A Performance Study of Packet Scheduling Algorithms for Coordinating Colocated Bluetooth and IEEE 802.11b in a Linux Machine

Hoi Kit Yip and Yu-Kwong Kwok
Department of Electrical and Electronic Engineering
The University of Hong Kong, Pokfulam Road, Hong Kong
Corresponding Author: Yu-Kwong Kwok (email: *ykwok@hku.hk*)

Abstract— **Due to the proliferation of hand-held short-range communication devices, coexistence between Bluetooth and IEEE 802.11b has become a performance critical issue. In this study, we performed an actual implementation of a Linux based network access point (NAP), in which Bluetooth and IEEE 802.11b are colocated. Such an NAP is expected to be crucial in supporting "hot-spot" systems targeted to serve nomadic users carrying either a Bluetooth or a IEEE 802.11b device. Specifically, the goal of our study is to investigate the efficacy of a software based interference coordination approach. We consider five most commonly used scheduling algorithms in a Linux environment. Our extensive experimental results obtained in a real environment indicate that a hierarchical scheduling approach exhibits the best performance in terms of aggregate bandwidth achieved by Bluetooth and IEEE 802.11b.**
Keywords: Linux, packet scheduling, wireless communications, coexistence, Bluetooth, IEEE 802.11b, interference, fair queueing.

I. Introduction

Bluetooth [5] and IEEE 802.11b [2] short-range wireless technologies are rapidly proliferating in the various ubiquitous hand-held gadgets. This is a direct consequence of their low-cost (both operate in the free ISM 2.4 GHz frequency spectrum) and low-power features. Indeed, in "hot-spot" systems (e.g., wireless cafe, airport, convention center, etc.), a network access point (NAP) is expected to serve nomadic users carrying either Bluetooth or IEEE 802.11b devices. As such, transceivers with these two disparate (albeit both operate in the same frequency band) short-range wireless technologies will "coexist" within a small geographical area. Such coexistence, if not coordinated judiciously, will inevitably lead to degraded performance (i.e., low bandwidth) for both types of devices.

To combat this coexistence problem, in a broad sense, there are currently two different approaches. The first approach is the MAC (multiple access control) layer approach, in which the firmware of the Bluetooth and IEEE 802.11b WLAN devices have to be significantly modified. Recently, this approach has been employed in some commercial products, such as TrueRadio by Mobilian [15] and Blue802 by Silicon Wave [17]. In this approach, two types of mechanisms can be used: collaborative and non-collaborative. Collaborative mechanisms require a centralized controller. For instance, in MEHTA (MAC Enhanced Temporal Algorithm) [10], traffic information is exchanged between the Bluetooth and 802.11b firmware to calculate the accurate traffic timings at the MAC layer, avoiding interference by proper synchronizations. Non-collaborative mechanisms are mainly based on *adaptive frequency hopping* (AFH) [7], [18]. In AFH, frequency channels are dynamically classified as *good* or *bad* so that only the good ones are used in the frequency hopping process. However, AFH has limited applicability as it requires fundamental changes in the hardware and firmware of the transceiver in order to implement the channel classification and adaptive hopping mechanisms.

The second approach is the network layer approach, which is a software based technique. Specifically, the transmissions of Bluetooth and IEEE 802.11b are coordinated by a packet scheduling algorithm. The advantage of this approach is that it can be used for existing Bluetooth and IEEE 802.11b devices. Indeed, for a dual-channel NAP considered in our study, this software based approach is attractive. In this paper, we focus on this approach. We conduct an experimental study to investigate the performance of various different coordination mechanisms for colocated Bluetooth and IEEE 802.11b (using point coordination function) wireless interfaces in a single Linux machine. In Section II, we briefly survey the various scheduling algorithms that are being used in the Linux environment. In Section III, we describe the performance evaluation environment in our study. Section IV contains our experimental results obtained in a real environment. We conclude this paper in Section V.

II. Scheduling in Linux

In Linux, packet scheduling (also known as traffic control) mechanisms are mainly based on different queueing disciplines [1], [8]. Specifically, there are four basic components [1]:
1. queueing discipline: a specific scheduling algorithm;
2. classes: classification of the network traffic;
3. filters: tools to divide the network traffic into classes; and
4. policing: rules to bound network traffic.

The queueing disciplines and classes implementation source code reside in the files `net/sched/sch_*.c`, while

This research was supported by a grant from the Hong Kong Research Grants Council under project number HKU 7162/03E.

the net/sched/cls_*.c files contain the code for the filters. Each queueing discipline provides operations with the struct Qdisc_ops in include/net/pktsched.h [1]. The operations include enqueue, dequeue, requeue, reset, and so on.

The enqueue operation is triggered when a packet is enqueued on an interface (dev_queue_xmit in net/core/dev.c). Next, dev_queue_xmit calls qdisc_wakeup in net/pktsched.h to send the packet on that device. Afterward, qdisc_wakeup calls qdisc_restart in net/sched/sch_generic.c, which polls the queueing discipline so as to transmit packets. The operation qdisc_restart first fetches a packet from the queueing discipline of the device. If it is done, the device's hard_start_xmit is called to send the packet. If this fails, the packet is sent to the requeue function of the queueing discipline. The function qdisc_wakeup can also be called by a timer event, which, for example, signifies the system to send a packet at the expiration time.

Classes in Linux traffic control use two ID schemes for identification: class ID and internal ID. The internal ID must be unique within a queueing discipline while it may not be so for class ID. Classes have to be supported by queueing disciplines with the struct Qdisc_class_ops in include/net/pkt_sched.h. The Linux kernel selects a class with the enqueue operation in the queueing discipline by calling tc_classify in include/net/pkt_cls.h which returns a struct tcf_result defined in include/net/pkt_cls.h. If the return value of tc_classify is -1, it means that TC_POLICE_UNSPEC, i.e., unclassified traffic, occurs. The header file include/linux/pkt_cls.h declares the return values of tc_classify.

For locally generated traffic, there is a shortcut to select the class. If the skb->priority (struct sk_buff in include/linux/skbuff.h) contains the ID of a class of the current queueing discipline, that class is selected. The struct tcf_proto_ops in include/net/pkt_cls.h provides the necessary functions to control the filters, such as classify, init, get, and so on. In particular, classify performs the main role as it gives the classification and returns one of the TC_POLICE_* values in include/linux/pkt_cls.h.

There are 12 types of queueing disciplines commonly supported in Linux [3], [19], namely,

1. Class Based Queue (CBQ)
2. Hierarchical Token Bucket (HTB)
3. Token Bucket Flow (TBF)
4. Clark-Shenker-Zhang (CSZ)
5. First in First Out (FIFO)
6. Priority Queueing (PRIO)
7. Priority Traffic Equalizer (TEQL)
8. Stochastic Fair Queueing (SFQ)
9. Asynchronous Transfer Mode (ATM)
10. Random Early Detection (RED)
11. Generalized RED (GRED)
12. Diffserv-Marker (DS_MARK)

In this paper, we focus on five of them: FIFO, HTB, SFQ, PRIO, and TBF. FIFO is the default Linux queueing discipline for every network interface. While both HTB and CBQ belong to the group of hierarchical link-sharing queueing disciplines, HTB is an improved version of CBQ [6]. As indicated in [8], CBQ is considered to be the most complicated queueing discipline and it is quite difficult to configure it properly. We only consider HTB in this study. HTB divides the network traffic into a hierarchy of classes. Each class can enjoy at least the minimum of the amount of bandwidth it requests. In the case of excess bandwidth available, it can be shared among the classes at the same hierarchy level [6]. SFQ is a probabilistic variant of max-min fairness queueing [14]. It uses a hash function to divide the network traffic into a pretty large number of FIFO queues, one for each connection. The traffic is then sent in a round-robin manner. PRIO divides the network traffic into a number of queues, according to their relative priorities. Packets can take their turns only if all the queues of higher priorities are empty. Within each queue, packets are managed using traditional FIFO [16]. TBF is specially designed to allow users to set the maximum rate of packets to be transmitted. Once this is set, a buffer (bucket), which holds "tokens," is created according to the maximum rate. The tokens are filled in the bucket at maximum rate set. As long as there are tokens in the bucket, the packets can be dequeued and get transmitted [8].

A major difficult issue in using Linux queueing disciplines for both Bluetooth and IEEE 802.11b interfaces at the same time is that each queueing discipline can only be attached to one interface. To tackle this problem, Intermediate Queueing Device (IMQ) [12], which is a virtual networking interface in the Linux kernel, is used to bind the WLAN and Bluetooth interfaces. As all the queueing disciplines in Linux lie in the Linux Kernel, they can only be invoked by the user via user-space tools. IPROUTE2 [9] is the package providing such tools. The most important tool in the package is tc, which is a user-space program allowing us to manipulate various Linux traffic control elements. In the kernel, net/core/rtnetlink.c and include/linux/rtnetlink.h provide the *rtnetlink* which acts as a bridge to connect the user-space tools to the kernel code [1].

III. Experimental Environment

The design of the Linux traffic control architecture contains ready-made classes and filters which are for IP (Internet Protocol) traffic. Thus, it is natural for Linux to schedule the network traffic at the IP level. In this study, we use the environment as shown in Figure 1.

In the setup depicted in Figure 1, all the links were brought up to the IP level. And each of the two clients downloaded a 600 MB file from the NAP (i.e., the Linux machine) via the FTP protocol. For the NAP (or FTP server), the outflow transmission was preprocessed by some QoS techniques in the Linux kernel. As mentioned above, the queuing disciplines (qdisc) considered are: HTB, PRIO, SFQ, and TBF. All the Linux traffic control set-

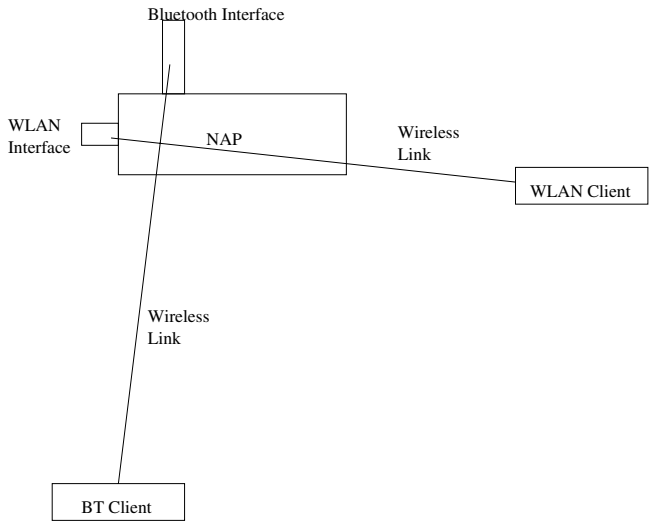

Fig. 1. The testing environment.

tings done in the tests are for *egress* traffic only. That is, we only scheduled the out-bound traffics of the NAP (the FTP download traffics of the clients).

Furthermore, all the three machines are Linux PCs. The FTP server used is the WU-2.6.1-16 bundled with Redhat 7.1 and the ftp client software is the NCFTP client. We use the modified `xnetload` [13] program to measure the bandwidth in bytes per second of the WLAN and BT interfaces at the NAP (only the out-bound traffic). The sampling period is around 700 secs to 800 secs in all tests.

For the WLAN driver, we used the HOSTAP driver [11]. This driver enables an easy configuration of the AP (access point) mode of the IEEE 802.11b interface. Indeed, in all the tests done, the WLAN is set in AP mode, which uses the point coordination function (PCF) as the MAC layer scheme. Moreover, the WLAN transceivers used are the Linksys WPC11 PCMCIA cards.

For the Bluetooth interface, we used the Billionton USB Bluetooth Dongle as the transceiver for both the NAP and the client. BlueZ [4] driver is the software driver of the devices. The Bluetooth connection is bought up to the IP level with the PAN (personal area network) profile [5] of Bluetooth, in which Bluetooth network encapsulation protocol (BNEP) and Bluetooth logical link control and adaptation protocol (L2CAP) [5] are employed. As the NAP is the master and the client is the slave, we deliberately set DH5 as packet type for network transmission at the NAP because DH5 gives the fastest data rate for the Bluetooth as the baseband layer at 723 Kbits/sec, which is about 90.375 KBytes/sec.

The following four types of tests were done:

FIFO. This is the Linux default scheduling setting.

HTB+SFQ. In this configuration, we used HTB as the first hierarchy to divide the traffic depending on the interfaces, i.e., WLAN and Bluetooth. Then, within each queue, an SFQ is attached. As indicated in Figure 2, the WLAN interface is at class 1:2 while the Bluetooth interface is at class 1:3. According to the HTB property, any excess bandwidth from class 1:2 or class 1:3 will be shared by them, as they are under the class 1:0.

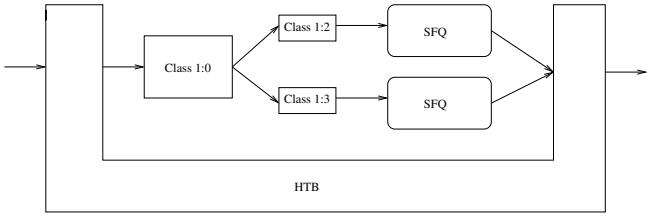

Fig. 2. The HTB+SFQ configuration.

PRIO+TBF. It is widely conceived that Bluetooth transmission survives more robustly than the IEEE 802.11b transmission in a colocated environment. Thus, we used the PRIO queueing discipline in Linux to give the WLAN connections a higher priority class. However, in order not to starve the traffic with lower priority (the Bluetooth traffic in this case), we attached a TBF scheduling algorithm for each of the class, as illustrated in Figure 3.

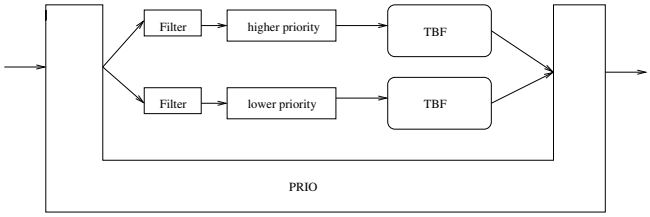

Fig. 3. The PRIO+TBF configuration.

SFQ. Finally, we simply used SFQ to schedule the WLAN and Bluetooth traffics in a mixed manner at the IMQ interface. We set ten seconds as the time for the system to change its hash functions to increase the randomness of the algorithm.

In our tests, we used the IMQ virtual network device to bind the WLAN and the Bluetooth interfaces together to be the IMQ0 interface in the Linux implementation. Thus, using the modified `xnetload` [13] program to monitor the network bandwidth in bytes/sec at the IMQ0 interface automatically gives us the aggregate bandwidth of both the WLAN and the Bluetooth interfaces. For simplicity, the aggregate bandwidth of the two interfaces is our metric of performance comparison among different queueing disciplines.

IV. Results

In this study, we compare the performance of different queueing disciplines in two major aspects:

- trend of the peak bandwidth; and
- range of major fluctuation of bandwidth.

We assume that the network transmission is performed in the *best effort* manner. Hence, the trend of the peak bandwidth could indicate us the channel condition. In this study, we presume that the channel condition varies according to the interference between WLAN and Bluetooth.

Among all the results, we notice that the WLAN transmission fluctuates every 5 seconds. And there exists a cycle of fluctuation of around 90 to 100 sec. Relative to WLAN, Bluetooth transmission fluctuates in a more random manner, from which no clear cycle can be seen. Such pattern can be seen in the figures below, which show the aggregate bandwidth of WLAN and Bluetooth. As the fluctuation could be as wide as from 100 KBytes/sec to 700 Kbytes/sec in extreme cases, it is more appropriate to consider the range of major fluctuation.

A. Default Scheduling in Linux

In Figure 4, we can see that the peak bandwidth of WLAN and Bluetooth together exhibits a gradual downward trend over the sampling period of around 800 seconds. The range of major fluctuation, as indicated by thicker or darker lines in the graph, is between 300 KBytes/sec and 600 KBytes/sec. A closer look at the data reveals that the downward trend of peak bandwidth results from the gradual deterioration of WLAN bandwidth under the interference of Bluetooth, as shown in Figure 5.

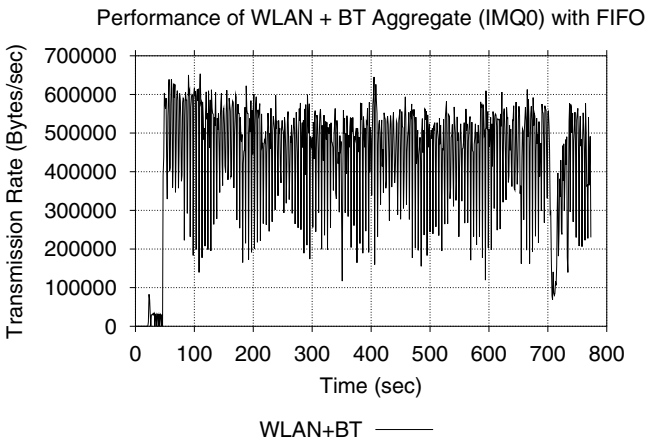

Fig. 4. Performance of WLAN and Bluetooth under default FIFO scheduling.

B. HTB+SFQ

Figure 6 shows that the peak bandwidth is steady around 600 KBytes/sec and it begins to rise well above this value toward the end of the sampling period of 800 seconds. The major range of fluctuation is from 350 KBytes/sec to around 600 KBytes/sec.

Figure 7 tells us the steady performance of HTB + SFQ mainly comes from its magic in protecting the WLAN bandwidth to be above 400 KBytes throughout the sampling period. However, the BT bandwidth is still well suppressed due to the scheme's inability to resolve the interference between WLAN and BT.

C. PRIO+TBF

The results of this queueing discipline gives us the most steady peak bandwidth among all the tests we have done,

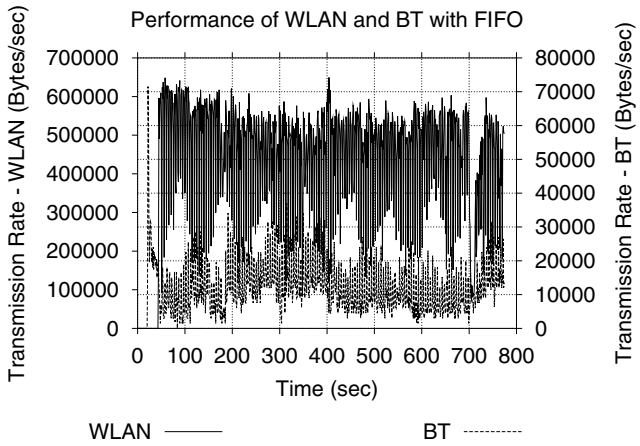

Fig. 5. Performance of WLAN and Bluetooth under default FIFO scheduling.

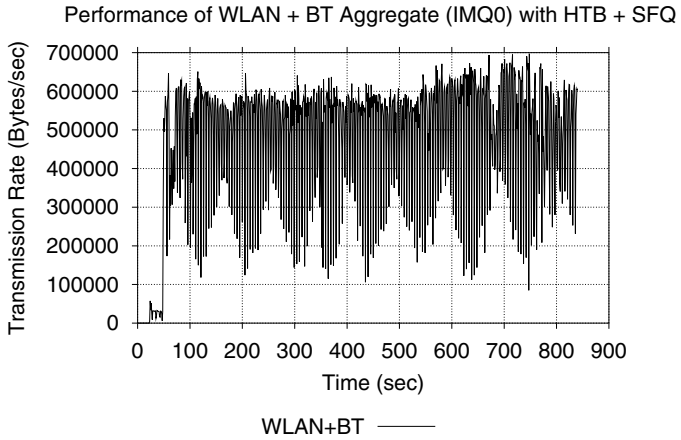

Fig. 6. Performance of the HTB+SFQ configuration.

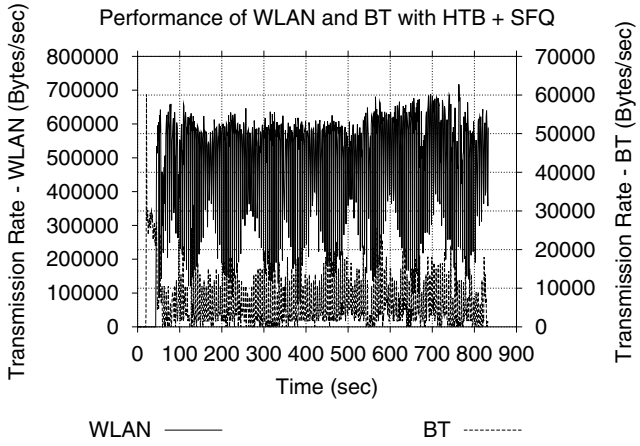

Fig. 7. Performance of WLAN and BT under HTB+SFQ configuration.

as illustrated in Figure 8. The peak bandwidth is rather stable at 660 KBytes/sec throughout the sampling period of 700 sec. More importantly, its range of major fluctuation is from 400 KBytes/sec to 660 KBytes/sec, indicating a more robust performance.

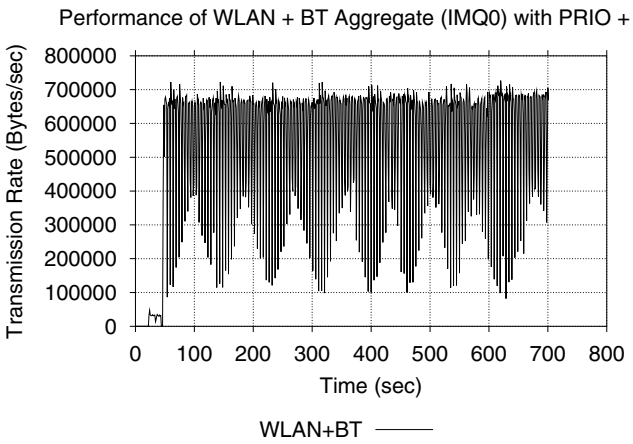

Fig. 8. Performance of the PRIO+TBF configuration.

Figure 9 gives us the picture behind the robust performance due to PRIO + TBF. Under that scheme, the bandwidth of WLAN could be steady above the 400 KBytes/sec mark. What is even more impressing is that the BT bandwidth could fluctuate in a rising trend over the sampling period, though it is still suppressed under interference. Also, this rising trend for the BT bandwidth does not accompany with a deterioration of the WLAN bandwidth. Such a rising trend could not be found in all other scheduling schemes tested in this study.

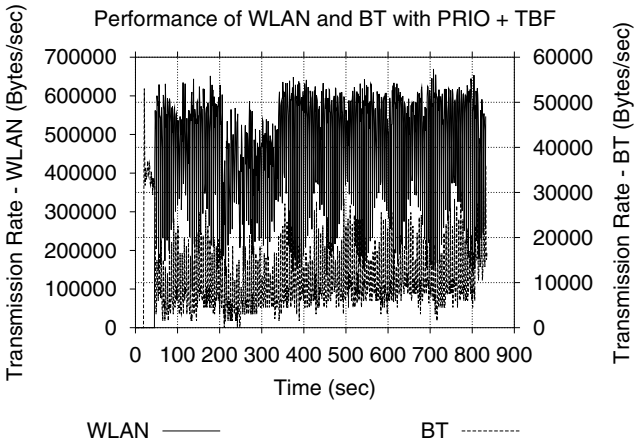

Fig. 9. Performance of WLAN and BT under PRIO+TBF configuration.

D. SFQ

As can be seen from Figure 10, the performance of SFQ looks pretty much like that of FIFO. Yet there is apparently no clear downward or upward trend of the bandwidth fluctuation. The range of major fluctuations remains wide, between 300 KBytes/sec and 600 KBytes/sec. The peak bandwidth is around 560 KBytes/sec. Though lower than that of the FIFO, it is still better than that of the FIFO in terms of steadiness.

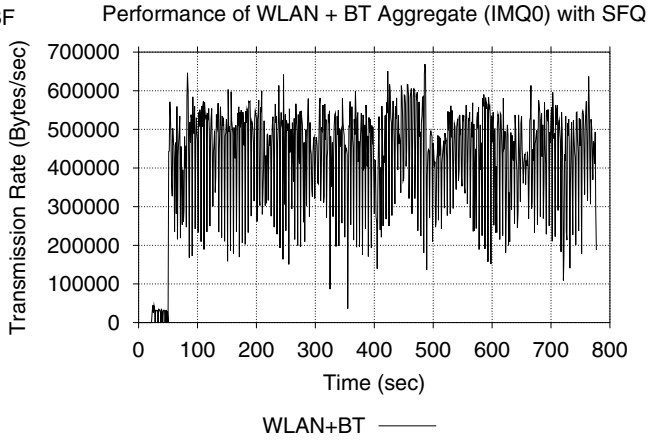

Fig. 10. Performance of SFQ.

V. Concluding Remarks

Based on the results obtained, we find that the performance ranking (in descending order) of the five scheduling algorithms are: PRIO+TBF, HTB+SFQ, SFQ, and FIFO. Specifically, our results indicate that to properly manage the two disparate (in terms of peak raw bandwidth; albeit they both operate in the ISM band) wireless technologies, a hierarchical approach seems to be more efficient. We are now designing a new hierarchical scheduling algorithm that can give even better performance in an interference-prone environment (e.g., existence of other microwave sources) in terms of aggregate bandwidth.

References

[1] W. Almesberger, "Linux Network Traffic Control—Implementation Overview," EFPL ICA., 2001.

[2] ANSI/IEEE Standard 802.11, *Local and Metropolitan Area Networks: Wireless LANs*, 1999 Edition.

[3] L. Balliache, "Differentiated Service on Linux HOWTO," http://opalsoft.net/qos/DS.htm, Aug. 2003.

[4] BlueZ, *Official Linux Bluetooth Protocol Stack*, http://www.bluez.org/, 2003.

[5] Bluetooth.org, *Bluetooth Specifications*, http://www.bluetooth.com/dev/specifications.asp, 2003.

[6] M. Devera, *HTB Linux Queueing Discipline Manual: User Guide*, http://luxik.cdi.cz/~devik/qos/htb/manual/userg.htm, 2003.

[7] H. Gan and B. Treister, "Adaptive Frequency Hopping Implementation Proposals for IEEE 802.15.1/2 WPAN," IEEE 802.15-00/367r0, http://www.ieee802.org/15/pub/TG2-Coexistence-Mechanisms.html, Nov. 2000.

[8] B. Hubert, G. Maxwell, R. van Mook, M. van Oosterhout, P. B. Schroeder, and J. Spaans, "Linux 2.4 Advanced Routing and Traffic Control HOWTO," http://www.ds9a.nl/2.4Routing, Apr. 2001.

[9] A. Kuznetsov, *IPROUTE2*, ftp://ftp.inr.ac.ru/ip-routing/, 2003.

[10] J. Linseed, "MEHTA: A Method for Coexistence between Co-located 802.11b and Bluetooth Systems," IEEE 802.15-00/360r0, http://www.ieee802.org/15/pub/TG2.html, Nov. 2000.

[11] J. Malinen, "Host AP Driver for Intersil Prism2/2.5/3," SSH Security Communications Corp., http://hostap.epitest.fi/, 2003.
[12] P. McHardy, *Intermediate Queueing Device*. http://trash.net/~kaber/imq/, 2003.
[13] R.F. Smith *xnetload*, http://www.xs4all.nl/ rsmith/software/, 2003.
[14] P. E. McKenney, "Stochastic Fairness Queuing," *Proc. INFOCOM'90*, June 1990.
[15] Mobilian, *TrueRadio*, http://www.mobilian.com/images/simop_final.pdf, 2003.
[16] C. Semeria, "Supporting Differentiated Service Classes: Queue Scheduling Disciplines," Juniper Networks, Dec. 2001.
[17] Silicon Wave, *Blue802*, http://www.siliconwave.com/Blue802.html, 2003.
[18] B. Treister, H. B. Gan, K. C. Chen, H. K. Chen, A. Batra, and O. Eliezer, "Components of the AFH Mechanism," IEEE 802.15-01/252r0, http://www.ieee802.org/15/pub/TG2-Coexistence-Mechanisms.html, May 2001.
[19] L. Wischhof and J. W. Lockwood, "Packet Scheduling for Link-Sharing and Quality of Service Support inWireless Local Area Networks," Technical Report WUCS-01-35, Department of Computer Science, Applied Research Laboratory, Washington University, http://www.cs.wustl.edu/cs/techreports/2001/wucs-01-35.pdf, Nov. 2001.