IST-2001-37652

Hard Real-time CORBA

Title

D3.6
Robot Control Testbed
Testing

Authors

Klas Nilsson (klas@cs.lth.se)
Mathias Haage (mathias@cs.lth.se)

Reference IST37652/089 Deliverable D3.6
Date 2003-10-23
Release 1.0
Status Final
Clearance Consortium

Partners

Universidad Politécnica de Madrid
Lunds Tekniska Högskola
Technische Universität Wien
SCILabs Ingenieros

Summary Sheet

IST Project 2001-37652
HRTC
Hard Real-time CORBA

D 3.6 Robot Control Testbed
Testing

Abstract:

The developed Robot Control Testbed (RCT) for Hard Real-Time CORBA (HRT CORBA) has been used to test communication techniques in a CORBA context. Tests using non-RT communication (TCP/IP) show timing variations and worst-case latencies that are too big for motion control, such as the control of a robotic arm. Testing is carried out both in separate setups of specific communication channels within the RCT, and also as full-scale test of the robot arm control.

Testing of the proposed hybrid approach with a HRT-communication aware ORB on the Linux user level, in combination with HRT-parts of the application and communication running in Linux kernel space, is done using both TCP/IP and ThrottleNet (TN) transports. Then TCP/IP is used from user space and is tunnelled over TN which works fine together with the HRT TN traffic. Using TN from the user level is tested and can be useful for implementation of HRT CORBA on a predictable RTOS, but the scheduling of user-space threads in Linux prevents predictability. Running TN from the HRT part of the application (in kernel space using RTAI) shows that standard fast switched Ethernet gives the desired predictability and performance for complete robot motion control, thus verifying the usefulness of HRT CORBA.

HRTC Partners:

Universidad Politécnica de Madrid
Lunds Tekniska Högskola
Technische Universität Wien
SCILabs Ingenieros.

Release Sheet (1)

Release:	**0.1 Draft**
Date:	2003-09-28
Scope	Initial version
Sheets	All

Release:	**1.0 Final**
Date:	2003-10-23
Scope	Final version
Sheets	All

Table of Contents

1 Introduction

This document describes tests performed on the testbed hardware and reports measurements and results derived from these tests. It does not document the testbed (D3.7), the hard realtime implementation (D3.5) and the soft realtime implementation (D3.4). Five different tests have been performed; IIOP stresstests showing minimum roundtrip latency and jitter using IIOP between two nodes using an intermediate switch. TN latency and jitter test for specific periodicity between two nodes. TNIOP test between two nodes demonstrating GIOP communication between two nodes using an intermediate switch. Emulated TCP/IP performance using emulated network TCP/IP delay on testbed system. The testbed experiment is run with emulated TCP/IP and emulated TN performance and with TN.

2 Hardware setup

The RCT as such is described in design, implementation, and documentation documents. The processors that are referred to, however, deserve some more specific performance data for the interpretation of the test results in the following.

Four types of processors, all running Linux if nothing else stated, are used in the tests. The properties are outlined in Figure 1. Note that in the test results, the hardware is referred to by the compiler version. That is,

- Cris refers to the Axis ETRAX port of the GNU C complier.
- PPC refers to the PowerPC (G3 on VME).
- i386 refers to the Intel family, here actually a i686 (Pentium-4).

The rest of the setup is according to the RCT implementation.

i686	processor : 0 vendor_id : GenuineIntel cpu family : 15 model : 2 model name : Intel(R) Pentium(R) 4 CPU 2.60GHz stepping : 9 cpu MHz : 2593.562 cache size : 512 KB fdiv_bug : no hlt_bug : no f00f_bug : no coma_bug : no fpu : yes fpu_exception : yes cpuid level : 2 wp : yes flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm bogomips : 5177.34
PPC	cpu : 604r clock : ??? revision : 49.2 (pvr 0009 3102) bogomips : 332.59 machine : PReP MVME 2600/2700 with MVME761 L2 : none
Cris	processor : 0 cpu : CRIS cpu revision : 11 cpu model : ETRAX 100LX v2 cache size : 8 kB fpu : no mmu : yes mmu DMA bug : no ethernet : 10/100 Mbps token ring : no scsi : yes ata : yes usb : yes bogomips : 99.73

Figure 1. /proc/cpuinfo from the different hardware platforms used in the testbed.

3 IIOP stress-test

The purpose of the IIOP stress-test is to determine the minimum CORBA call latency of the standard CORBA network protocol on the different hardware platforms utilized in the testbed. This is done to measure the performance of the ORB used in the testbed and to be able to compare performance with the TN solution.

The test is performed by physically isolating two computer nodes on the network, only allowing them to communicate between each other through a switch. The hardware used is documented in Figure 1. Application end-to-end delay is measured by executing a number of synchronous method calls measuring time spent in the call. See Figures 2 to 4 for the experiment code.

It should be noted that this test measures both the performance of the TCP/IP-stack and the ORB residing on top of it. It does not give a clear picture on time spent in the ORB alone.

```
module Test{
 interface tester_servant {
   long a(in long val);
 };
};
```

Figure 2. IDL used for measuring application round-trip delay.

```
    long seq;
    long res;
    struct timeval tv;
    for (seq=0; seq<log_size; seq++) {
      res = gettimeofday(&tv, NULL);
      memcpy(&(log_tv[seq]), &tv, sizeof(struct timeval));

      res = tsv->a(seq);
    }
```

Figure 3. Experiment code executed on the client machine for measuring round-trip application delay. The “a” method is called repeatable with IIOP used as transport. Before each call time is read and a timestamp is stored. The “gettimeofday” call has a resolution according to Figure 5.

```
class tester_impl: public virtual
POA_Test::tester_servant{

public:

  long a(long val)ICA_THROW_DECL(CORBA::SystemException){
    return val;
  }
};
```

Figure 4. Experiment code executed on the server machine for measuring round-trip application delay. IIOP is used as transport.

Type	Gettimeofday resolution (microseconds)
Intel i386	1.0
MVME PPC	2.3
ETrax CRIS	32

Figure 5. Measured gettimeofday resolution on the different hardware platforms used in the testbed.

Also note that since the test objects involved are executed on the user-level, the spikes that can be seen in Figure 6 may be due to system calls or other services delaying the execution. Therefore, hard-RT needs both hard-RT communication (TN in our case) and kernel-space execution (RTAI in our case).

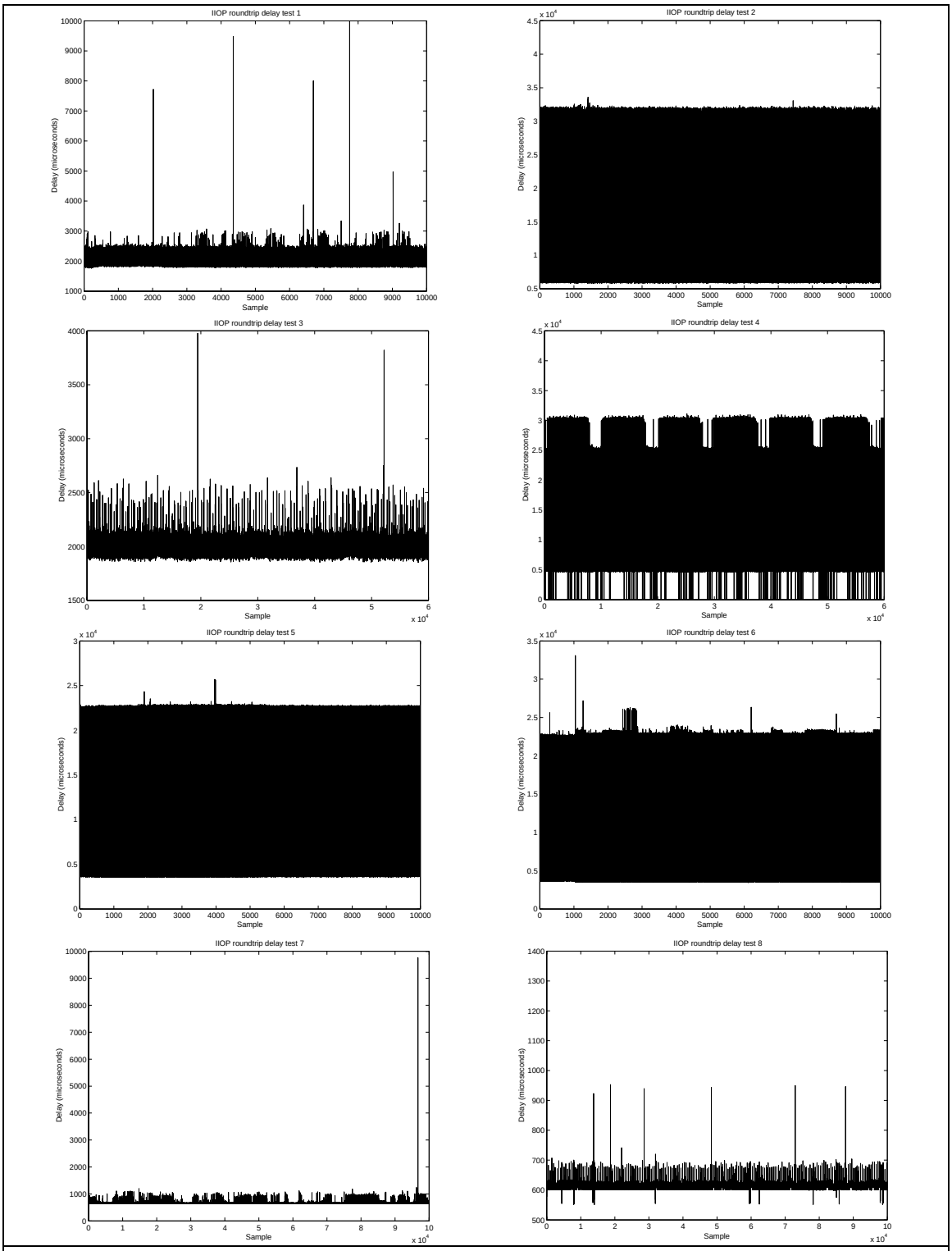

Figure 6. Round-trip IIOP performance measurements for test 1 to 8 as summarised in Figure 8. Note that the black areas such as in Figure 2 (upper right) contain very frequent delay variations, which due to the number of

samples fills the area.

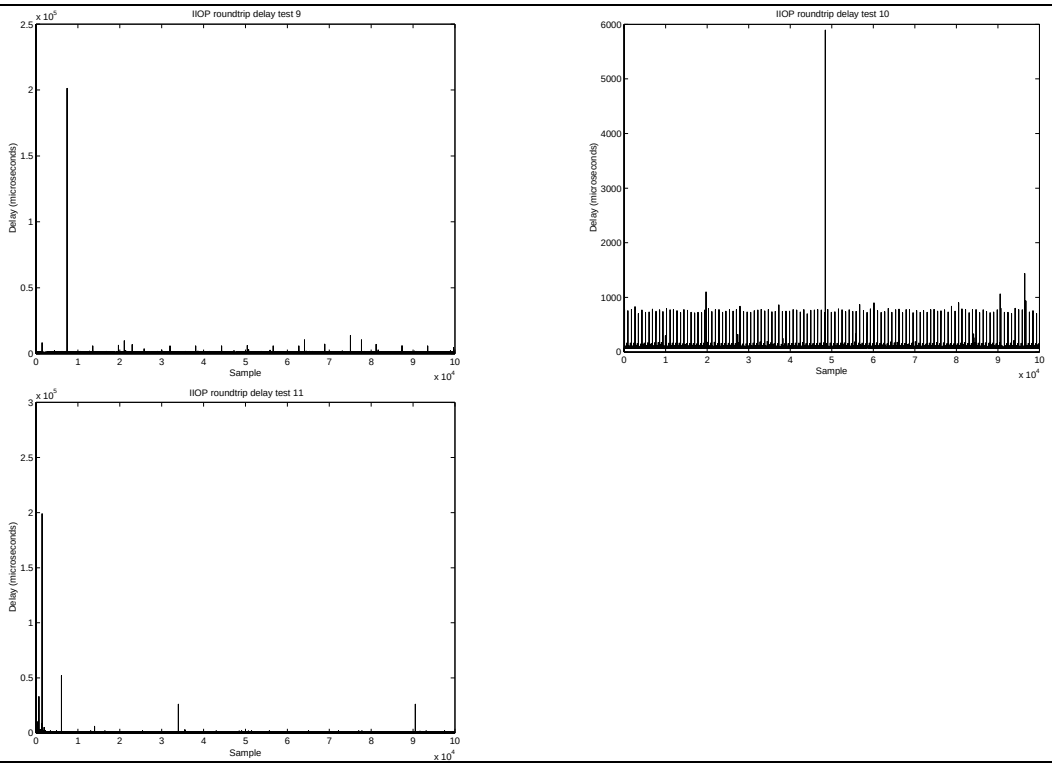

Figure 7. Round-trip IIOP performance measurements for tests 9 to 11. Test 11 is using a Java implementation of the test IDL for comparison purposes.

Server / Client	Cris 1	PPC	i386
Cris 2	test 4 samp 60000 mean 12323 std 11186 max 45000 min 26		
Cris 1	test 2 samp 10000 mean 12501 std 11042 max 40490 min 5774	test 3 samp 60000 mean 1969.6 std 59.7 max 3977 min 1852	test 1 samp 10000 mean 2143.6 std 316.6 max 9995 min 1764
PPC	test 5 samp 10000 mean 10002 std 8943.9 max 28948 min 3513	test 8 samp 100000 mean 608.6 std 7.7 max 1301 min 551	Test 9 samp 100000 mean 814.8 std 713.5 max 201150 min 494
i386	test 6 samp 10000 mean 10078 std 9091 max 33091 min 3452	test 7 samp 100000 mean 9764 std 35.8 max 9764 min 630	test 10 samp 100000 mean 70.8 std 31.3 max 5890 min 62

Figure 8. IIOP roundtrip delay test summary. Average, standard deviation, maximum and minimum values for the tests are given (all measurements in microseconds). Note the execution time asymmetry between client and server. The high max values are often due to high initial delays or spikes.

As can be seen from the tests, IIOP is not suitable for feedback/servo control. That also applies to the Java-based measurement (partly motivated by the on-line connection to the virtual testbed) as shown in Figure 9.

Server / Client	i686
i686	test 11 samp 100000 mean 374 std 1251 max 259000 min 0

Figure 9. IIOP roundtrip delay test summary for the Java comparison test. Average, standard deviation, maximum and minimum values for the tests are given (all measurements in microseconds). The high max value is due to high initial delay caused by Java.

4 TN latency and jitter test

Using TN in kernel space (RTAI) is the test case corresponding to the communication in the servo control of the robot. Compared to other tests on our legacy system [Mart2002], the Linux drivers appear to be less optimal for hart-RT execution (costing some 20 to 30 microseconds extra delay per transfer). The TN kernel-space test program is listed in Figure 10.

```
/***
 *
 *  throttlenet_test_3_server.c
 *
 *  Copyright (C) 2003 Anders Blomdell <anders.blomdell@control.lth.se>
 *
 *  This program is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation; either version 2 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program; if not, write to the Free Software
 *  Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 *
 */

#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <rtai.h>
#include <rtai_sched.h>
#include <rtai_fifos.h>
#include <throttlenet.h>
```

```
MODULE_LICENSE("GPL");

static throttlenet_receive_socket server;
static throttlenet_transmit_socket to_client;

static RT_TASK echo_task;


static void echo(int arg)
{
  server = throttlenet_open_receive_socket("test_3_server", 40, 10);
  if (server) {
    int receive_count, transmit_count;
    char data[1500];

    receive_count = throttlenet_receive(server, data, sizeof(data));
    to_client = throttlenet_open_transmit_socket("test_3_client",
                                        1500, 100);
    if (to_client >= 0) {
      while (1) {
       transmit_count = throttlenet_transmit(to_client, data, receive_count);
       receive_count = throttlenet_receive(server, data, sizeof(data));
      }
    }
  } else {
    printk("Could not create server socket\n");
  }
}

static int test_3_server_init(void)
{
  rt_task_init(&echo_task, echo,0,40960,10,0,NULL);
  rt_task_resume(&echo_task);
  return 0;
}

static void test_3_server_exit(void)
{
  rt_task_delete(&echo_task);
}

module_init(test_3_server_init);
module_exit(test_3_server_exit);
```

Figure 10. RTAI server implementation of TN test.

```
/***
 *
 *  throttlenet_test_3_client.c
 *
 *  Copyright (C) 2003 Anders Blomdell <anders.blomdell@control.lth.se>
 *
 *  This program is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation; either version 2 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
```

```
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program; if not, write to the Free Software
 *  Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 *
 */

#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <rtai.h>
#include <rtai_sched.h>
#include <rtai_fifos.h>
#include <throttlenet.h>
#include <linux/proc_fs.h>
#include <rtai_proc_fs.h>

MODULE_LICENSE("GPL");

#define BUF_SIZE 10000
#define DELAY_USEC 250LL


static throttlenet_receive_socket client;
static throttlenet_transmit_socket to_server;

static RT_TASK do_experiment_task;

typedef struct {
  int index;
} timing_packet_t;


static RTIME total_runtime;
static RTIME diff_buf[BUF_SIZE];
static RTIME abs_buf[BUF_SIZE];

static int test3_read_proc(char *page, char **start, off_t off,
                           int count, int *eof, void *data)
{
  PROC_PRINT_VARS;
  int i;

  PROC_PRINT("%c samples: %d\n", '%', BUF_SIZE);
  PROC_PRINT("%c total runtime (nanosec): %lld\n", '%', total_runtime);

  for (i = 0 ; i < BUF_SIZE ; i++) {
    PROC_PRINT("%lld %lld\n", abs_buf[i], diff_buf[i]);
  }
  PROC_PRINT_DONE;
}

static void do_experiment(int arg)
{
  client = throttlenet_open_receive_socket("test_3_client", 150, DELAY_USEC);
  if (client) {
    to_server = throttlenet_open_transmit_socket("test_3_server",
                                                 150, DELAY_USEC);
    rt_sleep(nano2count(1000000000LL));
    if (to_server) {
      int i;
```

```
      RTIME start;
      RTIME end;
      RTIME tic;
      RTIME tac;

      printk("Starting experiment\n");
      start = rt_get_time_ns();
      for (i = 0 ; i < BUF_SIZE ; i++) {
       RTIME diff;
       int receive_count;
       timing_packet_t data;

       // Send
       data.index = i;
       tic = rt_get_time_ns();
       throttlenet_transmit(to_server, (void*)&data, sizeof(data));

       // Receive
       receive_count = throttlenet_receive(client, (void*)&data,
                                           sizeof(data));
       tac = rt_get_time_ns();

       // Elapsed time
       diff = tac - tic;
       diff_buf[i] = diff;
       diff = tic - start;
       abs_buf[i] = diff;

       // Sleep
       rt_sleep(nano2count(DELAY_USEC*1000LL));
      }
      end = rt_get_time_ns();
      total_runtime = end - start;
      printk("Finished experiment\n");
    }
  }
}

static int test_3_init(void)
{
  static struct proc_dir_entry *proc_test;
  proc_test = create_proc_entry("test_3",
                                S_IFREG|S_IRUGO|S_IWUSR,
                                0);
  if (proc_test) { proc_test->read_proc = test3_read_proc; }
  // Timers should already be started by the throttlenet layer!
  rt_task_init(&do_experiment_task, do_experiment, 0,40960,10,0,NULL);
  rt_task_resume(&do_experiment_task);

  return 0;
}

static void test_3_exit(void)
{
  remove_proc_entry("test_3", 0);
  rt_task_delete(&do_experiment_task);
}

module_init(test_3_init);
module_exit(test_3_exit);
```

Figure 11. RTAI client implementation of TN test. Displayed test is tuned for 4kHz.

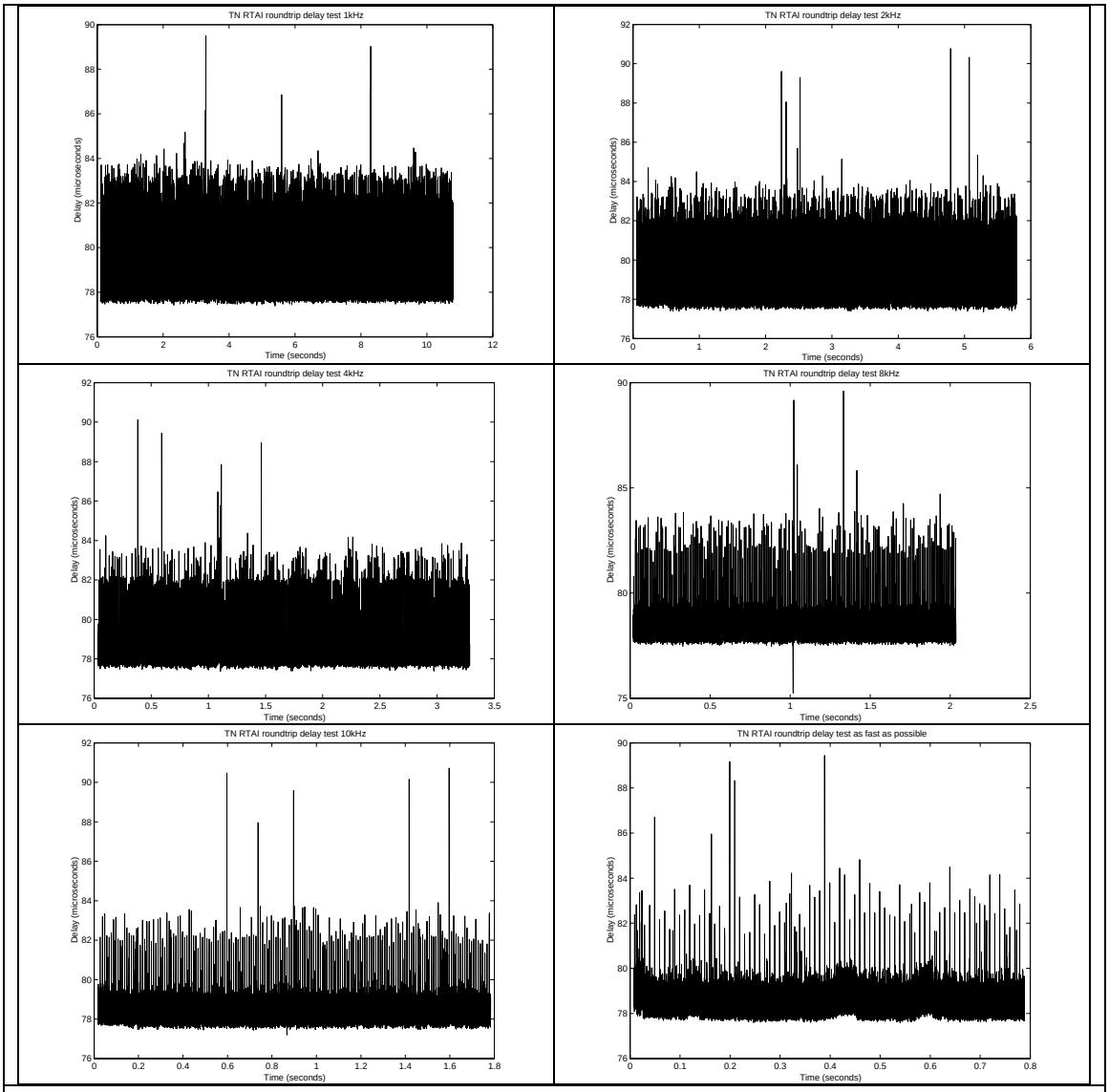

Figure 12. TN roundtrip performance measurements. Spikes are of much lower magnitude than in the IIOP case (microseconds instead of milliseconds).

Client/Server Frequency	i386/PPC
1 kHz	samp 10000 mean 78.9 std 1.5 max 89.5 min 77.4

2 kHz	samp 10000 mean 78.7 std 1.3 max 90.8 min 77.4
4 kHz	samp 10000 mean 78.6 std 1.1 max 90.1 min 77.4
8 kHz	samp 10000 mean 78.6 std 0.9 max 89.6 min 75.3
10 kHz	samp 10000 mean 78.4 std 0.9 max 90.7 min 77.2
As fast as possible	samp 10000 mean 78.6 std 0.8 max 89.4 min 77.6

Figure 13. Performance of TN from kernel space (hard-RT execution), showing desired performance and very low jitter.

5 Videos

Videos showing the effect of network delays on servo control as well as the ball-catching experiment are available on http://www.robot.lth.se/proj/hrtc, where also expanded versions of the main figures above showing the measured network delays are available. Videos should be played with sound since bad performance is mainly perceived through audio. Vibrations caused by network delays are typically of high frequencies of the audible spectrum. Vibrations are hard to see on the video images, but such a vibrating/noisy robot is industrially useless.

As shown in the networkdelay.mpg video, servo control using TCP/IP does not work, but with fractions of the TCP/IP the robot moves but is quite shaky. However, if the robot is started with RT communication and then IIOP delays are introduced, the robot moves for a while (but fails to catch the thrown ball) but then crashes as shown in the tcpcatch experiment. The normal reason for

the robot to stop when the communication gets to bad is that the drive electronics detects too jerky current references (resulting from the deficient timing), but in the tcpcatch.mpg video it was the mechanical vibrations that caused too much stress on the measurement system. That is, the vibrations due to the network delays caused the resolver measurement hardware for joint 2 to be partly disconnected, thereby giving wrong feedback from the joint angle.

In the tncatch.mpg video the robot moves as it should from a servo point of view, using TN communication on the servo level. In the outer visual control loop, however, IIOP (TCP/IP timing) is used. Together with the fact that the cameras are not synchronized (due to FireWire and camera properties) and the camera interface is running on Windows-2000, the result is that the visual control is not impressive. This is according to the aim and design of the RCT, but clearly the use of HRT CORBA/communication for the entire control would give better control and a robot catching every (?) thrown ball. Inspired by the accomplished properties of the RCT and the TN protocol, work towards such a robot system will continue after the completion of the current HRTC project. Note that due to the current results, we will be able to have Linux+RTAI+TN inside the next generation cameras. Thus, next generation of vision sensors can be HRT-CORBA enabled.

6 References

[CORB] Common Object Request Broker Architecture (CORBA/IIOP) Specification 3.01, Object Management Group, Needham, MA, U.S.A., 2002, http://www.omg.org

[RTCORB] OMG: Real-Time CORBA 1.0 Specification, http://www.omg.org

[InCORB] Mowbray, T. J. and W. A. Ruh: "*Inside CORBA*", Addison Wesley, 1997.

[ETRAX] Axis Developer Board LX, Axis Communications, 2002, http://developer.axis.com/products/devboard/index.html

[Mart2002] A. Martinsson: "Scheduling of realtime traffic in a switched Ethernet network", Master thesis, Dept. of Automatic Control, 2002.