Wireless Communication Using Real-Time Extensions to the Linux Network Subsystem

Barbara Hughes and Vinny Cahill
Distributed Systems Group,
Department of Computer Science, Trinity College, Dublin, Ireland
{barbara.hughes, vinny.cahill}@cs.tcd.ie

Abstract

Timely wireless communication is essential to allow real-time mobile applications, e.g., communication between mobile robots and inter-vehicle communication to be realized.

The current IEEE 802.11 ad hoc protocol is unable to provide real-time communication guarantees due to its underlying contention-based MAC layer. Our current research is addressing the implementation of a time-bounded MAC protocol as a layer above 802.11. The implementation of a timely MAC protocol requires predictable and deterministic behavior at the device driver level, currently unavailable in the Linux operating system.

This paper describes real-time extensions to the Linux operating system to provide real-time guarantees at the device driver level. To our knowledge, we are the first to implement a real-time ORiNOCO driver for real-time Linux. In addition we provide a low-level evaluation of the timeliness of packet transmission achievable using IEEE 802.11.

1. Introduction

With increased research in ad hoc networks in recent years new application domains such as communication between mobile robots, inter-vehicle and vehicle-to-roadside communication have evolved. Timely wireless communication is essential to allow applications in these domains to be realized.

The IEEE 802.11 wireless standards are dominant in the wireless local networking community today, and recent surveys suggest that IEEE 802.11 standards (particularly g), will dominate the WLAN roadmap for years to come. [1]. The current IEEE 802.11 wireless ad hoc protocol adopts a contention-based approach to medium-access control (MAC). A carrier sense multiple access and collision avoidance (CSMA/CA) mechanism and a random back-off scheme are used to reduce MAC contention and packet collisions. However, due to the underlying contention-based MAC layer the possibility of packet collisions is not eliminated. Thus, IEEE 802.11 is unable to guarantee timely access to the wireless medium critical for real-time communication.

Timely communication requires a real-time MAC layer, such as TBMAC[2]. TBMAC is a medium-access control protocol based on TDMA for dynamic but predictable slot allocation. TBMAC provides, with high probability, time-bounded access to the wireless medium to mobile hosts in a multi-hop ad hoc network. Our objective is to support real-time by implementing TBMAC as a layer above IEEE 802.11.

TBMAC synchronizes access to the wireless medium, but implementing the TBMAC protocol requires real-time behavior at the Linux device driver level. Standard Linux device drivers exhibit unpredictable and non real-time behavior due to interrupt handling, task scheduling and memory management in the Linux operating system. A prerequisite for a timely wireless MAC protocol is a timely Linux device driver.

In this paper we describe the implementation and evaluation of a real-time extension to the standard Linux ORiNOCO device driver and network subsystem., to provide real-time guarantees at the device driver level. To our knowledge, we are the first to implement a real-time ORiNOCO driver for real-time Linux and to provide a low-level evaluation of the timeliness of packet transmission achievable using IEEE 802.11. Our real-time wireless driver is an important basis upon which real-world, timely medium-access control protocols may be implemented and against which results, previously available in simulation only, may be evaluated.

The remainder of this paper is structured as follows: Section 2 describes the motivation, design and implementation of the real-time extension to the standard Linux network subsystem and ORiNOCO

drivers. Section 3 presents an evaluation of the real-time drivers using real-world experiments and section 4 describes our future work, the implementation of the TBMAC protocol using the real-time drivers available. Finally, Section 5 concludes this paper.

2. Design and implementation of real-time ORiNOCO drivers

2.1. Unpredictability of standard Linux drivers

Standard Linux device drivers were designed for a general purpose operating system where the timeliness of interrupt servicing, task prioritization and memory management, essential for real-time applications, are not provided. To achieve real-time behavior extensions must be implemented in the standard Linux operating system.

RTAI provides hard real-time capabilities in a Linux environment [3]. RTAI falls into the category of a hybrid operating system (RTOS) where the general-purpose operating system (e.g. standard Linux kernel device drivers) has control of most hardware resources but it is subject to the control of a deterministic hard real-time operating system (RTAI). The central idea of RTAI is that a RTOS can run alongside Linux and provide real-time capabilities, with the assumption that Linux does not divert critical hardware events or interrupts from the RTOS (as would occur using standard Linux device drivers), as it is these events that will dictate the timeliness of the system. The extent to which these real-time capabilities are realized and available depend on both the design of the real-time application, e.g., incorporating predictability and timeliness, and the level of integration with RTAI (RTAI-compliance [3]), e.g., the exclusive invocation of RTAI system operations and primitives.

Real-time behavior requires real-time design at all layers of the system architecture. For example, RTAI-compliant and non-interruptible interrupt service routines (ISRs) at the device driver level, coupled with real-time tasks, guaranteed to execute with the highest task priority, for real-time processing at the application layer. With these design considerations in mind and coupled with a brief overview of the operation of the standard Linux network subsystem, i.e., the combination of device dependent network drivers and device independent network kernel modules, it is clear that real-time behavior is not supported.

Network drivers transmit and receive packets asynchronously to/from the outside world. A loaded network driver, i.e., available for invocation, inserts a `struct net_device` data structure for each newly detected device into a global list of known available devices. The detected network device is initialized, e.g., pointers to device specific `open`, `hard_start_xmit` etc. functions are populated with device dependent references, prior to device registration in the Linux kernel.

Each packet handled by the kernel, throughout the Linux network subsystem, is contained in a socket_buffer structure (`struct sk_buff`), which is allocated dynamically, using `dev_alloc_skb` when required for either packet transfer or reception. For example, in Figure 1, the resolved device dependent `hard_start_xmit` expects a successfully allocated socket_buffer containing the physical packet as it should appear on the media, complete with transmission-level headers.

```
int (*hard_start_xmit) (struct sk_buff
*skb, struct net_device *dev);
```

Figure 1: Generic function protoype in the net_device structure

The successful allocation of a socket_buffer is critical for all network operations, directly impacting the predictability of both packet transmission and reception.

Dynamic allocation, although catering for a change in memory requirements during execution, introduces the critical situation where memory requests cannot be satisfied by the residual memory available, leading to unpredictable execution. For example, the worst-case scenario for packet transmission is that the failure to allocate a socket_buffer leads to a failure of the complete packet transfer.

General-purpose operating systems adopt a demand-paging approach to memory management [4]. However, demand paging is non-deterministic, for example, requiring access to disk, and thus is not suitable for real-time processing. The development of a real-time Linux network driver requires real-time memory management for the allocation of socket_buffers, and thus a fundamental change to the standard Linux network subsystem. Furthermore, investigating the design and implementation of one of the standard wireless network device drivers, the ORiNOCO driver, raises some issues about the real-time capabilities of the Linux network drivers in general.

To guarantee timely interrupt servicing, interrupts must be serviced in a real-time context, i.e., interrupts must be diverted first to RTAI by registering a real-time handler, to ensure predictable and guaranteed response times, and forwarded to Linux, i.e., by the standard Linux interrupt dispatcher, if Linux is also expecting the interrupt. Thus to achieve predictable and deterministic interrupt handling in the ORiNOCO driver all interrupts serviced must be diverted to RTAI in this manner. Another potential source of

unpredictability throughout the ORiNOCO network driver is that ISRs, e.g., `orinoco_ev_rx()`, are executed with interrupts enabled, with the implication that a context switch may occur during critical interrupt servicing, potentially leading to unpredictable delays for required real-time processing.

Providing real-time capabilities to the standard Linux network subsystem requires a fundamental amendment to the core functionality, and behavior, of the subsystem, i.e., the allocation and management of the `socket_buffer` structure. The use of, and interface with, the `socket_buffer` structure is omnipresent throughout the network subsystem. Thus, amending the `socket_buffer` structure has widespread consequences and cannot be treated in isolation. Furthermore, to guarantee predictability throughout the network subsystem real-time considerations must be implemented at both the kernel network modules and the low-level kernel network drivers.

The real-time network subsystem must co-exist with the standard Linux network subsystem, i.e., predictability and determinism must be guaranteed to real-time network drivers without hindering the operation of the standard Linux network subsystem.

Our real-time network subsystem, RT-WLAN, utilizes a real-time `socket_buffer` abstraction, (`RTsocket_buff`), with a new memory management module, (`MemoryManager`), and real-time network_device (`RTnet_device`), to address real-time design issues design in the standard Linux kernel network modules. The real-time design issues raised at the network device driver level are addressed in the `RTorinoco` real-time wireless driver.

2.2. Implementation

The RT-WLAN network subsystem is encapsulated in a number of hierarchical layers implemented as kernel level modules. The RT-WLAN hierarchy is illustrated in Figure 2. The interactions between the real-time modules and amendments made to support a real-time network subsystem, are described in this section.

The `rtcomm` module is at the core of RT-WLAN and is responsible for providing predictable `socket_buffer` allocation and management (`RTSocket_Buff`), the initialization and management of a real-time `net_device` abstraction (`RTDeviceWrapper`) and an encapsulation of real-time versions of standard Linux network modules, e.g., `RTNetworkInit`, the real-time version of `</drivers/net/net_init.c>`.

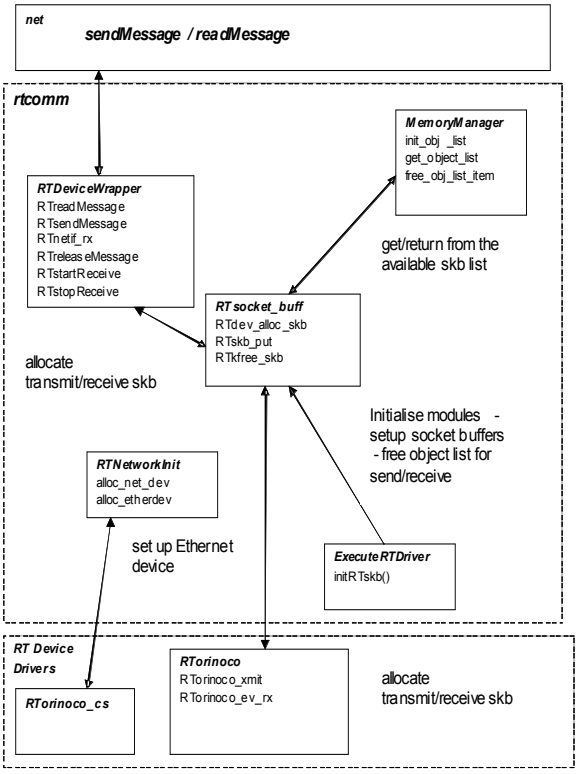

Figure 2: Interaction between modules in RT-WLAN

The provision of predictable socket_buffer allocation is achieved using the interaction between the `MemoryManager` and `RTSocket_Buff` modules. Real-time memory management design must guarantee that memory is available and maintained in physical RAM. To achieve this, the `MemoryManager` modules creates a static pool of `RTsocket_buff` structures[1] implemented as a fixed size, doubly linked-list which is guaranteed to remain in scope until explicitly removed. A call to `RTdev_alloc_skb()`, invokes the utility methods of the `MemoryManager` module to perform bounded traversal of the `socket_buffer` list, to locate and allocate an available item. The `RTSocket_Buff` module provides the real-time `socket_buffer` interface to all other modules in RT-WLAN.

A real-time network device, `RTnet_device`, structure encapsulates and extends the standard Linux `net_device` structure, with the real-time versions of the standard `net_device` services, for example, device `register` and `unregister`, as provided by the module `RTDeviceWrapper`. In a supplementary

[1] RTSocket_Buff is the real-time module and RTsocket_buff the real_time version of sk_buff.

role, RTDeviceWrapper provides an interface to send and read wireless packets from the real-time device driver, RTorinoco, illustrated in Figure 3.

```
int RTsendMessage(char *buffer, int size);
RTsocket_buff* RTreadMessage(void);
```

Figure 3: Real-time packet send/read function prototypes

As previously mentioned, the socket_buffer structure, or in this case, the real-time equivalent RTsocket_buff, is the unit of packet exchange for all transfers to/from a network device. Thus, prior to any interaction with the RTOrinoco driver, the generic data packet, identified in Figure 3 by buffer, must be transformed to a RTsocket_buff structure.

Real-time packet transfer using RTsendMessage encapsulates the allocation and population of a RTsocket_buff structure, using the RTSocket_Buff module interface, and initiates the packet transfer to the wireless card, in this case the invocation of the RTorinoco_xmit function, the header of which is shown in Figure 4.

```
RTorinoco_xmit(struct RTsocket_buff * skb , struct RTnet_device * rtndev ).
```

Figure 4: Real-time version of orinoco_xmit()

RTorinoco_xmit, is the real-time wireless interface for packet transfer and expects (and invokes) real-time network subsystem and RTAI primitives only.

In the standard Linux ORiNOCO driver, packet transfer from the wireless card is initiated upon reception of an orinoco_ev_rx interrupt notifying packet availability at the wireless interface. The packet payload is extracted and placed in a socket_buffer which is subsequently added to a queue of pending packets scheduled for future notification to higher layers, using the netif_rx interface of net_device. Real-time packet reception must not only remove the non-deterministic socket_buffer allocation, but must also eradicate the unpredictability of the pending packet notification, for an undetermined time in the future.

Substituting the dependency on standard socket_buffers with the real-time equivalent RTsocket_buff removes one source of unpredictability. Implementing a real-time version of netif_rx would eradicate the other. The RTDeviceWrapper module provides an interface to RTnetif_rx, the real-time netif_rx. Using this interface, a real-time memory pool, representing the pending packets, is maintained and immediate notification of packet availability is made to the higher layers.

The rtcomm module not only encapsulates the core functionality of the standard Linux network subsystem, but also provides an abstraction of the real-time semantics to layers above, i.e., application layer, and below, i.e., the network device driver. The lowest layer in the RT-WLAN subsystem is the real-time network device driver, encompassing the RTorinoco_cs, RTorinoco and RThermes modules.

An initial requirement for the real-time network drivers was to remove all dependencies on the standard network subsystem, i.e., by substitution with the real-time equivalent from rtcomm. In addition, specific amendments were necessary to the low-level semantics of both RTorinoco_cs and RTorinioco to achieve real-time behavior.

Timely interrupt servicing is a critical concern in a real-time network device driver. RTAI provides the required real-time guarantees by diverting the interrupt handling to RTAI and then Linux, or more precisely in this case, to PCMCIA. To achieve interrupt diversion as described, RTAI requires the specification and implementation of a real-time interrupt handler for the specific IRQ, which was implemented in this case in RTorinoco_cs. Once this interrupt handler is registered, as illustrated in Figure 5, any interrupts for which this interrupt handler is registered will be available to RTAI prior to PCMCIA.

```
retval=rt_request_global_irq(ndev->irq, orinoco_irq_handler) ;
```

Figure 5: Example of RTAI interrupt handler specification

Additional modifications were made to the RTorinoco module. For example, to guarantee ISRs execute in a real-time context, i.e., with interrupt disabled, and to provide RTAI-compliance throughout the module, i.e., RTAI versions of the standard Linux synchronization and coordination primitives used only. The latter modification was implemented throughout the real-time network drivers.

Situated directly above rtcomm, is the net layer, acting as the mediator between the application layer and the rtcomm module. The net layer, or more explicitly the RTCommWrapper module, provides a simple abstraction of the packet send and read interface of rtcomm to the application layer. The creation, manipulation and destruction of thread-safe real-time tasks are the critical real-time issues at the application layer. The behavior, e.g., periodicity, and operation of the real-time tasks are dependent on the requirements of the application. An example of packet transmission using the RT-WLAN network system is illustrated in

Figure 6, and described as follows. A real-time task, executing with the highest priority, invokes the `sendMessage` interface of the `RTCommWrapper` at the `net` layer, with a generic data buffer representing the packet payload. `sendMessage` in turn invokes `RTSendMessage`, of `rtcomm`, where the generic packet payload is inserted into a real-time socket_buffer. This socket_buffer is passed to `RTorinoco_xmit`, invoked from within RTSendMessage, for the low-level packet transmission by `RTorinoco`. The socket_buffer, with transmission-level headers added, is queued in the network device queue for transfer to the wireless card, where using IEEE 802.11 contention for the wireless medium, prior to packet transfer is started.

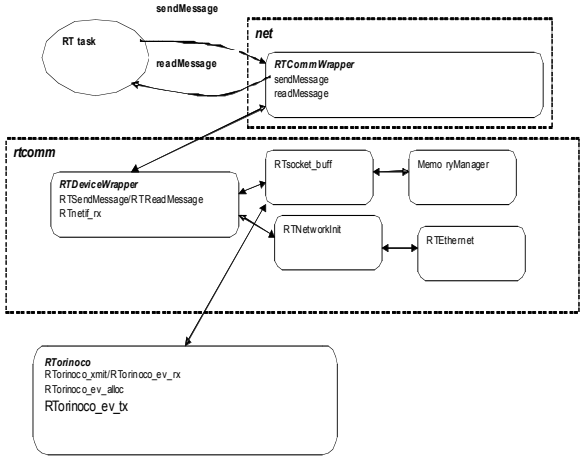

Figure 6: Packet transfer using RT-WLAN

The standard Linux network susbsystem, by design (e.g., dynamic memory allocation) and implementation (e.g., non real-time interrupt servicing), does not provide predictable and deterministic behavior. To introduce real-time behavior requires amendments throughout the network subsystem, which lead to our real-time network subsystem, RT-WLAN.

With the implementation of the RT-WLAN subsystem, we have the ability, for the first time, to provide real-time behavior in the wireless 802.11 network driver. In the next section we evaluate the predictability of the real-time drivers using real-world experiments.

3. Evaluation

3.1. Experimental setup

3.1.1. Experimental environment.

Our evaluation is based on experiments performed using two Lucent ORiNOCO 11Mbit/s Gold cards executing the real-time RTorinoco network drivers and interfacing with RT-WLAN kernel modules. The RT-WLAN network subsystem was run on two Pentium III notebooks with 1 Gigahertz, 512K L2 cache, 256MB RAM with PCMCIA wireless cards. Each notebook ran an identical version of the Red Hat Linux 7.3 operating system with the kernel 2.4.20 patch and the RTAI release version 3.0 applied.

We chose three different physical environments for our experiments to ascertain the influence that the environment, and particularly wireless interference and contention, exhibits on the results achieved. The environments selected were as follows:

- *Outdoor, open space*

  We selected a large open space area[2], free from physical obstacles, such as buildings and trees, and with a low probability of wireless interference.
- *Indoor, home environment*

  Our first indoor environment exemplifies a typical home environment, which may exhibit some wireless interference due to the existence of other wireless devices, e.g., wireless phones, typically found in the home.
- *Indoor, office environment*

  Our second indoor environment was chosen to exhibit a high probability of wireless contention. The location chosen was a computer science laboratory within range of other wireless networks and devices.

3.1.2.Metrics for 802.11 packet transmission

Throughout our experiments, all wireless packets are broadcast and all timing considerations are over one hop only. Using IEEE 802.11b, an ACK is not expected for broadcast transmissions and thus is not included in our benchmark calculations for packet transmission in the following section. Furthermore, our ORiNOCO cards, have RTS/CTS switched off, fragmentation disabled, maximum retry attempts limited to 4 and transmit broadcasts at an implicit maximum bit rate of 2Mb/s. The packet size for all experiments is 74 octets, with a fixed number of packets (1000) transmitted at a user-defined rate. The percentage of packets dropped is also analyzed, and discussed separately. Each experiment commences with one wireless host, the originator, transmitting a packet. The other wireless host receives, processes and replies with an amended packet. The experiment completes when the originator has processed the returned packet.

There were two main objectives of our real-world experiments. The first was to provide low-level timing analysis for all stages of packet transmission using Lucent ORiNOCO Gold cards, to the best of our

[2] Located at North Bull Island, Co. Dublin, Ireland.

knowledge, previously unavailable. Our second objective was to evaluate the predictability available in the real-time network device driver, the calculation of which is described in the next section.

3.1.3 Critical phases of packet transmission- latency

The critical stages for timing analysis in wireless packet transmission are illustrated in Figure 7. In RT-WLAN, packet transmission is initiated with the invocation of `sendMessage`, symbolized by $T_{start.}$ The packet is transferred to the `RTorinoco` device driver queue, implemented in software, with T_{queued} denoting when the packet has been queued. The interval between T_{queued} and T_{start} is the *software latency* incurred to transfer a packet to the device driver queue. Using RT-WLAN we expect a predictable software latency, regardless of the external physical environment or rate of packet transmission.

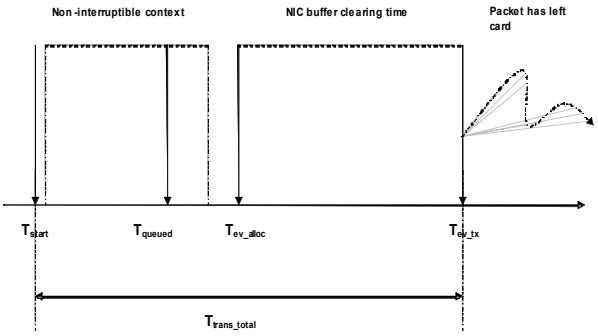

Figure 7: Timed actions in the packet transmission process

Following the transfer of the packet to the device driver queue, the latency incurred is either by *firmware*, i.e., the transfer of the packet to the physical wireless card, or, the 802.11b PHY *communication* latency, i.e., packet transfer using contention-based MAC. Both the firmware and communication latency are not within software control. The successful completion of each stage is notified via interrupts, which, using RT-WLAN, we service using the real-time interrupt handler, and record the time at which the interrupts occur. Transfer to the wireless card is denoted by T_{ev_alloc} and from the wireless card over the air, by $T_{ev\_tx.}$.

Throughout our experiments, our objective is to achieve a *predictable software latency* and to provide timing analysis of the firmware and communication latency of 802.11b, the granularity of which has not previously been available.

Due to the dominance of the IEEE 802.11 PHY communication latency in the latency for packet transmission, it is necessary to calculate lower time-bounds for the size of packets and equipment, regardless of the environment, used in our experiments.

As discussed in [5], the PLCP preamble used in the IEEE 802.11b PHY is 144 bits and the PLCP header is 48 bits, both of which are transmitted at 1Mbps. Thus the PLCP overhead is 192 μs. The time for a data frame with l octet payload to be transmitted over IEEE 802.11b PHY at rate r (Mbps) is

$$T_{data}(l,r) = tPLCPOverhead + ((l + 28) \cdot 8)/r = 192 + (((l + 28) \cdot 8)/r \quad (\mu s) \qquad (1)$$

The time for data frame transmission in our experiments is 488 μs. In IEEE 802.11b, a random back-off interval is introduced to reduce the probability of collisions. In our experiments, and using the equations from [5], minimum packet transmission delay in a contention-free environment, with a first back-off interval of 0, is 538μs The maximum random back-off interval, on the first transmission attempt, is 620μs. Thus, a successful first transmission attempt, i.e., with no collisions, may occur anywhere up to the upper bound of 1158μs, for our experiments. We use these theoretical time bounds to compare with our actual results.

3.2. Analysis of timed packet transmission

We start our evaluation by presenting some benchmark results, obtained in a low-contention, open-space environment with a packet transmission rate of 20ms. This periodic transmission rate is significantly greater than the expected round trip time (RTT), for the packet, i.e., the time for a transmitted packet to reach a receiver and a reply to reach the packet originator. Thus, this experiment exhibits a low probability of contention both in the environment and in the rate of packet transmission.

The average software latency achieved in this environment is illustrated in Figure 8. The average software latency, i.e., $T_{queued,}$ is 384 μs, with less than 0.4% of the packets transmitted in the sample space deviating from this average by more than 30μs. Thus, we consider 384μs, as the average software latency achievable using RT-WLAN. If the software latency is predictable, we expect our other experiments to exhibit approximately the same software latency, regardless of the contention in the physical environment or the rate of packet transmission.

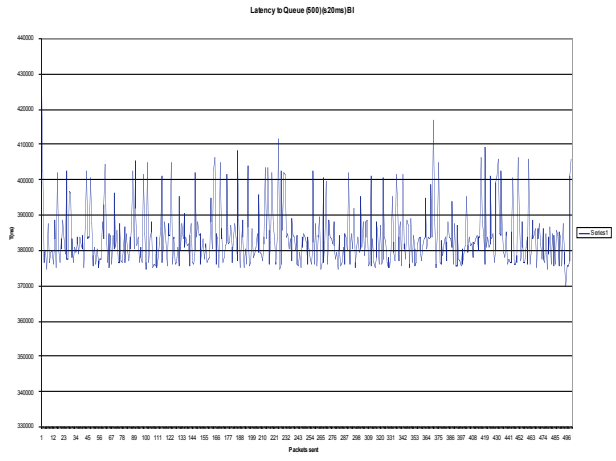

Figure 8: Benchmark software latency in RT-WLAN

The next stage of packet transmission is packet transfer to the physical wireless card and the latency incurred is attributed to the firmware. In this experiment the average firmware latency (i.e., the duration between the T_{alloc} interrupt and the time the packet was queued on the network device, or T_{alloc} - $T_{queued,}$ is 120μs. and less than 0.2% deviate from the average by +50μs. The results of this experiment are illustrated in Figure 9.

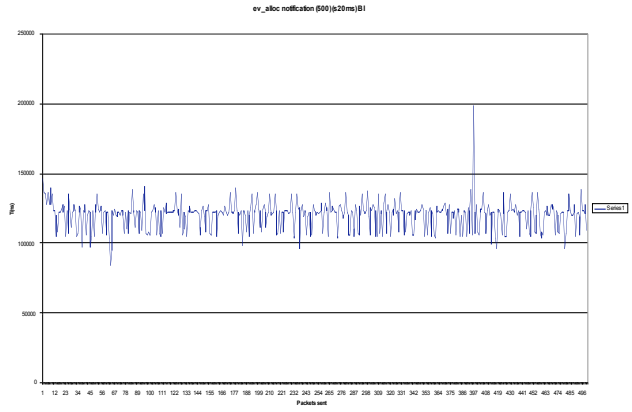

Figure 9: Firmware latency using RT-WLAN

Up to this stage, all packet transmission latencies have been incurred due to internal processing, to queue and then to transfer the packet to the wireless card. The next stage, packet transmission using IEEE 802.11b PHY, is the first point at which external factors influence the communication latency achievable, and is directly related to the utilisation of the wireless medium within the transmission range. A graph of the communication latency is illustrated in Figure 10.

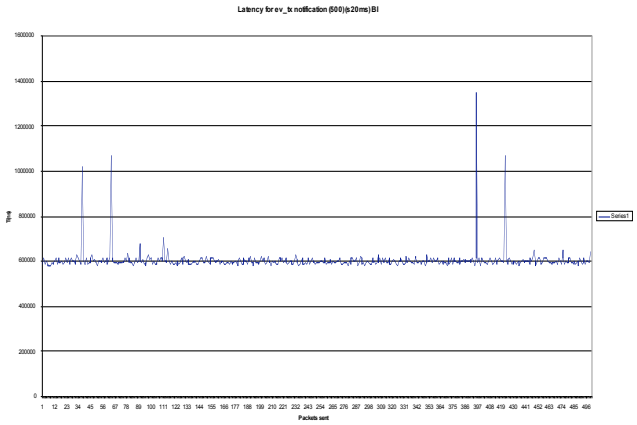

Figure 10: Communication latency for IEEE 802.11b

The average communication latency, i.e., the time for a packet queued in the wireless card to be successfully transmitted, is 604μs. As discussed previously, the time-bounds for successful packet transfer in our experiments range from 538μs to 1158μs. Using these values, it is assumed that those packets experiencing a communication latency greater than the upper bound were delayed, for example, due to contention for the wireless medium in the transmission range. In this experiment, 0.2% (i.e., 1 packet) experienced a delay greater than 1158μs, confirming the close to contention free status of our open space environment. In addition, 100% of the packets were transmitted successfully, and the average round trip time was measured as 2493μs.

The total send latency, $T_{trans\_total,}$ encapsulates the processing latency (software and firmware) and the communication latency (of IEEE 802.11 PHY). T_{trans_total} terminates when `RTorinoco_ev_tx` interrupt is received to notify that the packet has successfully left the wireless card. In this experiment, the average T_{trans_total} is 1108μs, with less than 0.8% of successfully transmitted packets deviating from this average by more than 300μs. It is interesting to note that the average achieved for *total send latency*, is still less than the upper bound for first attempt packet transmission communication latency, using IEEE 802.11b in a contention free environment. From these results it is clear that the random back-off interval is the dominant factor. It is observed that an "unlucky" choice of back-off interval (i.e., close to aCWmax[5],), will significantly increase the latency for packet transmission, possibly unnecessarily if contention is low. Given these results, an observation may be made that the dynamic adaptation of the bounds of the back-off interval to reflect the environment, e.g., gradually increasing the back-off interval if collisions are frequently occurring, would be beneficial for reducing communication latency in general. However, this

proposal would still not provide the predictability required for real-time communication.

To analyse the predictability achieved using RT-WLAN, and to further investigate the influence of the environment on firmware and communication latency, we performed our packet transmission experiment in each of our selected environments, at increasing rates of transmission, i.e., attempting to increase the probability of packet collisions. The results of these experiments are summarised in Table 1, for transmission rates of 5ms and 3ms. An interesting observation to be made from these results is that the worst-case software latency has remained within 6μs of the benchmark time-bound. Thus, predictable software latency is achievable regardless of the degree of contention in the physical environment.

Table 1: Average results for timed packet transmission at 5ms and 3ms

Latency to measure	***Average 5ms (BI)***	***Average 5ms (H)***	***Average 5ms (W)***
T_{queued}	387μs	387μs	388μs
T_{alloc}	122μs	121μs	121μs
T_{ev_tx}	647μs	644 μs	665μs
T_{trans_total}	1157μs	1151μs	1174μs
T_{RTT}	2595μs	2594μs	2612μs
Latency to measure	***Average 3ms (BI)***	***Average 3ms (H)***	***Average 3ms (W)***
T_{queued}	389μs	390μs	388μs
T_{alloc}	124μs	122μs	128μs
T_{ev_tx}	635μs	635 μs	654μs
T_{trans_total}	1149μs	1148μs	1170μs
T_{RTT}	2587μs	2600μs	2634μs

The result for communication latency, i.e., the T_{ev_tx} value, is also of interest. It is not surprising that the communication latency in our indoor, office environment with a transmission rate of 3ms, exhibits the greatest deviation from the average in the benchmark case, by approximately 55μs. What is not clear from the average result alone is that 2.5% of packet transmissions experienced a communication latency greater than 1158μs, exhibiting that wireless interference is evident in this environment. The worst-case communication latency in this indoor experiment was a substantial 2185μs, a value that includes multiple back-off and retransmission attempts to avoid contention for the wireless medium.

Wireless interference in the physical environment is one probable source of contention another is the high probability of forced contention and collision between the transmission of $packet_N$ and the reply of $packet_{N-1}$, or indeed any previous packet, at this high rate, i.e., 3ms, of packet transmission. Using the benchmark results obtained and referring to the calculation of the upper bound on successful first attempt packet transmission of 1158μs, it is clear that forced contention is occurring.

The average accumulated benchmark software and firmware latency, per transmission, is 504μs, i.e., (384 + 120)μs. Thus, the maximum successful first attempt total packet transfer latency, i.e., T_{trans_total}, could reach 1662μs, i.e., 1158μs + 504μs, for each transmitting wireless host. Thus, given the original send invocation was started at 0μs, the upper bound on the return packet reaching the originator, with successful first attempt transmission by both wireless hosts, is after 3324μs has elapsed, i.e., after the start of the next packet transfer at the originator, at 3000μs. In this case, packet transmission and reply have overlapped in time. Thus, if there is any wireless interference in the physical environment, evidence in our indoor office environment, causing additional back-off and retransmission attempts, the probability of original and reply packets being delayed and consequently overlapping in time and contending for the wireless medium, is even greater.

We target this behaviour explicitly in our next set of experiments. We increased the transmission rate even further, i.e., to 1.5ms, to observe the impact on wireless medium contention. The results achieved at 1.5ms transmission rate are illustrated in Table 2.

Table 2: Average results for timed packet transmissions at 1.5ms

Latency to measure	***Average 1.5ms (BI)***	***Average 1.5ms (H)***	***Average 1.5ms (W)***
T_{queued}	388μs	388μs	388μs
T_{ev_tx}	900μs	897 μs	940μs
T_{trans_total}	1148μs	1144μs	1183μs
T_{RTT}	3096μs	3107μs	3132μs
% dropped	82	82	83

Again, an initial observation is the predictability of the results for the software latency, i.e., within 4μs of the benchmark results, highlighting that increased transmission rates do not influence the real-time behaviour of the software latency.

The most obvious influence of the change of transmission rate is, unsurprisingly, on the communication latency T_{ev_tx}, which has increased by approximately 300μs on average, with a worst-case increase of 340μs, again as exhibited in the indoor, office environment. The results for T_{trans_total}, are also significant, with an average in the three environments of 1157μs. Given the periodic transmission rate of 1500μs in these experiments, it is obvious that packet transmission and reply are highly likely to overlap in time, increasing the contention for the wireless

medium, potentially for every reply packet, leading to random back-off, transmission retry attempts and finally dropped packets. As illustrated in Table 2, the forced contention in this experiment has lead to a startling rate of packets dropped, with 82% of packets dropped even in the open space experiments. Given the rate of dropped packets deviates by only 1%, throughout the different physical environments, it is clear that the forced contention for the wireless medium introduced at this rate of transmission, is the primary cause rather than external environmental influence.

In the 3ms experiments, although wireless contention lead to increased communication latency, no packets were dropped, with 100% successful packet transmissions achieved. Thus, between a 3ms and 1.5ms transmission rate, the time-bounds of the wireless device, i.e., the firmware latency, and the semantics of contention-based IEEE 802.11 medium-access, e.g., the random back-off intervals, play an increasingly critical role on the predictability of packet transmission, with the result that any contention for the wireless medium, (regardless of the physical environment) may contribute to packet loss.

The objective of our final set of experiments was to analyze the impact of an additional participating wireless host, with identical software and hardware, in the shared transmission range. As per the previous experiments, one wireless host was designated as the packet transmission originator, with the other two receiving, processing and replying with an amended packet. The experiments complete when the originator has processed the returned packet. An experimental run encompasses 100 packet transmissions from the originator.

The results obtained, for each receiving wireless host, are illustrated in Table 3, and relate to a 20ms transmission rate in our indoor, home environment.

Table 3: Average results for multi-node packet transmission at 20ms

Latency to measure	***Average HostA***	***Average HostB***
T_{queued}	393μs	397μs
T_{ev_tx}	573μs	579μs
T_{trans_total}	1198μs	1211μs
T_{RTT}	2825μs	2797μs
% dropped	4	76

Analyzing these results the predictability and scalability of the software latency is again evident. The most significant result is the high percentage of dropped packets, particularly in relation to HostB. Our future work will investigate further this bias towards one specific host, whereas our current analysis discusses the general cause for high packet loss in this experiment.

Our multi-node experiments were executed at a 20ms packet transmission rate, which as identified previously, is significantly greater than the expected worst-case round trip communication time. At a 20ms transmission rate we do not expect forced contention between packet transmission and reply, as described in our 1.5ms experiments.

The source of packet loss in this experiment is directly related to the inclusion of an additional participating wireless host in the transmission range. A typical scenario is as follows: the originator transmits a packet every 20ms. Both receiving wireless hosts receive, process and attempt to reply, with a high probability of directly contending for the wireless medium. Following the maximum number of retry attempts has been reached, the packet is dropped. Thus, in this experiment packet loss is due solely to the contention-based medium-access wireless ad hoc protocol of IEEE 802.11.

Observing the results achieved, the addition of one wireless host has caused non-deterministic packet transfer and significant packet loss, highlighting the scalability issues prevalent with any contention-based MAC protocol. From these results alone it is clear that the contention-based IEEE 802.11 MAC protocol is not suitable for predictable, and therefore real-time, communication in wireless ad hoc networks, even of limited density.

The objectives of our experiments were to evaluate the predictability of the RT-WLAN network subsystem and provide low-level timing analysis of each of the stages of packet transmission. From the granularity of the timing analysis, it is clear that predictable software latency, regardless of the physical environment and packet transmission rate, has been achieved. What has also become evident is the extent to which the physical environment, the utilization of the wireless medium and the network density, influences the communication latency, directly related to the contention-based IEEE 802.11 MAC protocol.

Removing unpredictable wireless contention from packet transmission, i.e., a real-time medium-access protocol, would provide the predictability required for real-time wireless communication. We discuss our current and future work on the implementation of a real-time MAC protocol in the next section.

4. Future work

Predictable medium-access implies contention for the medium is removed. Our medium access control protocol, TBMAC [16], provides, with high probability, time bounded access to the wireless medium to mobile hosts in a multi-hop ad hoc network. Our current and future work is to implement TBMAC as a layer above IEEE 802.11 using our real-time network subsystem, to provide time-bounded wireless communication.

The TBMAC protocol is based on time division multiple access with dynamic but predictable slot allocation. To reduce the probability of contention between mobile hosts, the geographical area occupied by the mobile hosts is statically divided into a number of cells in a similar approach to [6]. Each cell is allocated a particular radio channel to use. Each mobile host is required to know its location (using GPS) and from this which cell it is in and what radio channel to use to communicate with other mobile hosts in the cell.

Similar to the IEEE 802.11 standard, the TBMAC protocol divides access to the wireless medium within a cell into two distinct time periods:

- Contention Free Period (CFP)
- Contention Period (CP)

Both the CFP and the CP are divided into a set of slots. A CFP followed by a CP constitute a round of the TBMAC protocol. Dividing access to the medium into these two well-known time periods requires the clocks of all the mobile hosts in the network to be synchronized (e.g., each host using a GPS receiver [7]).The critical point for real-time communication is that once a mobile host has been allocated a CFP slot, it has predictable access to the wireless medium until it leaves the cell or fails. Mobile hosts that do not have CFP slots allocated to them contend (similarly to IEEE 802.11) with each other to request CFP slots to be allocated to them in the CP.

Further discussion of the operational modes of TBMAC, i.e., the impact of empty or non-empty cells, and the derivation of the time-bound and associated probability of slot allocation are beyond the scope of this paper, but are available in [8].

The TBMAC protocol has been implemented by extensive simulation using ns2[9]. However, a real-world implementation has never been possible, given the reliance on the unpredictable standard Linux network subsystem. A first step towards a real-world implementation of TBMAC is a predictable real-time network subsystem, which we now have available using RT-WLAN. The next step is to provide real-time wireless medium-access by implementing a real-world version of the TBMAC protocol, using RT-WLAN, as a layer about IEEE 802.11

5. Conclusion

Real-time wireless communication requires predictable and guaranteed latency at all phases of wireless packet transmission. In this paper we discussed our design, implementation and evaluation of RT-WLAN, our real-time Linux network subsystem. Furthermore, using RT-WLAN we presented timed packet transmission, the granularity of which has not been previously available. Future work includes implementing a real-time MAC layer above IEEE 802.11, to provide predictable access to the wireless medium, essential for real-time communication.

Acknowledgements. The work described in this paper was partly supported by the FET programme of the Commission of the European Union under research contract IST-2000-26031 (CORTEX).

References

[1] J. Yee, ""The 802.11g reality drives 802.11a's future dominance"," Portable Design, 2004.

[2] R. Cunningham and V. Cahill, ""Time bounded Medium Access Control for Ad Hoc Networks"," presented at Principles of Mobile Computing (POMC'2002), Toulouse, France, October 30-31, 2002.

[3] P. Mantegazza, E. Bianchi, M. Angelo, D. Beal, and K. Yaghmour, ""DIAPM RTAI Programming Guide 1.0"," September, 2000.

[4] A. Rubini and J. Corbet, *"Linux Device Drivers"*, 2nd ed: O'Reilly & Associates, June, 2001.

[5] A. Jain, D. Qiao, and K. G. Shin, ""RT-WLAN: A Soft Real-Time Extension to the ORINOCO Linux Device Driver"," presented at 14th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC2003), Beijing, China, September 7-10, 2003.

[6] A. Konstantinos, ""Space-Time Division Multiple Access (STDMA) and Coordinated Power-Aware MACA for Mobile Ad Hoc Networks"," presented at IEEE Symposium on Ad Hoc Wireless Networks (SWAN01), San Antonio, Texas., 2001.

[7] P. Verissimo, L. Rodrigues, and A. Casimiro, ""Cesiumspray : a precise and accurate global time service for large-scale systems"," *Journal of Real-Time Systems*, vol. 12, pp. 243-294, May, 1997.

[8] R. Cunningham, ""Time Bounded Medium Access Control for Ad-Hoc Networks"," PhD. Thesis, University of Dublin, Trinity College., October, 2003.

[9] K. Fall and K. Varadhan, *"The ns manual"*: The VINT Project, UCB, LBL,USC/ISI and Xerox PARC, http://www.isi.edu/nsnam/ns/doc, April, 2002.