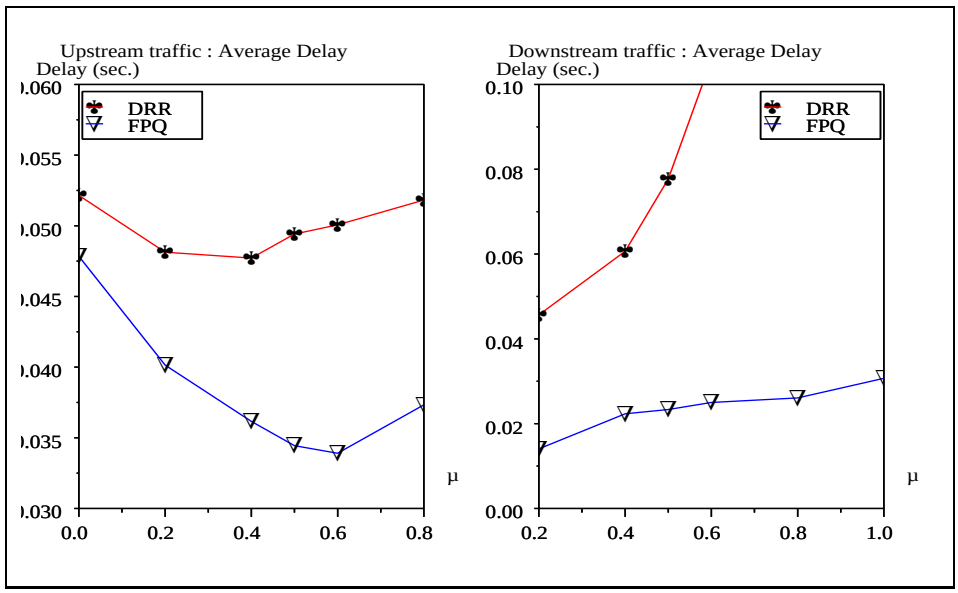

Figure 6.1: Average delays versus distribution of upstream/downstream flows (μ)

Figure 6.3 shows the distribution of delays for each flow, with $\mu = 0.5$ and using MD requests. The following results can be compared to those obtained on Figure 6.2. We can notice the efforts made by the poller to fulfill the new requests for both streams. Nevertheless, as we use Poisson process, we can not statistically assure fulfillment of all the QoS requests. However, these results show that FPQ does take into account the MD requests and respects these requests in more than 98% of cases. The changes in the different distributions are particularly noticeable in proximity of the critical values $MD_{m,i}$ and $MD_{s,i}$. As in the upstream traffic model, slaves with low MD requests now obtain a better distribution, without penalizing too much the other slaves.

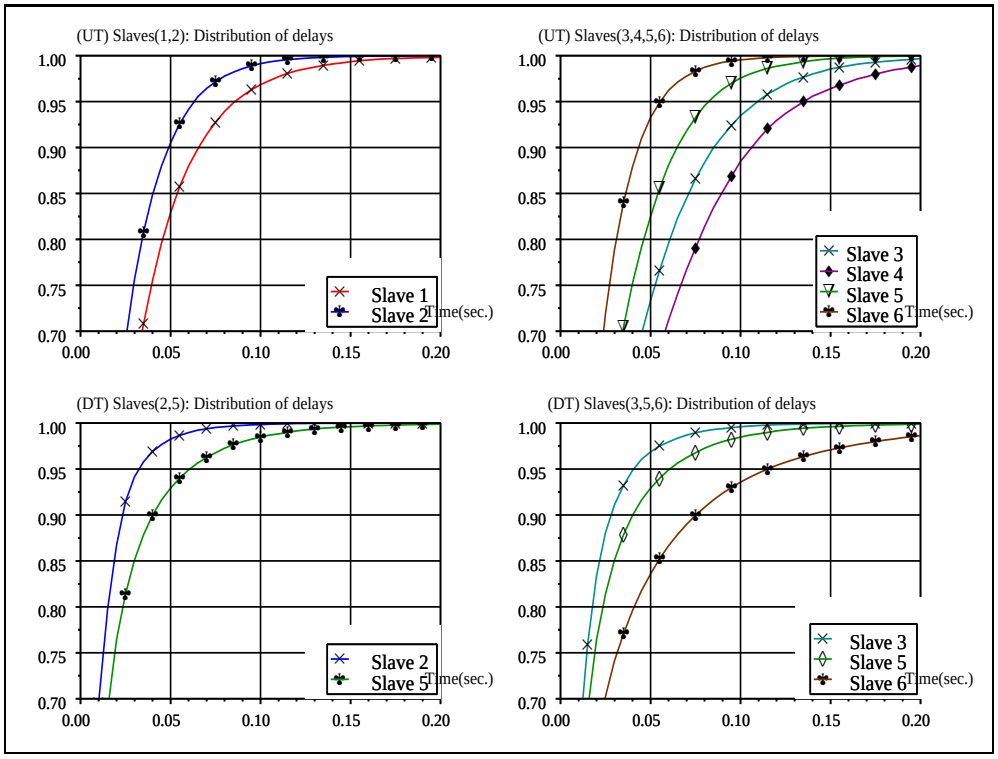

Figure 6.2: Distribution of delays versus time without MD requests

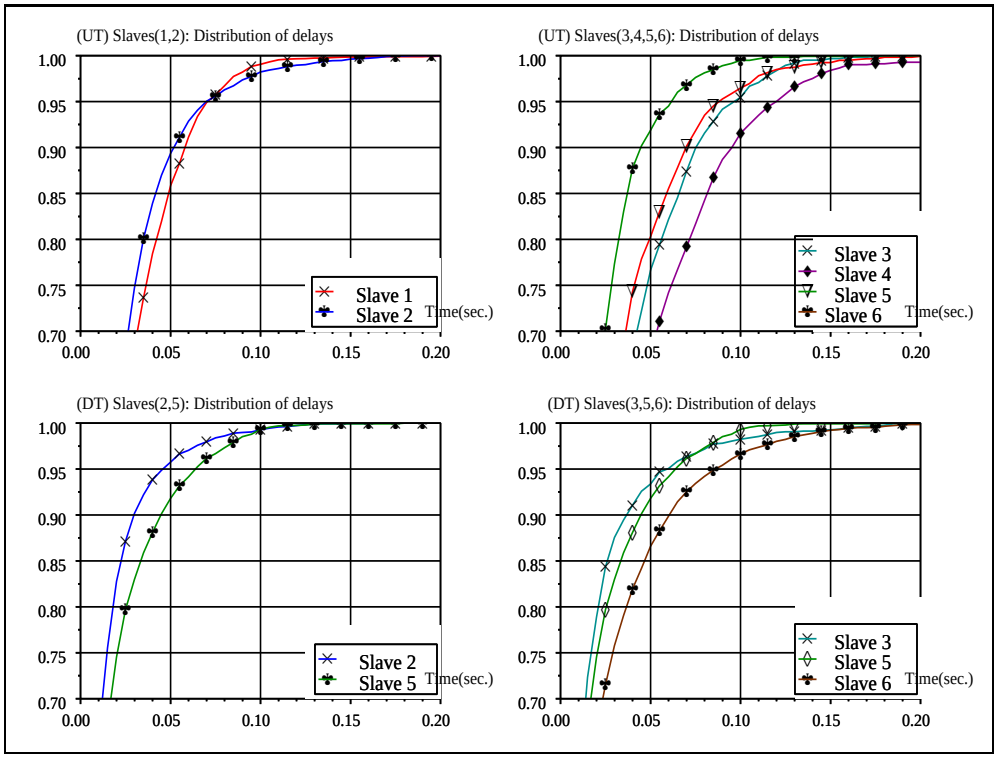

Figure 6.3: Distribution of delays versus time with MD requests

Chapter 7

Conclusions

We have designed and evaluated a new MAC scheduling for Bluetooth Piconet, with the aim of supporting asymmetric flow rates and QoS requests. The various simulations we have made prove the good performance of FPQ, compared to other schemes in terms of throughput, delay, fairness, and ability to support both a Flow Rate request and a Maximum Delay request. With the exception of a few changes in the HCI for QoS negotiations, FPQ does not imply any changes in other layers and is not too complex, which is required by Bluetooth.

In our work, we have focused on the ACL traffic. However, FPQ is also able to support SCO links, with reserved time slots: the support of SCO links could be provided by the Segmentation and Reassembly scheme proposed in [3]. Basically, the scheme takes into account the number of SCO connections, and thus, limits the size of Baseband packets, in order that ACL traffic fits into the time slots reserved for SCO link.

Future works seek to design an algorithm in order to adapt the value of the parameter α of FPQ, with respect to QoS requests and traffic conditions.

Bibliography

[1] R. Ait Yaiz and G. Heijenk, "Polling in Bluetooth - a Simplified Best Effort Case", *in proceedings CTIT Workshop on Mobile Communications*, ISBN 90-3651-546-7, February 2001.

[2] R. Bruno, M. Conti and E. Gregori, "Bluetooth: Architecture, Protocols and Scheduling Algorithms ", *Cluster Computing*, Vol. 5, Number 2, pp 117-131 , April 2002.

[3] A. Das, A. Ghose, A. Razdan, H. Saran and R. Shorey, "Enhancing Performance of Asynchronous Data Traffic over Bluetooth Wireless Ad-hoc Network" *in the proceedings of IEEE INFOCOM '2001*, Alaska, USA, April 2001.

[4] R. Epsilon, J. Ke and C. Williamson, "Analysis of ISP IP/ATM Network Traffic Measurements", *in Performance Evolution Review*, vol. 27,no. 2, pp 15-24, 1999.

[5] G. Heijenk and R. Ait Yaiz, "Predictive Fair Polling", Provisionnal Application for United States Letters Patent 60/241,314 , filed October 18, 2000.

[6] M. Shreedhar and G. Varghese, "Efficient fair queuing using deficit round robin", *IEEE/ACM Trans. on Networking*, Vol. 4, No 3, pp. 375-385, 1996.

[7] M. Van der Zee, G. Heijenk, "Quality of Service in Bluetooth Networking" *Technical Report University of Twente*, TR-CTIT-01-01 , January 2001.

[8] Bluetooth Special Interest Group, "Specifications of the Bluetooth System 1.1", *http://www.bluetooth.com* 2001.

[9] The Bluehoc Web site *"http://www-124.ibm.com/developperworks/opensource/bluehoc"*.

[10] The Network Simulator (ns2). Software and documentation available from *"http://www.isi.edu/nsnam/ns"*.

[11] FPQ implementation for Bluehoc and simulations available from *"http://www-sop.inria.fr/planete/turletti/bluetooth/fpq.tar.gz"*.

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifi que
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399