RTLinux Ethernet Device Drivers

Florian Bruckner
OpenTech EDV-Research GmbH [1]
Lichtenstein Str 31, 2130 Mistelbach, AUSTRIA
florian.bruckner@aon.at

Abstract

This paper is about the current development of REDD, the Realtime Ethernet Device Drivers. First of all the previous state of the original REDD driver will be described, which was the base for our work. Then we will explain the most important changes we did during our development, like a common POSIX interface, multidevice support, `/proc` interface, and the error handling mechanism. The next step will be to introduce some examples that have been developed concurrently with the drivers. Our intention is to offer some easy to use tools to test the drivers functionality, like a ping request/reply or a simple benchmark tool. The last part will deal with future development of the driver itself, and of cause with the possibilities it offers. Examples are global time, redundant links, distributed shared memory.

1 Introduction

1.1 Brief History

REDD is a project started in 2002 and maintained by Sergio Prez Alcaiz [2]. The initial development of real-time ethernet for RTLinux/GPL was derived from the lw-IP [3] project with drivers merged from etherboot [4] and was released as RTL-LW-IP in the framework of the OCERA project [5]. Aside from the etherboot drivers not targeting real-time and there being some serious problems with lw-IP, it seemed preferable to depend future development on the main stream Linux kernel drivers. One of the main problems with lw-IP was its memory management, which was resolved in REDD by utilizing TLSF [6]. Secondly the code-base for lw-IP was large, most of which was not suitable for RT-applications, and thus neither well maintainable nor well integratable into the RTLinux/GPL effort. Departing from the shortcommings of lw-IP required a complete reimplementation which was REDD.

1.2 What is REDD

REDD consists of a set of ethernet device drivers suitable for RTLinux and a common lowlevel interface to access them in an RT-safe way. Currently rtl8139, e100 and 3c59x cards are supported and a VIA Rhine drive is being developed. The drivers are originally based on the standard Linux drivers, modified to fit the RT-constraints, i.e. some non-deterministic performance-improvement features are turned off in order to guarantee realtime behaviour. Examples of such features are

- early interrupts - which are raised before the whole packet is actually transfered to the input buffer.
- multiple events handled per interrupt - Modern NICs also allow to handle more than one packet within one interrupt handler, which would dramatically decrease realtime performance.

Another important difference to Linux drivers is error handling. Realtime applications cannot easily retransmit a packet without introducing inaceptably high worst case latency or ignore a link-state-change. Thus there must be a possibility to detect these errors in a timely manner and allow a clean shutdown of the system to reach a safe state. In order to make ethernet realtime capable there are also some restrictions imposed on the topology of the network. The simplest case, preventing collisions, is to use point-to-point connections only - which can be achived by the use of cross-over cables or hubs. Another solution would be to use a Time Triggert approach in a multinode network. A variant of which, namely TDM, is implemented in RT-Net [7].

2 Current State of Development

Currently a layered module set is used, offering a common interface via the `redd.o` module to the RT-application layer based on a POSIX open/read/write/etc calls. `redd.o` registers an REDD major device, which allows access via standard device mechanisms. Additionally it offers a register function for the device drivers. The device driver has to offer some common access functions like a send, receive function. During the registration of the REDD device a device structure is created. The minor number of the device allows to distinguish all REDD devices.

This design allows the usage of multiple devices including a loopback device to test your applications. Currently loopback only supports point-to-point connections, but it is planned to implement a switch/hub mechanism to simulate a virtual network. It should be possible to configure a logical network as it will be used in real world (e.g. 3 nodes connected by a hub). Of course the timing of the real network cannot be simulated, but it can be very helpful for testing. Additionally there should be an error injection mechanism to simulate some hardware errors like link-change, wrong checksums, etc . This should allow better testing of your application under some special error conditions which would otherwise be difficult to create and support device validation efforts similar to netemu.

Another possibility that this design offers is to implement drivers that need access to more than one NIC. Examples for such driver would be redundent links, bonding drivers, routing threads, etc.

3 Problems during development

One problem that surfaced during the development of REDD is the error handling strategy. Nearly all errors that can be detected by the NIC are serious errors for realtime applications and will lead to a shutdown (in case of single link systems). Exceptions are errors that can be recovered from within the driver code (e.g. if a transmit FIFO underrun occurs, the packet can be correctly transfered but then the tx-threshold is set to a higher level) or from within the protocol layer.

The main problem are the class of connection independant or general errors that do not belong to a specfic packet/connection and thus show a bad detectability (i.e. won't be noticed until the device requests a transmission or receive), usage of heart-beat mechanisms can resolve these issues with an acceptable overhead. Additionally it is problematic that during the, de-facto mandatory non-blocking send procedure, error information becomes available only when that packet has been sent by the NIC and not when the write call returns, thus delaying recovery or safe shutdown. The current implementations uses a global (per device) errno variable that stores the current error values. Every POSIX call checks this variable and returns the errors code if something went wrong. The error may not correspond with the actual packet. One possibility to directly read the error values would be to use IOCTLs.

4 Examples and benchmarks

During the development of the driver we also created some example applications for testing reasons. First example was a simple ping reply and ping request. The ping request is able to create a ARP reply for address resolution and if it gets an ICMP request it simple swaps sender and receiver address and changes the ICMP code to ICMP-reply. The ping request first hast to send an ARP request and then creates ICMP-request packets of different sizes. Additionally the roundtrip time of the packet is measured.

Another example is the test_bandwidth example that can be used to measure the roundtrip time as well as the throughput of packets of different size. It simply sends packets with a sequence number and a certain size to the receiver. For one measurement `NUM_ITERATIONS` packets are sent with the sequence number incremented after each packet. If the receiver get a packet with sequence number equal zero is replies an acknowledge packet. The receiver also checks that no sequence number is missing. The sender repeats this procedure `NUM_MEASUREMENTS` times and always takes timestamps before the first send and after the reception of the acknowledge. If `NUM_ITERATIONS` is set to 1 the difference between these timestamps is the roundtrip time of the packet, if it is set to a very high value you get the inverse of the maximum throughput.

First benchmarks with two equivalent 8139too 100Mbit/s NICS connected by a crossover cable show that the roundtrip time start at 50us for small packets (60 Bytes) and rises up to 200us for MTU-sized packets (1500 Bytes). The throughput measurement shows a nearly linear connection between packet size and time. For MTU-sized packets it takes about 120us with gives us a bandwidth of about 95 MBits/s which is a very good values. These a values of without any systemload. If the system running lmbench or netpipe during the benchmarks the roundtrip times got worse value but the throughput values almost stayed the same. A big problem was that lmbench and netpipe do not create the same load over time, which makes it hard to find some real worst case times for each packet size.

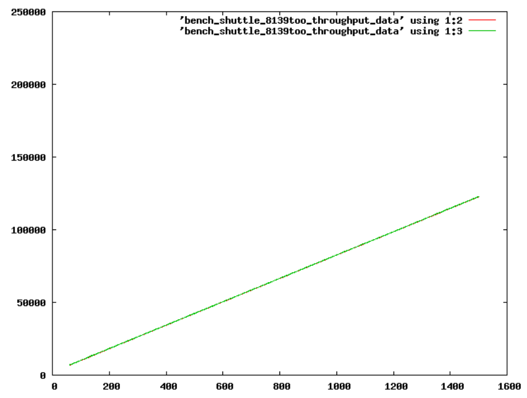

FIGURE 1: *Throughput*

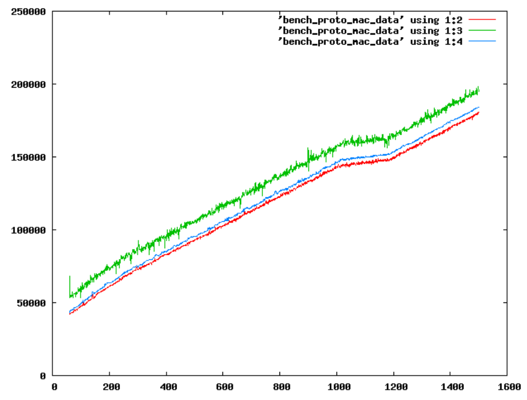

FIGURE 2: *Round-Trip Time*

5 Conclusion and possibilities for future development

Currently REDD offers a low level interface to some ethernet cards. It uses a read/write interface instead of a heavy socket interface. This keeps the interface overhead at the minimum but leads to some restrictions in usage. In contrast to a general purpose network stack it is not possible to layer many different protocols on top of this interface. But for realtime behaviour this would not be a good idea anyway. The REDD interface will allow different protocols but the access to the NIC will be restricted to one protocol at a time.

Examples for such protocols are distributed shared memory implementations, global time, time triggered protocols, MPI RT, and so on. The next steps in REDD development will be to automatically add a MAC header with the local source address and a multicast destination address. Additionally an interface should be added that allows to bind several physical links to one logical link and provide redundancy.

References

[1] Opentech, *OpenTech - RTLinux Support in Europe*, 2006-09-19, `http://www.opentech.at/`

[2] Sourceforge, *REDD: RTLinux Ethernet Device Drivers*, 2006-09-19, `http://redd.sourceforge.net/`

[3] Sourceforge, *RTLinux Lightweight TCP/IP Stack*, 2006-09-19, `http://rtl-lwip.sourceforge.net/`

[4] Sourceforge, *EtherBoot/GPXE*, 2006-09-19, `http://etherboot.sourceforge.net/`

[5] Ocera, *Open Components for Embedded Real-time Applications*, 2006-09-19, `http://www.ocera.org/`

[6] TLSF, *Real-Time Dynamic Memory Allocation*, 2006-09-19, `http://rtportal.upv.es/rtmalloc/`

[7] RTnet, *Hard Real-Time Networking for Real-Time Linux*, 2006-09-19, `http://www.rtnet.org/`