Wireless Access to Internet via Bluetooth: Performance Evaluation of the EDC Scheduling Algorithm

Raffaele Bruno, Marco Conti, Enrico Gregori
Consiglio Nazionale delle Ricerche
Istituto CNUCE
Via G. Moruzzi, 1, 56124 Pisa – Italy
Tel: (0039) 050-3153063, Fax: (0039) 050-3138091
e-mail:{Marco.Conti,Enrico.Gregori}@cnuce.cnr.it
Raffaele.Bruno@guest.cnuce.cnr.it

Abstract
Bluetooth is an emerging technology for constructing ad-hoc wireless Personal Area Networks (WPANs). In this paper we analyze an innovative scheduling algorithm for asynchronous data traffic specifically tailored to the Bluetooth characteristic. This algorithm, named Efficient Double-Cycle (EDC), dynamically adapts the polling frequency to the traffic conditions. By considering a scenario where a Bluetooth master is used as wireless access point to the Internet, we show that our EDC scheduler significantly enhances the system performance with regard to a Round Robin (RR) scheduler.

Keywords
Bluetooth, TCP, Scheduling, Medium Access Control (MAC), Polling, Automatic Repeat Request (ARQ).

1. Introduction
The technologies for WPANs, as the emerging Bluetooth technology, offer a wide space for innovative solutions and applications that could bring to a radical change in everyday life. In particular, the Bluetooth technology is much more than a wireless connection for a nomadic access to the Internet or for cable replacement, but it wants to be an enabling technology for the global mobility of devices and services [2], [4].

Before Bluetooth applications can be deployed, it is necessary a considerable effort from the research community to resolve the technical issues specific to this technology. In particular, the strong need for low-cost, low-power and low-complexity devices has lead the Bluetooth standardization forum to adopt a centralized Time Division Duplex (TDD) access scheme as the MAC protocol for the channel access. In a Bluetooth network, named as *piconet*, one station has the role of master and all other Bluetooth stations are slaves. Up to seven slaves can participate to the piconet communications. The master decides which slave is the one that can access the channel. More precisely, a slave is authorized to deliver a single packet to the master only if it has received a polling message from the master.

Hence the scheduling algorithm is a key component in a Bluetooth network, the standard documents initially proposes a Round Robin (*RR*) scheduler [1]. Previous works [6], [4] have shown that the RR algorithm may introduce bandwidth wastage. Therefore, efficient MAC scheduling algorithms need to be designed. The original contribution of this paper is the performance evaluation of an efficient scheduling algorithm named Efficient Double-Cycle (*EDC*), considering Internet traffic. Our algorithm tunes the polling order to the network traffic conditions to limit the channel-bandwidth wastage caused by the polling of empty stations.

Johansson et al. have proposed a Fair Exhaustive Polling (*FEP*) [6]. Similarly to our algorithm, FEP tries to avoid the polling of inactive stations. Specifically, they define the meaning of active and inactive state for the slaves, and introduce polling sub cycles where only the active slaves are polled in a round robin fashion. Our algorithm further extends this idea i) by separating the scheduling of the uplink and downlink transmissions, and ii) by adopting a truncated binary exponential-backoff algorithm to determine the amount of time an inactive slave is removed from the polling cycle.

The design of a scheduler suitable for the Bluetooth MAC layer was also considered by Shorey et al. They have proposed several MAC scheduling algorithms [8], [9] where they distinguish the master-slave pairs based on the state of the queues at the master and slaves. They assume that the information regarding the status of the queue at the slave is available in the master because it comes directly from the slaves, which appropriately set some bits in the MAC header. Once that different classes/priorities are assigned to each master-slave pair, various service policies can be devised. Our scheduling algorithm departs from this approach because it does not rely upon any information coming from slaves.

In [10] we have exhaustively studied the EDC behavior from the MAC layer standpoint. Specifically, we have investigated several performance figures, as aggregate link utilization, MAC delays and power indexes, and we have

This work was carried out in the framework of the CNUCE-CNR GMD-FOKUS bilateral project "SIMTO – A Framework for a Flexible SIMulator Toolset for Future IP Networks ".

found that EDC always outperforms RR. This analysis was a theoretical one, as we do not consider the real traffic and the impact of high-layer protocols on the traffic spacing. Since one of the most interesting Bluetooth applications is expected to be the ubiquitous wireless access to Internet, in this work we extend the previous analysis to a realistic case in which the impact of TCP flow-control mechanisms has been investigated.

The performances of TCP over Bluetooth have been investigated by others. A preliminary analysis of TCP performance over Bluetooth is presented in [7], but this analysis is only related to the TCP Vegas that is rarely adopted in current TCP implementation. In [9] a more accurate analysis of TCP performances has been executed, but no insight is given on the impact of TCP parameters over the performances of a connection delivered via a Bluetooth link.

The results presented in the following indicate that EDC provides a significantly throughput improvement for the TCP connections, when compared with the RR scheduler. Particular attention has been devoted to the study of the role of the TCP Maximum Segment Size (MSS) over the throughput performance. We also investigate the impact of channel errors over the TCP and the effectiveness of the ARQ scheme adopted in Bluetooth for error recovery.

The paper is organized as follows. In Section 2 we briefly describes the Bluetooth MAC layer, with a particular attention to the channel access scheme. In Section 3 we give the specification of our scheduling algorithm. Then, Section 4 presents a complete performance evaluation of the scheduling algorithm. Concluding remarks are summarized in Section 5.

2. Overview of the Bluetooth MAC Protocol

From a logical standpoint, Bluetooth belongs to the contention-free token-based multi-access networks [4]. A Time Division Duplex (TDD) scheme of transmission is adopted. The channel is divided into time slots, each 625 µs in length. The time slots are numbered according to the Bluetooth clock of the master. The master can begin a new transmission in even numbered time slots. Odd numbered time slots are reserved for the beginning of slaves' transmissions. As stated before, the channel access is managed according to a polling scheme. The master decides which slave is the only one to have the access to the channel by sending to him a packet. The master packet may contain data or can simply be a polling packet. When the slave receives a packet from the master it is obliged to transmit in the next time slot, to acknowledge the master transmission. This implies that the scheduling decisions are always related to a couple of stations: the master and the slave that receives the master transmission. The slave packet can contain data or can simply be a NULL packet.

There are two types of physical links that can be established between Bluetooth devices: a Synchronous Connection-Oriented (*SCO*) link, and an Asynchronous ConnectionLess (*ACL*) link. The first type of physical link is a point-to-point, symmetric connection between the master and a specific slave. It is used to deliver delay-sensitive traffic, mainly voice. The SCO link rate is 64 kbps and it is settled by reserving a couple of consecutive slots for master-to-slave transmission and immediate slave-to-master response. For SCO links the master periodically polls the corresponding slave, instead the polling is asynchronous for ACL links.

The SCO link can be considered as a circuit-switched connection between the master and the slave. The second kind of physical link, the ACL link, is a connection between the master and all slaves participating to the piconet, and it can be considered as a packet-switched connection between the Bluetooth devices that supports point-to-multipoint transmissions from the master to the slaves. The ACL channel guarantees the reliable delivery of data: a fast Automatic Repeat Request (*ARQ*) scheme is adopted to assure data integrity.

The MAC layer also accomplishes the Segmentation and Reassembly (*SAR*) procedure to improve the protocol efficiency by supporting a maximum transmission unit size (*MTU*) larger that the ACL packet sizes.

Table 1. ACL packets' maximum payload sizes (bytes)

DH_1	DH_3	DH_5	DM_1	DM_3	DM_5
27	183	339	17	121	224

It is worth pointing out that a tight constraint of Bluetooth technology is that ACL packets can covers 1, 3 or 5 time slots. According to Bluetooth specification [1], the ACL packets are of two different groups, one denoted DMx and the other one denoted DHx. The former has a payload encoded with a rate 2/3 FEC and the latter hasn't got a FEC encoding. The subscript x stands for the number of slots that are necessary to transmit the packet. Table 1 reports the different payload size of ACL packets.

More details about the Bluetooth Mac protocol can be found in [1], [3].

3. Efficient Double Cycle Scheduling Algorithm for a Bluetooth Piconet

The Efficient Double-Cycle (*EDC*) algorithm is based upon two main ideas: first of all it is necessary to avoid NULL transmissions towards and from the slaves; furthermore the fairness typical of a Round Robin scheme should be preserved. These targets can be accomplished if the selection of the slave to be polled takes into consideration the master knowledge of traffic from and to the slaves. In the rest of our discussion, we assume a system model where the master employs a separate queue for each slaves participating to piconet communications. It is worth pointing out that the master has a deterministic knowledge of the state of its local queues, i.e., the queues containing the packets for the slaves. On the other direction (i.e., traffic from the slaves), we only assume that the master has a probabilistic knowledge based on the feedback the master gets when polling the slaves. Therefore the master can only estimate the probability that a slave will send a NULL packet by exploiting the knowledge of the slave behavior in the previous polling cycles. As stated in the introduction, our goal is to define an efficient scheduling algorithm that is not implementation dependent, namely, it does not require that all slaves are able to set specific header fields according to the particular scheduling algorithm adopted by the master. Hereafter, we indicate as *uplink* the link direction from slaves to master, and as *downlink* the link direction from master towards slaves.

An additional problem to have a fair and efficient scheduling in Bluetooth is caused to the coupling between the transmission in uplink and downlink (i.e. a master to slave transmission implies also a polling of the slave and hence a, possibly NULL, slave to master transmission). Therefore, it is not possible to remove a slave from the polling cycle without blocking at the same time the master transmissions towards this slave. To introduce a (partial) decoupling in the scheduling of the transmissions in uplink and downlink we introduce the idea of a double polling cycle: an *uplink polling sub cycle*, hereafter called $Cycle_{UP}$, and a *downlink polling sub cycle*, hereafter called $Cycle_{DW}$. During both of these cycles the master selects a subset of slaves that are *eligible* for the polling. For the sake of brevity we will refer to the subset of eligible slaves selected during a $Cycle_{UP}$ as *E(UP)*, and to the subset of eligible slaves selected during a $Cycle_{DW}$ as *E(DW)*. *E(DW)* is calculated considering only the state of master local queues, i.e. the ongoing traffic loads, whereas *E(UP)* is calculated considering only the estimated slaves' activity, i.e. the ingoing traffic loads. The distinction between polling rules for the downlink and the uplink permits to have a "*fairness separation*": in the *downlink* (*uplink*) *sub cycle* fairness is guaranteed only in the downlink (uplink) direction.

3.1 EDC specification

A detailed EDC specification through pseudo-code can be found in [10]. Due to the space constraints we provide only an EDC specification through the natural language.

The outcome of our polling algorithm is the next slave to be polled, also referred as *next.* Because there is a double cycle, we have a *nextUp* calculated during the $Cycle_{UP}$, and a *nextDw* calculated during the $Cycle_{DW}$. A polling interval c_i and a polling window w_i are associated to each slave S_i. These variables are used to estimate the slave activity, and to implement a *truncated binary exponential backoff* mechanism to control the slaves polling frequency during a $Cycle_{UP}$.. Specifically, at the end of each S_k's packet reception, the master updates the polling interval c_k and the polling window w_k with the following rules:

1. If S_k sends a packet with a null payload, then its polling window w_k is doubled till a maximum value w_{MAX} is reached.
2. If S_k sends a packet with a not null payload, then w_k is settled to be 1.
3. The polling interval c_k takes the w_k value.

During a $Cycle_{DW}$ all the S_i that belongs to *E(DW)* are polled in a cyclic order (i.e. we adopt a Round Robin policy for the *E(DW)* set). After a slave has been polled it is extracted from *E(DW)*. Therefore a $Cycle_{DW}$ is completed only when *E(DW)* becomes empty. At the same way, during a $Cycle_{UP}$ all the S_i that belongs to *E(UP)* are polled in a cyclic order (i.e., we adopt a Round Robin policy for the *E(UP)* set). After a slave has been polled it is extracted from *E(UP)*. Therefore a $Cycle_{UP}$ is completed only when *E(UP)* becomes empty. We consider a polling cycle as completed if all the slaves that belong to both *E(UP)* and *E(DW)* have been polled.

One of the most important tasks that is accomplished by the master at the beginning of each polling sub cycle is the update of *E(UP)* or *E(DW)* according to its knowledge about traffic loads. Specifically, *E(UP)* is constituted by all slaves S_i that have c_i null at the beginning of a $Cycle_{UP}$. Because *E(UP)* is updated before the beginning of each $Cycle_{UP}$, if S_i has a c_i equal to *n*, then *n* $Cycle_{UP}$ must elapse before S_i is polled during a $Cycle_{UP}$. On the other hand, *E(DW)* is constituted by all slaves S_i to which the master has traffic to send at the beginning of a $Cycle_{DW}$, i.e., slaves corresponding to the not-empty master local queues.

At the beginning of each new master transmission, according to the piconet timing, the master executes the following actions:

1. If the polling cycle is finished then EDC updates *E(DW)* and a new polling cycle begins with a $Cycle_{DW}$.
2. If the polling cycle is not finished, EDC determines if the next transmission belongs to a $Cycle_{DW}$, i.e., *E(DW)* is not an empty set, otherwise EDC decreases the polling interval variables, it updates *E(UP)* and a new $Cycle_{UP}$ begins.

It is worth noting that if at the beginning of a new polling cycle *E(DW)* is an empty set then EDC decreases the polling interval variables and a new $Cycle_{UP}$ begins. This behavior corresponds to have a $Cycle_{DW}$ with null duration. If *E(UP)* and *E(DW)* are both empty sets EDC has no information to discriminate a slave from the others, then EDC applies a round robin polling rule till the master does not receive a data packet from a slave or at least one of its local queues is not empty.

In the following section we present a complete simulative analysis of the EDC algorithm performances when it is used to schedule Internet traffic. By exploiting this analysis we demonstrate the performance enhancement achieved by EDC, with respect to a round robin scheduling algorithm. Furthermore, we point out some specific issues related to the transport of TCP traffic over the Bluetooth link to give a better understanding of EDC behavior.

4. Simulation Model and Performance Evaluation

The network model simulated is a single piconet constituted by a master and up to seven slaves. A detailed description of the Bluetooth architecture and its protocol stack can be found in [1], [2]. For the sake of the following discussion we refer to Figure 1, where we report a simplified architecture of the system.

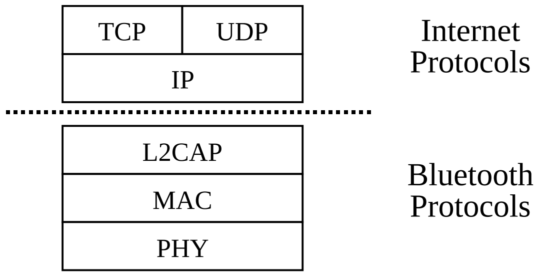

Figure 1. Protocol stack

The traffic sources generate TCP and UDP packets that are sent to the network layer, which adds its header (20 bytes). The L2CAP layer receives the data segments from the upper layers, it adds 4 bytes of L2CAP header and it queues the L2CAP packets in a buffer. The size of this buffer is chosen equal to 64 Kbytes that is the standard *advertised window* used in the TCP receiver. Since no more that an advertised window of TCP traffic can be generated by a TCP sender, this buffer size guarantees that no buffer overflow losses are experimented by TCP traffic[1] It means that each TCP source see an infinite size buffer. This assumption permits to study the dynamic of TCP connections under an EDC scheduler without biasing effects due to the limited buffer size.

Large L2CAP packets must be segmented into smaller Baseband packets before their transmission can take place. A new L2CAP packet cannot be served till all fragments (generated during the segmentation at the MAC layer) of the previous L2CAP packet have been successfully transmitted. The segmentation procedure is accomplished, just before the transmission, in such a way to maximize the amount of data conveyed by each baseband packet. In particular, the segmentation procedure executes the following steps:

1. Divide the L2CAP packet into an integer number of 5 slot baseband packets.
2. If the size *x* remaining to be fragmented is larger than the size of a 3 slot baseband packet, sent it as a 5 slot packet.

[1] Let us remind that a slave establishes a single TCP connections with the master

3. If x is larger than the size of a 1 slot baseband packet but shorter than the size of a 3 slot baseband packet, send it as a 3 slot baseband packet.
4. Otherwise, send x as a 1 slot baseband packet.

It would be possible to devise several SAR procedures (see [8], [9]), but this issue is left for future discussion.

Figure 2 shows the network configuration and traffic parameters, hereafter referred to as *scenario A*, used in the experiments presented in the next section. The TCP sources are asymptotic connections, i.e., they have always a data packet ready to be transmitted (ftp-like traffic). The TCP version considered is the TCP-Reno, the most worldwide adopted TCP implementation [5]. The UDP sources generate CBR traffic. In the figure the arrows indicate the direction of data packets in the connection slave-master. For each connection, the type of traffic (i.e., UDP or TCP), the time instant of beginning ad termination, and the bit rate (only for the UDP sources) are given.

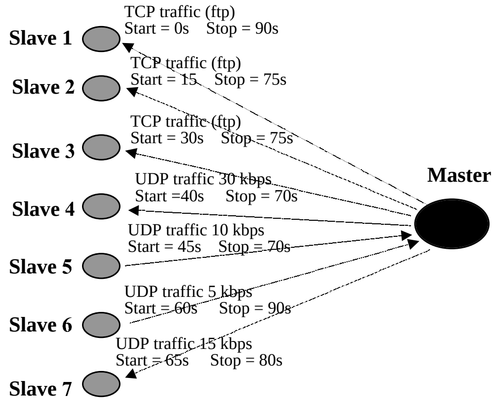

Figure 2. Traffic sources used in Scenario A

We consider sources with different time interval of activity to capture the dynamic behavior of the scheduling algorithm. We also locate some sources in the master and other sources in the slaves to evaluate the fairness of scheduling algorithm between the uplink and downlink direction. Finally, we analyze the ability of the scheduling algorithm to adapt to asymmetric loads by considering different CBR rates.

During all the simulations carried out we have assumed w_{MAX} equal to 32, and we have used ACL packets with no FEC encoding, i.e., only DH*x* packets (see Table 1).

4.1 Numerical comparison between EDC and RR behavior

In this section we consider and ideal channel with no errors, and we use a constant TCP packet size of 1024 bytes, a constant UDP packet size of 500 bytes and a TCP ACK size of 20 bytes.

Figure 3 shows the throughput for the TCP connection of Slave1 achieved in the scenario A with a master that adopts either the EDC algorithm or the Round Robin (RR) algorithm. We observe that EDC guarantees a throughput always significantly higher than the one achieved with a RR scheduler.

In the time interval [0, … 15sec] there is a single active TCP connection. During this time interval the throughput obtained with EDC is more than two times the one obtained with RR, due to the capacity of EDC to adapt the polling frequency to the sources' activity. The ripples observed after 40 seconds are due to the starting of the UDP sources. The same results have been obtained for the other TCP connections and they are not reported here.

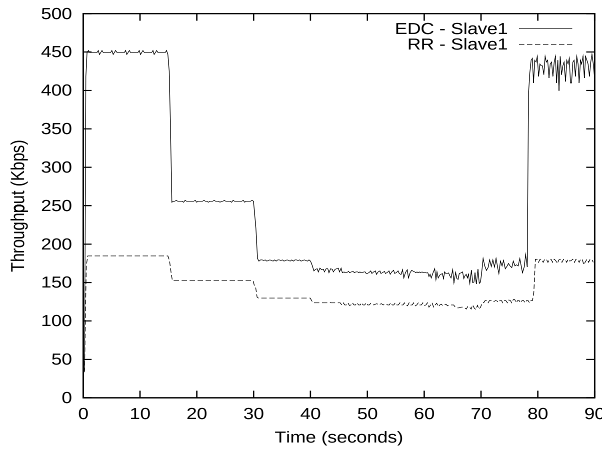

Figure 3. TCP throughput of Slave1 connection

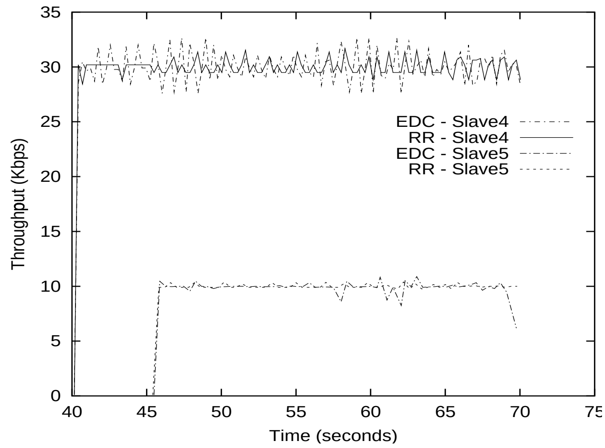

Figure 4. UDP throughput of Slave4 and Slave5

Figure 4 shows the throughput of two CBR sources, the former located in the master (curve labeled as Slave4) and the latter located in a slave (curve labeled as Slave5). It is worth noting that in the link slave4–master all the UDP packets are scheduled during the *downlink polling sub cycle*, whereas in the link slave5–master all the UDP packets are scheduled during the *uplink polling sub cycle*. However, the scheduler gives a fair treatment to these flows. Furthermore, we observe that EDC increases the throughput of TCP flows with respect to RR, but it also permits to guarantee a throughput for UDP flows equal to their rate, as RR.

Figure 5 shows the throughput of the three TCP connections when they are contemporarily active. We observe that EDC behaves fairly with the TCP flows, as a RR scheduler does. We have proved the fairness of EDC when all the TCP data packets are scheduled during $Cycle_{DW}$. In the following experiment we check if the different polling rules adopted during $Cycle_{DW}$ and $Cycle_{UP}$ can have an impact over the TCP performances.

Figure 6 shows the throughput of a TCP connection established between the slave2 and the master when the TCP sender is either located in the master (scenario A), or in the slave (scenario B). All the other sources behave exactly as in scenario A. This experiment shows that the TCP throughput slightly increases when the data packets flow is from the slave towards the master. This is due to the different frequency by which the scheduler polls slave2 during $Cycle_{UP}$ in the two cases.

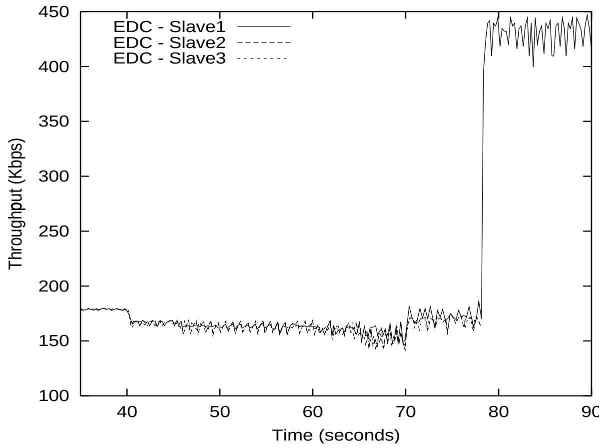

Figure 5. Fairness of EDC algorithm with respect to TCP connection

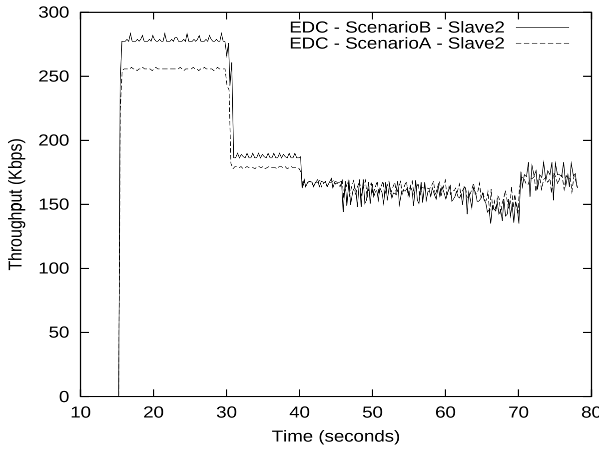

Figure 6. TCP throughput of Slave2 connection when reversing the connection direction

In scenario A, the slave2 queue contains the acknowledgement traffic. When the master sends a fragment of a TCP packet, it is highly probable that it receives a NULL packet from the slave2 (the ACK cannot be generated by the TCP receiver till the TCP packet is completely received), therefore the polling interval for slave2 increases, and the slave2 will skip some of the following *uplink polling sub cycles*. This behavior can reduce the rate by which the acknowledgement traffic is served, therefore also lowering the rate of TCP data (let us remind that the polling of one direction also means the polling of the other one). On the other hand, in scenario B the slave2 queue contains the data traffic, and it is highly probable to find a queued TCP packet (the TCP source is asymptotic), therefore no $Cycle_{UP}$ is skipped. Furthermore, as soon as the ACK for the slave2 is generated, the master will serve it in the first available $Cycle_{DW}$, with no additional delay. Obviously, this phenomenon is more evident when a few sources are active, because the polling cycle is short.

In conclusion, the results presented so far demonstrate that EDC significantly improves the throughput performance of TCP flows in a piconet, when compared to a RR scheduler. Furthermore, we have clarified that the decoupling of scheduler decisions related to the polling of the uplink and downlink can imply some issues when the traffic flows from a slave to the master and in the inverse direction are correlated (as for a TCP connection).

4.2 Effect of Maximum Segment Size over the TCP throughput

In this section we study in depth the role that the *Maximum Segment Size* (*MSS*) of a TCP connection has in determining the TCP throughput. In the following we assume that the TCP packet has a constant size equal to the MSS.

Figure 7 shows the throughput achieved from a TCP connection established from a master to a single slave versus the MSS. Obviously, when the packet size increase, the overhead due to the IP header and the L2CAP header has a decreasing weight. For example, with a MSS equal to 315, 654, 993 or 1332 bytes the delivery of a single TCP packet requires (including all upper layer overheads) exactly an integer number of DH_5 packets (1, 2, 3 or 4, respectively). Therefore the throughput increase is due to the decrease in the percentage of the overheads due to the upper layer. However, the dominant factor in determining the total throughput is the segmentation accomplished by the MAC layer. Specifically, as stated in Section 2, the segmentation mechanism has to use packets with discrete and fixed lengths.

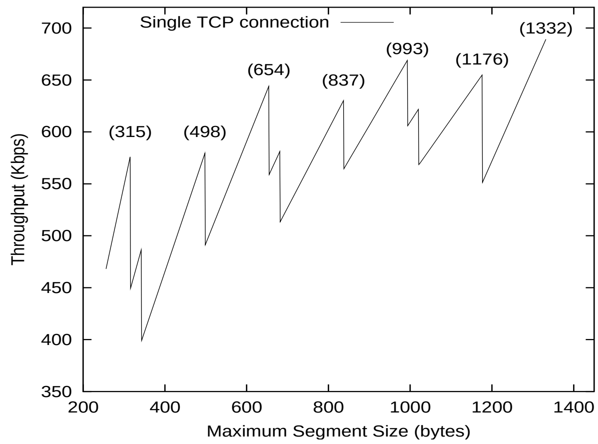

Figure 7. TCP throughput vs. MSS

For example, with a MSS=993 bytes, the delivery of a single TCP packet requires (including all upper layer overheads) exactly three DH_5 packets. If we increase the MSS to 994 bytes, then we need a further DH_1 packet, which conveys only a single byte of information. Therefore we measure a sharp decrease in the throughput, as reported in Figure 7. By exploiting similar considerations, we can easily understand the behavior of the TCP throughput curve shown in Figure 7.

These results presented so far, it follows that the selection of the MSS value for a TCP connection is a critical task in the Bluetooth technology. In the next experiment, we investigate the behavior of EDC and RR when the TCP connections adopt different MSSs. Specifically, we have considered a piconet with a master that establishes seven TCP connections towards as many slaves. Figure 8 shows the throughput obtained by each connection. The MSS adopted by each slave is reported over the related column. We have performed this experiment with both EDC and RR scheduler. We observe that EDC does not fairly assign the bandwidth to the flows. In particular the connection with the smallest MSS takes the highest throughput. This is due to the different ACK generation rate that characterizes the TCP connections.

Due to the delayed ACK mechanism, an ACK is generated at least after the reception of two TCP packets (after the reception of a single TCP packet if the time interval between two consecutive receptions is greater than 200 msec) [5]. The slave1 generates a number of ACK greater than the other connections, since each ACL packet sent by the master towards the slave1 delivers a new TCP packet. Therefore slave1's queue has more traffic to send than the other slaves' queues, and the EDC polls more frequently slave1 that the other slaves during a $Cycle_{UP}$. With RR all the slave receive the same polling frequency, therefore the throughput difference between the connections are at most related to the different percentage of the ACL packets occupied by the upper layer overheads (see Figure 7). It is worth pointing out that the aggregate throughput achieved with EDC is greater than the one achieved with RR, in particular 624 Kbytes against 611 Kbytes, even if all the TCP source are asymptotic.

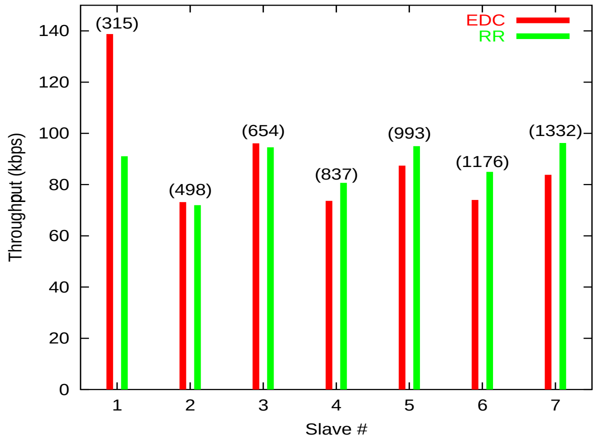

Figure 8. TCP throughput of connections with different MSSs

Figure 8 seems to suggest that RR behaves better than EDC for some MSSs, like MSS=1332. More precisely, from Figure 8 we can only derive that if a TCP connection adopts a short MSS can become more aggressive towards TCP connections with a long MSS, from a throughput standpoint.

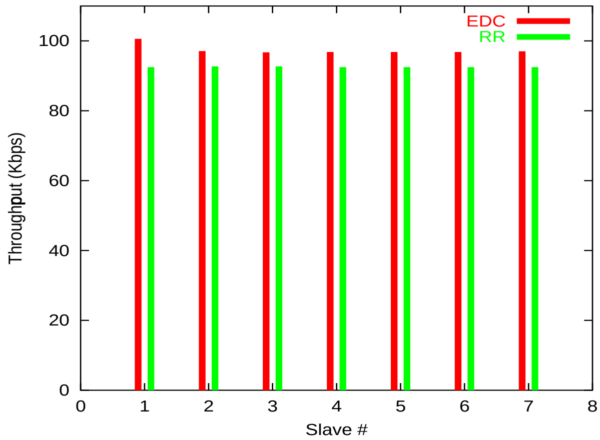

Figure 9. TCP throughput of connections with MSS=1332

Figure 9 shows the throughput obtained by each connection when all of them adopt a MSS=1332. We observe that EDC is fair as RR when all the TCP connection have the same MSS. Furthermore EDC guarantees a greater throughput than RR even if all connections are asymptotic. It is worth noting that the performance gain guaranteed by EDC increase with not-asymptotic TCP flows. Specifically, the EDC behavior is the same of RR behavior when all the queues are never empty. In the previous section, we have shown that EDC is able to adapt its polling frequency to the traffic characteristics when a source alternates periods of activity with period of inactivity. Hence, the analysis with asymptotic TCP flows is a worst-case analysis. The same results are obtained with the other MSSs.

In conclusion, we can assess that TCP connections with a small MSS are more aggressive than the ones with a large MSS when the EDC algorithm is adopted. Further studies are required to introduce a fair behavior of the scheduler also towards TCP connections with different MSSs.

4.3 Effect of the ARQ scheme adopted in Bluetooth over TCP behavior

In this section we evaluate the impact of channel errors on the TCP behavior. In particular, we are interested in investigating the effectiveness of the unnumbered ARQ scheme[2] adopted by the Bluetooth MAC layer to hide the channel errors to the transport layer.

We model the wireless channel as a discrete two-state Markov Chain [11]. In each state the bit error rate (BER) is constant, but in one state, i.e. the *Bad state*, the BER is significantly higher than in the other one, i.e. the *Good state*. Hereafter, we will consider the BER in Good state equal to 2x10E-6, and the BER in Bad state equal to 10E-4. We assume that, in the Markov chain model, the average sojourn time in each state is expressed as a multiple T of the time slot. Furthermore, we assume that the BER remains constant during the packet transmission. We have also considered a different model for the wireless channel with uniformly distributed errors. In this case the BER has been chosen equal to the average bit error rate experimented in the two-state model. This model is used to investigate the impact of error recovery mechanism over the TCP both with burst of errors and uniformly distributed errors.

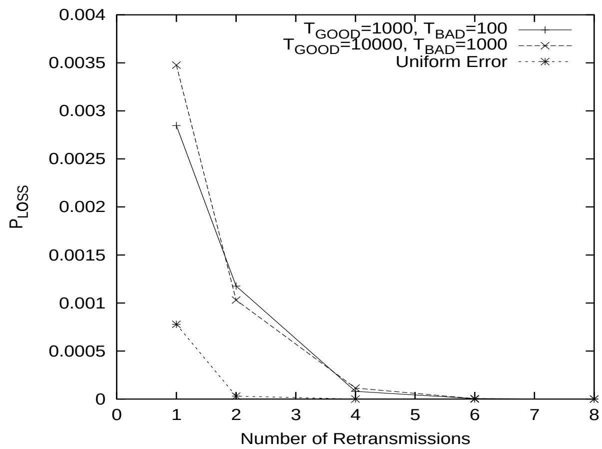

Figure 10. TCP packet loss probability with MSS=1332 bytes

It is worth pointing out that the main objective of this section is to illustrate the behavior of the ARQ scheme in presence of bursty losses on the wireless channel, hence an approximate characterization of the wireless channel is sufficient. To the best of our knowledge, an exhaustive characterization of the piconet wireless environment based on experimental measures is not yet available, and the numerical values chosen in this paper are compliant with the one adopted by other researchers [9], [12].

We have studied the impact of the maximum number of consecutive retransmissions for the same fragment, say it

[2] A Stop and Wait like error-recovery mechanism [1]

ritr_thresh, on the TCP packet-loss probability. It is worth pointing out that when a fragment is discarded because the *ritr_thresh* is exceeded, the MAC layer discards all the subsequent fragments belonging to the same TCP packet. It is possible because the MAC layer itself executes the segmentation procedure.

Figure 10 shows the TCP packet loss probability for a connection that adopts a MSS=1332 bytes versus the *ritr_thresh*. We observe that the unnumbered ARQ scheme utilized by the MAC layer is very efficient, and for a *ritr_thresh* equal to 4 the lost TCP packets are less then one every 10000 packets sent. Furthermore, we observe that a bursty channel is more critical than a uniform channel, because the concentrated errors cause to close the congestion window for the TCP more rapidly than spare errors. Finally, in the two bursty cases considered there are not significant differences.

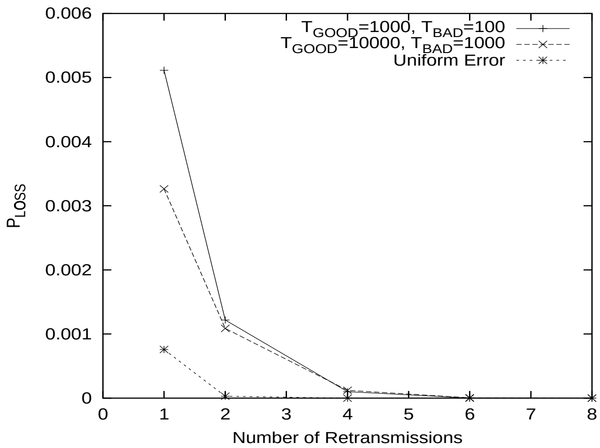

Figure 11. TCP packet loss probability with MSS=315 bytes

Figure 11 shows the same experiments when the connection adopts a MSS=315. In Figure 11, the TCP packet-loss probability shows the same behavior as in figure 10. Hence, similar considerations can be taken.

We can conclude that the ARQ scheme adopted by Bluetooth is effective to hide the channel errors to the transport layer, just with a *ritr_thresh* of 4.

5. Conclusions and Future Work

The paper proposes a new scheduling algorithm, named EDC, for the Bluetooth MAC layer. EDC is evaluated when the Bluetooth technology is used as the wireless technology for the Internet access. EDC exploits master knowledge of local queues' occupancy to avoid NULL packet transmission, and it employs two polling cycles with different polling rules to guarantee a separate and fair treatment of uplink and downlink connections. EDC significantly improves the throughput of TCP connections when compared to a RR scheduler.

In this paper we have investigated the importance of the MSS parameter in determining the TCP throughput. In particular, we have shown that the presence of TCP connections that adopt different MSSs causes unfairness. Further studies are necessary to improve the fairness of Bluetooth technology with regard to TCP connections with different characteristics. In all the experiments executed in this work we have only considered sources placed in either the master or a slave. However the traffic sources can be located in any point in the Internet. The impact of the delays introduced when the TCP flows cross the real Internet network will be studied in future works.

6. References

[1] Bluetooth Special Interest Group, "Specification of the Bluetooth System 1.0b, Volume 1: Core", December 1999.

[2] B. A. Miller, C. Bisdikian, "Bluetooth Revealed", Prentice Hall, 2000.

[3] R. Bruno, M. Conti, E. Gregori. Chapter 4: "Traffic Integration in Personal, Local and Geographical Wireless Networks", in "*Handbook of Wireless Networks and Mobile Computing*", John Wiley & Sons, New York.

[4] R. Bruno, M. Conti, E. Gregori, "WLAN technologies for mobile ad hoc networks", Proc. *Hawaii International Conference on System Sciences, (HICSS-34),* Maui, Hawaii, January 3-6, 2001.

[5] W. Richard Stevens, "TCP/IP Illustrated, Volume 1: The Protocols", Addison-Wesly , 1994.

[6] N. Johansson, U. Korner, P. Johansson, "Performance Evaluation of Scheduling Algorithm for Bluetooth", Proc. *IFIP Broadband Communications*, Hong Kong, November 1999.

[7] N. Johansson, M. Kihl, U. Korner, "TCP/IP over Bluetooth Wireless Ad-hoc Network", Proc. IFIP Networking 2000, Paris, May 1999, pp. 799-810.

[8] M. Kalia. Deepack Bansal, R. Shorey, "Data Scheduling and SAR for Bluetooth MAC", Proc. *IEEE VTC 2000*, Tokyo, Japan. May 2000.

[9] A. Das, A. Ghose, A. Razdan, H. Saran, R. Shorey, "Efficient Performance of Asynchronous Data Traffic over Bleutooth Wireless Ad-hoc Network", Proc. *IEEE INFOCOM 2001*, Anchorage, AK, USA, April 2001.

[10] R. Bruno, M. Conti, E. Gregori, "Bluetooth: architecture, protocols and scheduling algorithms". *Cluster Computing*, to be published.

[11] M. Zorzi, R.R. Rao, "On the statistics of block errors in bursty channels", *IEEE Trans on Comm.*, Vol. 45, N.6 1997, pp. 660-667.

[12] S. Zurbes, W. Stahl, K. Matheus, J. Haartsen, "Radio Network Performance of Bluetooth", Proc. *ICC 2000*, New Orleans, USA, June 2000, pp.1563-1567.