RT-WLAN: A Soft Real-Time Extension to the ORiNOCO Linux Device Driver

Amit Jain Daji Qiao Kang G. Shin
The University of Michigan
Ann Arbor, MI 48109, USA
{amitj,dqiao,kgshin}@eecs.umich.edu

Abstract—**The current IEEE 802.11 Wireless LAN (WLAN) systems are unable to support real-time applications because the underlying contention-based MAC (Medium Access Control) protocol causes unpredictable delays. In this paper, we present the implementation details of a new RT-WLAN device driver module, which extends the original Linux device driver for the popular Agere ORiNOCO cards to support soft real-time communications. To our best knowledge, this is the first effort in providing real-time support in the WLAN environment at the device driver level. By shifting the design focus from the MAC layer, which is normally hard-coded in the NIC (Network Interface Card), to the device driver level, which is between the system kernel and the MAC layer, our scheme has a clear advantage. Users can simply replace the original ORiNOCO driver with RT-WLAN, and then enjoy the benefits of real-time communications without having to change the NIC firmware or re-compile the Linux kernel.**

RT-WLAN uses two separate queues for real-time and non-real-time traffic. The real-time queue is served according to the EDF (Earliest-Deadline-First) policy, while the non-real-time queue is served in a FIFO (First-In-First-Out) manner. Besides, an adaptive traffic smoother is implemented in RT-WLAN to regulate bursty non-real-time traffic before they are injected into the network, thus giving higher priority to in-progress real-time transmissions. Experimental results show that the desired real-time support and service differentiation among multiple real-time sessions are achieved by using RT-WLAN.

I. Introduction

In recent years, the number of laptop and palmtop users has been increasing drastically, and more and more people are relying on various wireless networks to communicate with each other and exchange information. WLAN (Wireless Local-Area Network) is the one that has received the most significant attention, because it provides higher bandwidth than wide-area cellular systems and can support multimedia services in addition to the usual data service. The IEEE 802.11 [1] is the first international standard for WLANs, and has been widely used in most commercial WLAN products available in the market, e.g., the popular Agere ORiNOCO (or formerly Lucent WaveLAN) devices [2]. The IEEE 802.11 standard specifies two different MAC (Medium Access Control) schemes: the contention-based DCF (Distributed Coordination Function) and the polling-based PCF (Point Coordination Function). At present, most 802.11-compliant products only implement the mandatory DCF, and due to the contention nature of the DCF, the current 802.11 systems yield unpredictable delay characteristics and do not support prioritized transmission of real-time traffic.

The IEEE Task Group E has been working on the new 802.11e standard [3][4], which defines enhancements to the current 802.11 MAC to support applications with QoS (Quality of Service) requirements. One of the new mechanisms is called the EDCF (Enhanced Distributed Coordination Function), which realizes the QoS support by introducing the concept of TCs (Traffic Categories). A single station may implement up to eight transmission queues whose service priorities are determined by different queue management parameters. Each queue corresponds to a certain TC. Before the new 802.11e standard is finalized by the IEEE standardization committee and introduced to the market, the DCF-mode 802.11-compliant WLAN devices are expected to continue their dominance of the market. Actually, even after the new 802.11e devices are introduced to the market, there will still be many legacy 802.11 WLAN devices deployed in various sectors. In order to support real-time applications within the current 802.11 systems, appropriate real-time extensions are essential.

A number of approaches have been proposed to support prioritized transmission of real-time traffic. The authors of [5] proposed a prioritized MAC scheme, by modifying the current 802.11 standard, which allows the real-time control traffic to co-exist with the multimedia and batch traffic. In [6], the authors presented a distributed priority scheduling technique that piggybacks the priority tag of a station's head-of-line packet onto handshake and data packets. By monitoring transmitted packets, each station maintains a scheduling table which is used to assess the station's priority level relative to other stations. The existing 802.11 backoff scheme is then modified to incorporate this scheduling table, so as to approximate the ideal schedule. However, both approaches require changes in the actual NIC (Network Interface Card) firmware, since the MAC functions are normally hard-coded in a WLAN card. In [7], an adaptive traffic smoothing scheme was proposed to support real-time traffic in the Ethernet environment. The key idea is to smooth the non-real-time traffic and give priority to real-time transmissions. The evaluation results in [7] show that adaptive traffic smoothing is very effective in providing soft real-time guarantees over Ethernet. However, the authors implemented this idea in the OS kernel, and thus, users have to re-compile the OS kernel before the adaptive traffic smoothing scheme takes effect.

We address this problem at the device driver level; in particular, we implement an enhanced Linux device driver — called RT-WLAN — for ORiNOCO cards, which extends the original

The work reported in this paper was supported in part by AFOSR under Grant No. F49620-00-1-0327 and DARPA under US AFRL Contract F30602-01-02-0527.

0-7803-7822-9/03/$17.00 © 2003 IEEE.

ORiNOCO driver to support soft real-time applications. In contrast to the approaches of changing the OS kernel or modifying the NIC firmware, our scheme has a significant advantage: it can be used along with the existing OS kernel and protocol stack as well as the off-the-shelf ORiNOCO devices, so users can simply replace the original ORiNOCO driver with RT-WLAN and enjoy the significantly better real-time support. In RT-WLAN, two separate queues are used for real-time and non-real-time traffic, respectively. The real-time queue is served according to the EDF (Earliest-Deadline-First) policy [8], while the non-real-time queue is served in a FIFO (First-In-First-Out) manner. Besides, in order to have real-time traffic access the shared wireless medium with higher priority than non-real-time traffic, we borrow the idea of adaptive traffic smoothing from [7] and implement it as part of our new device driver. In addition, being close to the actual physical layer enables us to get more timely feedback about the transmission results, thus making our approach more responsive.

The rest of this paper is organized as follows. Section II introduces the DCF of the IEEE 802.11 MAC as well as the IEEE 802.11b physical layer (PHY). The implementation details of RT-WLAN are presented in Section III. Section VI presents and evaluates the experimental results, and finally, the paper concludes with Section V.

II. System Overview

A. DCF of IEEE 802.11 MAC

The DCF, as the basic access mechanism of the IEEE 802.11 MAC, achieves automatic medium sharing among compatible stations through the use of CSMA/CA (Carrier-Sense Multiple Access with Collision Avoidance). A wireless station is allowed to transmit only if its carrier-sense mechanism determines that the medium has been idle for at least DIFS (Distributed Inter-Frame Space) time. Besides, in order to reduce the collision probability among multiple stations accessing the medium, a station is required to select a random backoff interval after deferral, or prior to attempting to transmit another frame after a successful transmission.

The SIFS (Short Inter-Frame Space), which is smaller than the DIFS, is the time interval used between reception of a data frame and transmission of its Ack frame. Using this small gap between transmissions within the frame exchange sequence prevents other stations — which are required to wait for the medium to be idle for a longer gap (e.g., at least DIFS time) — from attempting to use the medium, thus giving priority to completion of the in-progress frame exchange. The timing of successful frame exchanges is shown in Fig. 1. On the other hand, if no Ack frame is received due possibly to a collision on the wireless channel, as shown in Fig. 2, the transmitter will contend again for the medium to retransmit the frame after an Ack timeout. According to the *Specification and Description Language* formal description of the IEEE 802.11 MAC operation [1], an Ack timeout is equal to a SIFS time, plus the Ack transmission time, and plus a Slot time. Note that the crossed block in Fig. 2 represents a frame collision.

Moreover, the DCF defines an optional mechanism, which requires the transmitter and receiver exchange short RTS (Request-To-Send) and CTS (Clear-To-Send) control frames prior to the actual data transmission.

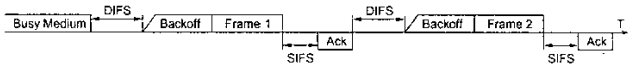


Fig. 1. Timing of successful frame exchanges under the DCF

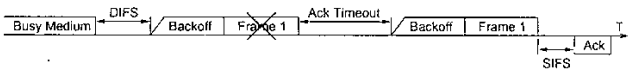


Fig. 2. Frame re-transmission due to collision

B. Backoff Behavior of IEEE 802.11 DCF

The random backoff interval is in the unit of *tSlotTime*, and this random number is drawn from a uniform distribution over the interval [0, *CW*], where *CW* is the contention window size and its initial value is *aCWmin*. In the case of an unsuccessful transmission, *CW* is updated to $[2\times(CW+1)-1]$. Once *CW* reaches *aCWmax*, it will remain at this value until it is reset to *aCWmin*. In the case of a successful transmission, the *CW* value is reset to *aCWmin* before the random backoff interval is selected. The average backoff interval before the i^{th} transmission attempt, or equivalently, the $(i-1)^{th}$ re-transmission attempt, denoted by $\overline{T}_{bkoff}(i)$, can be calculated by

$$\overline{T}_{bkoff}(i) = \frac{\min\left[2^{i-1}\cdot(aCWmin+1)-1, aCWmax\right]}{2}\cdot tSlotTime. \quad (1)$$

For the Agere ORiNOCO silver cards used in our experiment, if the RTS/CTS option is turned off and fragmentation is disabled, the number of frame transmission attempts is limited to 4 ($1 \le i \le 4$) before the frame is eventually discarded by the NIC and a delivery failure indication is sent back to the device driver.

Each station decrements its backoff counter every *tSlotTime* interval after the wireless medium is sensed to be idle for DIFS time. If the counter has not reached zero and the medium becomes busy again, the station freezes its counter. When the counter finally reaches zero, the station starts its transmission. Fig. 3 illustrates such an operation of decrementing the backoff counter.

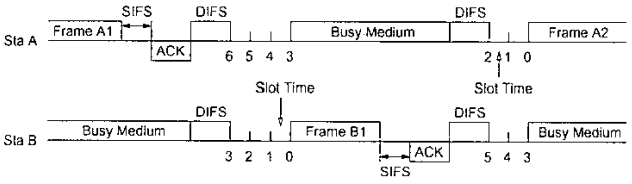


Fig. 3. An example of data frame transmissions and backoff decrements

C. IEEE 802.11b PHY

The Agere ORiNOCO silver card is designed based on the IEEE 802.11b PHY [9], which is one of the high-speed extensions to the IEEE 802.11 and is referred to as HR/DSSS (High Rate Direct Sequence Spread Spectrum). It extends the data transmission rate to 5.5 Mbps and 11 Mbps using the advanced CCK (Complimentary Code Keying) modulation technique. The frame exchange between MAC and PHY is under the control of the PLCP (Physical Layer Convergence Procedure) sublayer. Table I lists the related characteristics for the IEEE 802.11b PHY.

TABLE I
IEEE 802.11B PHY CHARACTERISTICS

Characteristics	Value	Comments
tSlotTime	20 μs	Slot time
tSIFSTime	10 μs	SIFS time
tDIFSTime	50 μs	DIFS = SIFS + 2 × Slot
aCWmin	31	min contention window size
aCWmax	1023	max contention window size
tPLCPOverhead	192 μs	PLCP overhead

D. MAC/PHY Layer Overheads

In the IEEE 802.11 MAC, each MAC data frame, or MPDU (MAC Protocol Data Unit), consists of the following components: *MAC header*, *frame body* of variable length, and *FCS* (Frame Check Sequence). The MAC overheads due to the MAC header and the FCS are 28 octets in total. Besides, the size of an Ack frame is 14 octets. During the transmission, a PLCP preamble and a PLCP header are added to an MPDU to create a PPDU (PLCP Protocol Data Unit). In the IEEE 802.11b PHY, the PLCP preamble is 144 bits and the PLCP header is 48 bits, and both are transmitted at 1 Mbps. So, the PLCP overhead is 192 μs.

Therefore, the time for a data frame with ℓ octets payload to be transmitted over the IEEE 802.11b PHY at rate r (Mbps) is

$$\begin{aligned} T_{data}(\ell, r) &= tPLCPOverhead + \frac{(\ell + 28) \cdot 8}{r} \\ &= 192 + \frac{(\ell + 28) \cdot 8}{r} \quad (\mu s). \end{aligned} \tag{2}$$

Similarly, the Ack transmission time at rate r (Mbps) is

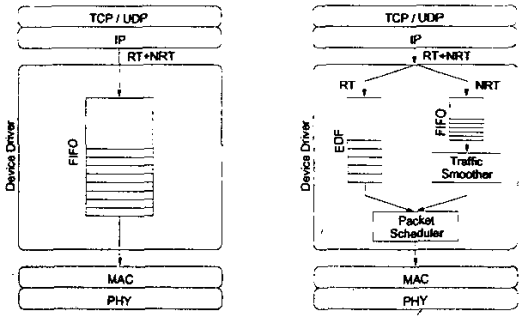
$$\begin{aligned} T_{ack}(r) &= tPLCPOverhead + \frac{14 \cdot 8}{r} \\ &= 192 + \frac{112}{r} \quad (\mu s). \end{aligned} \tag{3}$$

III. RT-WLAN

RT-WLAN is implemented by modifying the original Linux device driver for Agere ORiNOCO cards (*orinoco.c* and *orinoco_cs.c*, version 0.08 [10]). The versions of the Linux kernel and the PCMCIA package, which RT-WLAN is based on, are 2.4.12 and 3.1.31, respectively. The key modification is to add soft real-time extensions to the original driver so that the deadline requirements of the real-time applications can be better guaranteed.

As shown in Fig. 4(a), the original ORiNOCO driver simply serves the packets in a FIFO (First-In-First-Out) manner without differentiation between RT (Real-Time) and NRT (Non-Real-Time) traffic. In contrast, RT-WLAN provides separate queues for RT and NRT traffic, and the service preference is given to the RT queue. Besides, in order to have most real-time packets meet their deadlines, we apply the EDF (Earliest-Deadline-First) policy to the RT queue in RT-WLAN, so that the real-time packets with the closest deadlines are served first in the RT queue.

Note that the above extension only provides the service differentiation between RT and NRT traffic, as well as among the RT packets, within the same station. Hence, RT traffic may still suffer high latency due to the potential collisions with other traffic on the shared wireless medium and the consequent backoff operations according to the IEEE 802.11 standard. A large burst of NRT traffic at one station makes it very hard to provide bounded transmission delays for the RT traffic at another station. To deal with this problem, we apply adaptive traffic smoothing [7] to NRT traffic in RT-WLAN. The key idea is to regulate bursty NRT traffic before they are injected into the network, thus giving higher priority to in-progress real-time transmissions. Since RT traffic (e.g., multimedia or real-time control applications) usually arrives pseudo-periodically, it need not be smoothed [7]. The RT-WLAN architecture is illustrated in Fig. 4(b).



(a) Original ORiNOCO driver (b) RT-WLAN

Fig. 4. Comparison of two device driver architectures

A. User Interface

We have provided well-formulated APIs that are easily usable by application programmers. An application can indicate whether the packet it creates is a real-time packet, and specify the corresponding deadline information, if necessary, by using the function call: *set_priority(int packet_type, double relative_deadline)*. The *packet_type* parameter can take the value of 0 (for non-real-time packets) or 1 (for real-time packets). The *relative_deadline* parameter specifies the relative deadline that each real-time packet should try to meet after it is generated. If a packet is specified as a non-real-time packet, the value of *relative_deadline* is simply ignored.

The *set_priority()* function call is implemented by using the *setsockopt()* system call. The real-time and non-real-time packets are differentiated by setting the TOS (Type-Of-Service) field in the IP header. The absolute deadline of each real-time packet is obtained by adding the relative deadline to the current time at the instant of packet generation, and this deadline value is carried in the IP_OPTIONS field of the IP header. Besides, we extend the *ioctl()* system call by which the application programmer can revert back easily to the original ORiNOCO driver.

B. RT Queue and EDF Policy

In RT-WLAN, the real-time packets are served according to the EDF policy. A packet with a smaller absolute deadline receives priority over other packets with larger deadlines. Therefore, the RT queue is maintained by keeping the real-time packets in the increasing order of their absolute deadlines, and the packet with

the earliest deadline is always positioned at the head of the RT queue. Whenever a new real-time packet arrives from the upper layer, an appropriate position will be found for this new packet so as to maintain the sorted order.

C. NRT Queue and Adaptive Traffic Smoother

In RT-WLAN, the NRT queue is maintained in a FIFO manner: all the non-real-time packets are served in the order that they were en-queued. Besides, RT-WLAN requires each packet to pass through an additional traffic smoother before it is actually de-queued. This traffic smoother decides whether a non-real-time packet should be sent directly to the NIC or returned to the NRT queue for a deferred transmission.

A traffic smoother regulates bursty NRT traffic to reduce the chance of packet collisions and keeps the network utilization under a certain limit. More specifically, a traffic smoother regulates the NRT packet stream using a credit bucket, which is the same as the well-known leaky-bucket regulator [11]. The credit bucket has two parameters: CBD (Credit Bucket Depth) and RP (Refresh Period). A credit of CBD bytes are replenished into the bucket every RP seconds, so the station input limit is given by CBD/RP. The traffic smoother used in RT-WLAN is adaptive in the sense that the station input limit may vary according to the current network utilization. It uses a simple adaptation mechanism called the HIMD (Harmonic-Increase and Multiplicative-Decrease) adaptation as follows. HIMD decreases RP by a fixed constant δ every τ seconds when the network utilization is low, thus increasing the station input limit harmonically. The station input limit may be increased as long as the overall network utilization does not cause real-time packets to experience larger delays. On the other hand, whenever a non-real-time packet reaches the traffic smoother, the traffic smoother will check the time instant when the network utilization was last indicated high and compare it with the current time. If this time difference falls within a certain bound α, the traffic smoother assumes that another station is trying to transmit a real-time packet. In this case, it abstains from transmission by depleting the current credits and doubling the RP, thus decreasing the station input limit multiplicatively. The values of CBD, RP, δ, τ, and α may be modified through the extended *ioctl()* system call. The procedural description of the adaptive traffic smoother is shown in Fig. 5.

```
Adaptive_Traffic_Smoother() {
    if (Last_High_Network_Utilization_Indication.Time
        ≥ (Current_Time − α)) {
        send_packet_back_to_queue;
        Number_of_Credits = 0;
        RP = min(RP_max, 2*RP);
    }
    else if (Number_of_Credits > 0) {
        return NRT_packet;
    }
    else send_packet_back_to_queue;
    return NULL;
}
```

Fig. 5. Procedural description of the adaptive traffic smoother

Note that, in order to implement such an adaptive traffic smoother, it is very important to detect a change in the network utilization. At the device driver level, the estimation of the network utilization can be indirectly obtained either from the collision status report by the NIC after it detects the packet collisions, or by measuring the clearing time of the NIC buffer. The latter one is used in RT-WLAN. The rationale behind it and the related analysis will be presented next.

D. NIC Buffer Clearing Time: Network Utilization Indicator

An adaptive traffic smoother in the Ethernet environment — for example, the one presented in [7] — may use the collision status report as the network utilization indicator, since most Ethernet device drivers can easily collect the collision status information by querying the NIC. However, we are dealing with the WLAN environment, and the original ORiNOCO driver does not support the collision status report. Besides, the register details of the Hermes chip-set used in the ORiNOCO silver cards are not available to public. In RT-WLAN, we get around this problem by measuring the NIC buffer clearing time as the transmission delay of a packet, and also, as the indication of the current network utilization. The NIC buffer clearing time is measured as the time interval between when a packet is copied to the NIC buffer and when a successful packet delivery is reported by the NIC to the device driver. Clearly, our scheme works correctly only if the packets are served one at a time, i.e., the NIC buffer holds at most one packet at any time. This is also the way the NIC buffer is used by the original ORiNOCO driver.

Obviously, when a packet is successfully delivered without encountering any contention and/or collision on the wireless medium, the corresponding NIC buffer clearing time is small. Otherwise, the packet has to wait in the NIC buffer for a longer time until the wireless medium is cleared. To show how our scheme works, we ran two experiments, and the results are plotted in Fig. 6. The circle points represent the benchmark case when only one station is transmitting continuously. The cross points represent the case when two stations are contending for the wireless medium. In both cases, the packets are transmitted at 11 Mbps, the packet size is fixed at 1300 octets, the RTS/CTS option is turned off, and fragmentation is disabled. We have two observations. First, the NIC buffer clearing time in the benchmark case varies in a small range and the average value is less than 1000 μs. Second, in the contention case, although some packets still show small transmission delays that are comparable to the benchmark case, most of them present much higher transmission delays than the benchmark case, and there are significant gaps in between. Reasons for such phenomenon can be explained as follows.

In the benchmark case, there are no contentions on the wireless medium, so all the packets are successfully transmitted in their first attempts. The random backoff interval before the transmission is in the unit of *tSlotTime* (20 μs) and this random number is uniformly selected from the minimum contention window [0, 31]. Therefore, the difference between the maximum transmission delay and the minimum transmission delay is 620 μs, which is exactly what we observed from Fig. 6. By referring to Fig. 1, the average packet transmission delay in the benchmark case can be

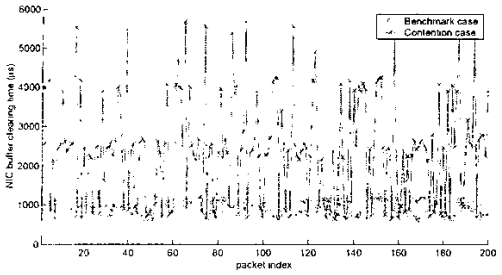


Fig. 6. Comparison of NIC buffer clearing time

calculated by[1]

$$\overline{T}_{benchmark} = tDIFSTime + \overline{T}_{bkoff}(1) + T_{data}(1300, 11) + tSIFSTime + T_{ack}(1), \quad (4)$$

where $\overline{T}_{bkoff}(\cdot)$, $T_{data}(\cdot)$, and $T_{ack}(\cdot)$ are given by Eqs. (1), (2), and (3), respectively. In the contention case, there are three possible scenarios resulting in extra delay of a packet transmission.

In the first scenario, the wireless channel is busy due to an in-progress transmission when the packet arrives the NIC buffer. The extra waiting time (Δ_{busy}) could be any value from 0 to a full packet transmission time, so it is difficult to distinguish this scenario from the benchmark case.

In the second scenario, the wireless station freezes its backoff counter since the other station starts transmitting first. By referring to Fig. 3, the extra waiting time is given by

$$\begin{aligned}\Delta_{freeze} &= T_{data}(1300, 11) + tSIFSTime + T_{ack}(1) \\ &\quad + tDIFSTime \\ &= 1424\ \mu s. \end{aligned} \quad (5)$$

In the third scenario, the transmitted packet collides on the wireless medium and the wireless station has to re-contend for the channel to re-transmit the packet. By referring to Fig. 2, the average extra waiting time is given by

$$\begin{aligned}\Delta_{collision} &= T_{data}(1300, 11) + Ack\_timeout + \overline{T}_{bkoff}(2) \\ &= T_{data}(1300, 11) + \overline{T}_{bkoff}(2) \\ &\quad + [tSIFSTime + T_{ack}(1) + tSlotTime] \\ &= 2024\ \mu s. \end{aligned} \quad (6)$$

The cross points between 2000 μs and 3500 μs in Fig. 6 can be explained by these two scenarios.

Note that a packet transmission may experience multiple backoff freezes and/or collisions, thus resulting in even larger extra delays — for example, the cross points above 3500 μs.

Based on the above analysis, in RT-WLAN, we select 2000 μs as the threshold: any NIC buffer clearing time larger than 2000 μs indicates that the current network utilization is high. Actually, using the NIC buffer clearing time as the network utilization indicator is more accurate than using the collision status report, since packet collision is only one of the above three scenarios that may cause extra delay of a packet transmission.

E. Packet Scheduler

The procedural description of the packet scheduler is shown in Fig. 7. It monitors both the RT and NRT queues and gives priority to the RT queue over the NRT queue. Only NRT traffic is smoothed in order to keep the station traffic arrival rate — which includes both RT and NRT traffic — under the station input limit.

```
Packet_Scheduler() {
    if (RT_Queue.size > 0) {
        remove_the_packet_from_head_of_RT_queue;
        send_packet_to_NIC;
        Number_of_Credits = Number_of_Credits
                            − RT_Packet.size;
    }
    else if (NRT_Queue.size > 0) {
        NRT_packet = Adaptive_Traffic_Smoother();
        if (NRT_packet ≠ NULL) {
            send_packet_to_NIC;
            Number_of_Credits = Number_of_Credits
                                − NRT_Packet.size;
        }
    }
}
```

Fig. 7. Procedural description of the packet scheduler

If the RT queue is not empty, the real-time packet at the head of the RT queue is immediately transferred to the NIC, regardless of the number of available credits, and as many credits as the size of the packet are removed from the credit bucket. So the balance of credits could be negative. On the other hand, for a non-real-time packet, the adaptive traffic smoother is called upon to decide whether it should be transferred to the NIC.

IV. Performance Evaluation

In this section, we experimentally evaluate the effectiveness of our RT-WLAN device driver. The Agere ORiNOCO silver cards are used for wireless communications between laptops and are running in the IBSS (Independent Basic Service Set) ad hoc mode.

For all the traffic sources used in the experiments, packets are generated in succession and transmitted at 11 Mbps. The packet size is fixed at 1300 octets, the RTS/CTS option is turned off and fragmentation is disabled.[2] Moreover, for a real-time packet we measure the time interval between when it is generated and when it is successfully delivered by the NIC. This time interval is referred to as the latency the real-time packet experiences, which includes the queuing delay as well as the transmission delay. The duration of each experiment run is 45 seconds.

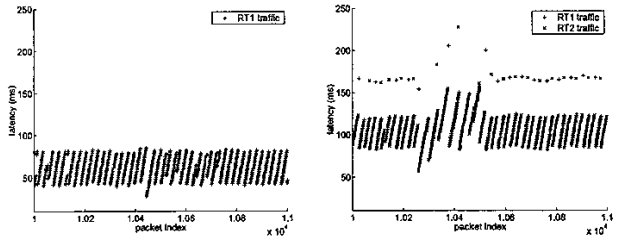
[1] Based on our calculation, the average transmission delay in the benchmark case should be around 1800 μs. However, it is quite different from our experimental results (less than 1000 μs). This may be due to our mis-interpretation of the HREG_EV_TX event [12], which we use as the indication of a successful packet delivery. Fortunately, the observed delay difference between the benchmark case and the contention case is still consistent with our analysis. Besides, we assume that the Ack frames are transmitted at the most conservative rate of 1 Mbps.

[2] Experimental results, when the RTS/CTS option is turned on and/or fragmentation is enabled, yield very similar observations to what we will present in this section, and hence, are not included in the paper.

A. Peer-to-Peer Real-Time Streaming

In this experiment, only two laptops are communicating with each other. The transmitter has two real-time traffic sources, namely RT1 and RT2. The purpose is to show the benefit of applying the EDF policy to the RT queue.

First, we investigate the behavior of the original ORiNOCO driver. Figs. 8(a) and (b) represent the benchmark case when only RT1 is activated and the case when both sources are activated, respectively. We can see that RT1 latency in the benchmark case is always less than 100 ms, and when both traffic sources are activated, the latency performances of both RT1 and RT2 are equally affected and deviate significantly from the benchmark case. Based on this observation, in the following experiments, we set the relative deadline for RT1 traffic to 140 ms such that all the RT1 packets in the benchmark case will meet the deadline requirement, while a significant amount of RT1 packets will miss the deadline when both traffic sources are activated. Then, we vary the relative deadline for RT2 traffic to see the benefit of applying EDF.

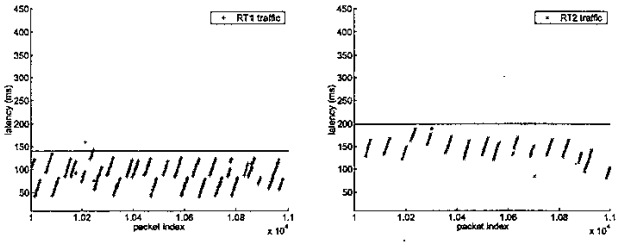


(a) single RT traffic (benchmark) (b) two RT traffic

Fig. 8. Latency comparison for RT traffic with a FIFO queue

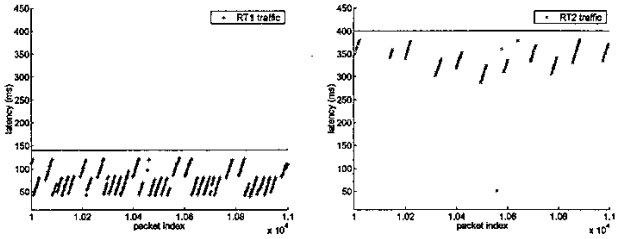
Now, we replace the original ORiNOCO driver with RT-WLAN. Fig. 9 shows the results when the relative deadline for RT2 traffic is set to 200 ms. The thick solid lines represent the relative deadlines for both traffic. Due to the less stringent deadline requirement of RT2 traffic, a higher transmission priority is given to RT1 traffic. As a result, less RT1 packets miss their deadlines at the expense of RT2 packets experiencing larger latencies. In Fig. 9, the integer number along the X-axis represents the order of the transmitted packets, which may belong to either RT1 or RT2. We can see that both sub-figures show certain degrees of data sparseness and the empty positions actually correspond to the packet transmissions from the other source. Clearly, more RT1 packets are transmitted. Similar observations can be found in Fig. 10, where the relative deadline for RT2 traffic is increased to 400 ms, and as expected, even less RT1 packets miss their deadlines and more transmission opportunities are offered to RT1.

In order to evaluate the benefit of using an EDF RT queue quantitatively, we calculate the deadline miss ratio for RT1 traffic and show the results in Fig. 11(a). We also count the number of packets transmitted from either source, from RT1 only, and from RT2 only during the 45-second experiment run, and the results are shown in Fig. 11(b). The X-axis represents the difference of the relative deadlines of RT1 and RT2 traffic. Note that, when the deadline difference is zero, all the packets are actually served in a FIFO manner, so RT1 and RT2 are equally competing for the service. As a result, almost an equal number of RT1 and RT2 packets are transmitted, and RT1 traffic presents a large deadline miss ratio (>0.06). As the deadline difference increases, RT1 is assigned a higher transmission priority, thus resulting in a smaller deadline miss ratio and more shares of bandwidth. On the other hand, the total number of transmitted packets remains the same regardless of the deadline difference. Based on the above observations, we draw the following conclusion: by applying the simple EDF policy to the RT queue, we are able to achieve service differentiation among multiple real-time sessions with different deadline requirements without sacrificing the total throughput.



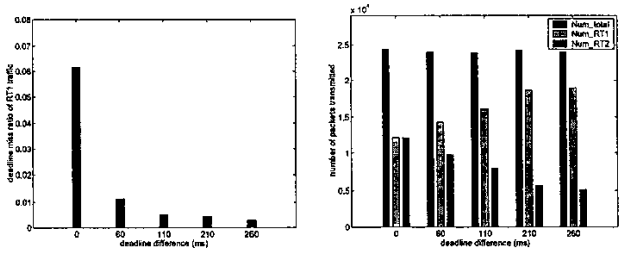
(a) RT1 (*relative_deadline* = 140 ms) (b) RT2 (*relative_deadline* = 200 ms)

Fig. 9. Latency comparison for RT traffic with an EDF queue



(a) RT1 (*relative_deadline* = 140 ms) (b) RT2 (*relative_deadline* = 400 ms)

Fig. 10. Latency comparison for RT traffic with an EDF queue



(a) Deadline miss ratio for RT1 (b) Throughput comparison

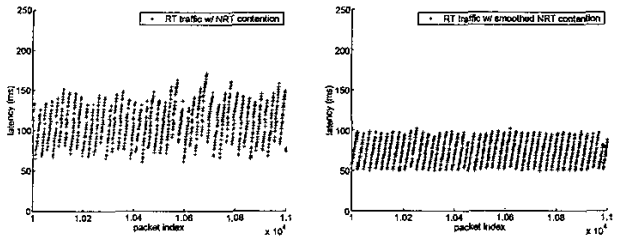
Fig. 11. More experimental results for RT traffic with an EDF queue

B. Real-Time Streaming in the Presence of Third-Party Non-Real-Time Traffic

In this experiment, three laptops are used. Two of them generate RT and NRT traffic, respectively, and the third laptop serves as the common receiver to both. RT and NRT traffic are contending for the shared wireless medium. The purpose is to show the benefit of applying adaptive traffic smoothing to NRT traffic.

We create two different scenarios in our experiment and compare their latency performances. First, NRT traffic is injected into the network through the original ORiNOCO driver, and contends

with RT traffic for the wireless medium without adaptive traffic smoothing. Second, the original ORiNOCO driver is replaced by RT-WLAN, and thus, NRT traffic is smoothed before contending for the wireless medium. The parameters of our adaptive traffic smoother are: α = 10 ms, δ = 100 μs, CBD = 1500 octets, RP_{max} = 50 ms, RP_{min} = 3 ms, and τ = 10 ms. The corresponding results are plotted in Figs. 12(a) and (b), respectively. We can see that, without adaptive traffic smoothing, RT traffic experiences much higher latency due to the NRT contention. In contrast, with adaptive traffic smoothing, the latency performance of RT traffic is only slightly affected compared to the benchmark scenario, which is shown in Fig. 8(a). This is because the traffic smoother stops sending non-real-time packets and lowers its station input limit as soon as it finds out that its on-going packet transmission experiences contention and/or collision on the wireless medium.



(a) RT (+ NRT contention) (b) RT (+ smoothed NRT contention)

Fig. 12. Latency comparison for adaptive traffic smoothing

We also compare the throughput performances for these two scenarios, and the results are shown in Fig. 13. We have three observations. First, without adaptive traffic smoothing, equal numbers of real-time and non-real-time packets are transmitted, because RT and NRT traffic are contending equally for the wireless medium. Second, with adaptive traffic smoothing, more real-time packets are transmitted, while still a reasonable number of non-real-time packets are served when the wireless medium is available. Third, there is about a 5% drop in the total throughput when adaptive traffic smoothing is applied. The rationale behind the drop is that the cautious nature of the adaptive traffic smoother results in a conservative transmission strategy for non-real-time packets. Therefore, the wireless medium may not be fully-utilized under our experimental setup.

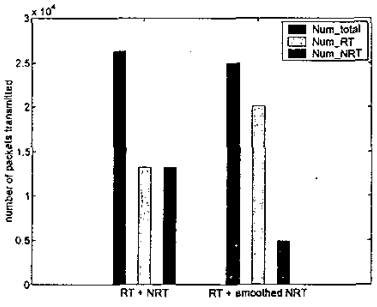


Fig. 13. Throughput comparison for adaptive traffic smoothing

V. Conclusion and Future Work

In this paper, we present the implementation details of RT-WLAN, a soft real-time extension to the original ORiNOCO Linux device driver, which supports the IEEE 802.11b-compliant ORiNOCO silver cards under the Linux operating system. RT-WLAN is implemented as a loadable device driver module and is very easy to deploy. Users can simply replace the original ORiNOCO driver with RT-WLAN, and then realize soft real-time communications without having to change the NIC firmware or re-compile the Linux kernel.

RT-WLAN uses two separate queues for RT and NRT traffic. The high-priority RT queue is served according to the EDF policy, while the low-priority NRT queue is served in a FIFO manner. Besides, an adaptive traffic smoother is implemented in RT-WLAN to regulate bursty NRT traffic before they are injected into the network, thus giving higher priority to in-progress real-time transmissions. Experimental results show that the latency of RT traffic is only slightly affected even when a significant amount of NRT network traffic is present, and the service differentiation among multiple real-time sessions is also achieved.

We plan to extend our work in the following directions. First, since the focus of RT-WLAN is to give transmission priority to real-time traffic, so when there are only non-real-time traffic in the network, the bandwidth utilization may be low because the adaptive traffic smoother results in conservative transmission attempts of non-real-time packets. We are working on the enhancement of the current traffic smoother to deal with this situation. Second, we will add multiple non-real-time queues to RT-WLAN, and each non-real-time queue is followed by a different traffic smoother. In this way, we may achieve service differentiation among non-real-time traffic as well, by mimicking the IEEE 802.11e EDCF at the device driver level.

References

[1] IEEE 802.11, *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications*, Standard, IEEE, Aug. 1999.

[2] Agere Systems, *Brochure for ORiNOCO PC Card*, Oct. 2001.

[3] IEEE 802.11e/D2.0, *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Medium Access Control (MAC) Enhancements for Quality of Service (QoS)*, Draft Supplement to IEEE 802.11 Standard - 1999 Edition, 2001.

[4] Stefan Mangold, Sunghyun Choi, Peter May, Ole Klein, Guido Hiertz, and Lothar Stibor, "IEEE 802.11e Wireless LAN for Quality of Service," in *Proc. European Wireless (EW'2002)*, Florence, Italy, Feb. 2002.

[5] Hong Ye, Gregory C. Walsh, and Linda G. Bushnell, "Real-Time Mixed-Traffic Wireless Network," *IEEE Transactions on Industrial Electronics*, vol. 48, no. 5, pp. 883–890, Oct. 2001.

[6] V. Kanodia, C. Li, A. Sabharwal, B. Sadeghi, and E. Knightly, "Distributed Multi-Hop Scheduling and Medium Access with Delay and Throughput Constraints," in *Proc. ACM MobiCom'01*, Rome, Italy, Jul. 2001.

[7] Seok-Kyu Kweon, Kang G. Shin, and Gary Workman, "Achieving Real-Time Communication over Ethernet with Adaptive Traffic Smoothing," in *Proc. IEEE Real-Time Technology and Applications Symposium*, Jun. 2000, pp. 90–100.

[8] C. M. Krishna and Kang G. Shin, *Real-Time Systems*, McGraw Hill, 1997.

[9] IEEE 802.11b, *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: High-speed Physical Layer Extension in the 2.4 GHz Band*, Supplement to IEEE 802.11 Standard, Sep. 1999.

[10] "http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Orinoco.html," Online link.

[11] R. L. Cruz, "A Calculus for Network Delay, Part I: Network Elements in Isolation," *IEEE Transaction on Information Theory*, vol. 37, pp. 114–131, Jan. 1991.

[12] Lucent Technologies Inc., *Draft Software Interface Specification for Wireless Connection Interface for WaveLAN/IEEE (HCF-Light)*, Nov. 2001, Doc No. S0005, Rev. 11.