Real-time Performance Measurements using UDP on Windows and Linux

Gunnar Prytz
ABB AS Corporate Research Center
Bergerveien 12
N-1375 Billingstad, Norway
Gunnar.Prytz@no.abb.com

Svein Johannessen
ABB AS Corporate Research Center
Bergerveien 12
N-1375 Billingstad, Norway
Svein.Johannessen@no.abb.com

Abstract
The move towards Ethernet based communication solutions in automation is accompanied by a need for determinism and real-time performance. The subject of this paper is the measured real-time performance across a switched Ethernet segment.

Performance at 100 Mbps and 1 Gbps network speed was measured using UDP on Windows and Linux. Output stack times in the 8-16 µs range were measured with low processor load. The real-time performance decreased significantly at high processor load.

Measurements of performance across the network showed that 96-98% of the time was spent in the nodes at 1 Gbps network speed. At 100 Mbps 79-87% of the time was spent in the nodes. This paper concludes that the real-time Ethernet performance is determined mostly by the end node performance.

1 Introduction
As the network speed of Ethernet based communication is increasing to a gigabit per second and beyond, the real-time performance of the communication is usually no longer limited by the network speed [4], [14]. The most relevant real-time performance characteristics in this context are jitter and response times. At high network speeds the real-time performance is determined by factors like stack traversal times and delays related to interrupt scheduling and processing in the end nodes of the network.

Traditionally, Ethernet has not been viewed as a candidate technology in real-time communication due to the determinism problem related to the CSMA/CD (Carrier Sense Multiple Access/Collision Detection) mechanism. CSMA/CD enables network devices to detect collisions, e.g. when two or more devices try to send at the same time on the same link. After detecting a collision, a device waits a random delay time and then attempts to retransmit the message. If the device detects a collision again, it waits twice as long to try to re-transmit the message. This procedure can be repeated numerous times and in a worst case scenario the transmission may be aborted. Thus there is no absolute upper time limit on the communication and no determinism when relying on CSMA/CD based Ethernet communication.

A number of protocols focusing on achieving deterministic communication on Ethernet have evolved to varying degrees of maturity during the last few years. These include Ethernet Powerlink [5], PROFINET IRT [12], EtherCAT [2] and SERCOS III [15]. Most of the deterministic protocols utilize some sort of time-slicing strategy, i.e. each network node has a given amount of time for communication. In this way no collisions occur and determinism can be guaranteed. While providing fast and presumably reliable communication, the specific protocols for real-time communication to a variable degree suffer from such drawbacks as lack of standardization, cost of proprietary hardware and cost of stack implementations etc.

Switched Ethernet on the other hand uses a star based topology that does not have any collisions since it provides a private collision domain to each of the ports of a switch. This opens up for the use of Ethernet as a reliable deterministic communication technology. With a satisfactory level of deterministic behavior, switched Ethernet is believed to enter the automation and process control arena [1].

The UDP protocol (User Datagram Protocol) [16] is a freely available and well-known standard with low protocol overhead. It is therefore of interest to investigate the real-time performance that can be achieved with the connectionless UDP protocol using state-of-the-art PC technology and switched Ethernet. Results from real-time performance testing on switched Ethernet have only to a limited degree been reported before due to the relatively recent entry of this technology, although some studies exist [9], [10]. Most studies presenting results on Ethernet performance are concerned with bandwidth, i.e. system throughput, under various conditions [6], [11]. However, bandwidth is only a prerequisite for real-time performance. Keeping latencies under control and as low as possible is crucial in order to make a fast deterministic Ethernet based network.

Both the network bandwidth and the performance of network components are rapidly increasing, making increased levels of performance available. End-to-end latencies down to approximately 25 µs have recently been reported for 10 gigabit Ethernet communication.
through a switch using dual Xeon 2.2 GHz processors at the end nodes [7], [8]. The results presented in this paper show that the end-to-end latency on gigabit Ethernet communication through a switch using UDP and Intel Celeron 2.5 GHz processors in the end nodes is approximately 25 μs on Linux and approximately 53 μs on Windows XP for the particular set of components used.

It is of particular interest to investigate the stack traversal times and the communication delays related to interrupt scheduling in the end nodes of the network, i.e. the computers. In this paper the stack traversal times have been measured. The total end-to-end delays have also been measured, i.e. by measuring the cycle time. The cycle time is defined as the time needed to send a packet from one computer to the other and then sending a reply packet back, including all stack traversals and network delays on cables and a switch. From these measurements the various contributions to communication latencies can be compared and discussed. Also the behavior of the network interface card (NIC) is discussed. Finally, a comparison of the real-time UDP performance of Windows XP and Linux will be performed.

The test results will also show that high processor load has a detrimental effect on networking performance, even though the network interface card had a high system priority. Both the stack times and the jitter increased significantly when the processor experienced heavy load.

The results presented in this paper clearly points out the end nodes as the bottlenecks in a high-performance real-time network. In order to achieve a high real-time performance in an Ethernet based system, the stack traversals have to be very fast and the interrupt related latencies in the system have to be very small.

2 Experimental setup

The experimental setup consisted of two identical computers running either Windows XP or Gentoo Linux 2004.2 with kernel 2.4.26 through dual booting, as shown in Figure 1. The computers had a Marvell Yukon Gigabit (10/100/1000 Mbps) Ethernet adapter integrated on the motherboard. Both computers had a Intel Celeron D325 2.53 GHz processor and 512 MB PC3200 DDR-DIMM RAM.

The computers were connected to each other via a D-Link DGS1216-T 10/100/1000 Mbps switch. The switch was disconnected from the internet during the tests to avoid interference from other Ethernet traffic.

During tests with low processor load no user processes other than the measurement program was running on the computers. To perform tests also with high processor load, two load processes were started on the computers. One of the processes calculated the Euler $\gamma$ constant using the formula

$$\gamma = \lim_{p \to \infty} \left( \frac{1}{p} \right) - \log p$$

The other load process calculated the natural logarithm of 2 using the series expansion

$$\ln(1 + x) = x - \frac{x^2}{2} + \frac{x^3}{3} - ...$$

The load processes did run at least as long as the duration of the tests before they were terminated. Due to limited resolution of the system clock (typically 1-10 ns), the tests were repeated 10000 times to get an average value with resolution 10000 times better than the system clock. In addition, each experiment was repeated 25 times and the mean and the standard deviation of the measurement values were calculated. From statistics it is estimated that approximately 99.7% of all measurements will be within ±3 standard deviations, thus the average value ±3 standard deviations will be presented. The tests were performed with the network interface speed set to (I) 100 Mbps and (II) 1 Gbps.

Minimum Ethernet packets of 84 bytes including all overhead were used. This corresponds to a minimum IP frame of 46 bytes, of which 20 bytes are IP overhead and 8 bytes are UDP overhead, leaving 18 bytes for data.

The following three tests were performed:

- **output stack time**
- **output-input stack time**
- **cycle times - performance across the network**

The UDP output stack performance was measured by sending UDP packets to port 2747 on IP address 192.0.2.96 (which does not exist on the network). The test was performed both with low and high processor load. In these tests, only the performance of one of the nodes in the network (i.e. one of the computers in Figure 1) was considered.

The UDP output-input stack time was measured by sending UDP packets to port 2748 on IP address 127.0.0.1 (the loopback address, i.e. the sender and the receiver is the same computer) which was connected to a UDP socket. The data sent out on the port will be received by the socket and pass into the stack again. Thus, the packet will be transported through both the output and the input stack. The test was performed both with low and high processor load. In these tests, only the performance of one of the nodes in the network (i.e. one of the computers in Figure 1) was considered.

In the case where cycle times were tested, one cycle was defined as the time required to send a UDP packet from computer A to computer B and then to send a reply UDP packet from computer B on computer A. The received packets had to be read before the reply packets could be sent. Two UDP sockets were connected to port 2749 on computer B and to port 43953 on computer A, respectively. The time it took to complete 10000 cycles was measured.
Figure 1. The measurement setup consisting of two identical computers with gigabit Ethernet interfaces connected via a switch.

3 Results

This section contains the results from the experiments. Both output stack times, output-input stack times and performance across the network was measured. No other network traffic interfered with the communication generated by the test programs.

3.1 Windows

The test results from the UDP performance measurements on the Windows XP platform are presented in Table 1. Each test was repeated 25 times and the mean value and standard deviation of the measured times were calculated.

Table 1. Summary of the results from the measurements of real-time performance of the UDP protocol on Windows XP. Mean value ± 3 standard deviations is presented.

<table>
<thead>
<tr>
<th>Time (μs)</th>
<th>Processor load</th>
<th>Network speed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>100 Mbps</td>
</tr>
<tr>
<td>UDP output stack time</td>
<td>Low</td>
<td>15±2.3 μs</td>
</tr>
<tr>
<td></td>
<td>High</td>
<td>91±29 μs</td>
</tr>
<tr>
<td>UDP output-input stack time</td>
<td>Low</td>
<td>23±2.8 μs</td>
</tr>
<tr>
<td></td>
<td>High</td>
<td>97±47 μs</td>
</tr>
<tr>
<td>UDP performance across the network (cycle time)</td>
<td>Low</td>
<td>125±2.6 μs</td>
</tr>
<tr>
<td></td>
<td>High</td>
<td>171±24 μs</td>
</tr>
</tbody>
</table>

3.2 Linux

The test results on the Linux platform are presented in Table 2. It was not possible to measure the total output-input stack traversal times on the Linux platform since the stack seemed to be bypassed when sending to and receiving on the same IP address. This behavior was observed both when using the host loopback address 127.0.0.1 and the true IP address of the computer. The measured times for UDP output-input stack time are therefore very short and not realistic. Each test was repeated 25 times and the mean value and standard deviation of the measured times were calculated.
Table 2. Summary of the results from the measurements of real-time performance of the UDP protocol on Linux. Mean value ± 3 standard deviations is presented. The measured values for the UDP output-input stack times are unrealistic, see the text for details.

<table>
<thead>
<tr>
<th>Time (µs)</th>
<th>Processor load</th>
<th>Network speed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>100 Mbps</td>
</tr>
<tr>
<td><strong>UDP output stack time</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low</td>
<td>7.8±0.4 µs</td>
<td>7.9±0.1 µs</td>
</tr>
<tr>
<td>High</td>
<td>28±15 µs</td>
<td>27±2.3 µs</td>
</tr>
<tr>
<td><strong>UDP output-input stack time</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low</td>
<td>3.3±0 µs *</td>
<td>3.2±0 µs *</td>
</tr>
<tr>
<td>High</td>
<td>13±8.4 µs *</td>
<td>12±1.7 µs *</td>
</tr>
<tr>
<td><strong>UDP performance across the network (cycle time)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low</td>
<td>76±0.4 µs</td>
<td>51±3.7 µs</td>
</tr>
<tr>
<td>High</td>
<td>101±17 µs</td>
<td>73±7.8 µs</td>
</tr>
</tbody>
</table>

4 Discussion

The results from the tests of UDP performance on Windows XP and Linux shows that the UDP protocol to some extent has potential to be used for real-time communication but that the performance depends heavily on the processor load. As could be expected, the jitter becomes very significant for a system experiencing high processor load.

4.1 Stack traversal

For Windows XP, the output stack times at 100 Mbps were found to be 15±2.3 µs at low processor load and increased to 91±29 µs with high processor load. For Linux the corresponding times were measured to be 7.8±0.4 µs with low processor load and 28±15 µs with high processor load. Thus, the performance decreased significantly when the processor was experiencing high load.

At a network speed of 1 Gbps, the output stack times for Windows XP were found to be 16±1.9 µs and 80±36 µs with low and high processor load, respectively. The theoretical minimum time to transmit 84 bytes (=672 bits) is 6.72 µs on a network running at 100 Mbps and 0.672 µs on a 1 Gbps link. Since the measured values for both Windows XP and Linux are significantly higher than these low limit values it can be concluded that the UDP stack output performance is not limited by the network link speed. Furthermore it can be noted that the output stack times are not significantly different on 100 Mbps and 1 Gbps network speeds (taking 3 standard deviations as the confidence limit).

Due to some sort of optimization of the Linux UDP stack the output-input stack time could not easily be measured. However, on Windows XP output-input stack times of 23±2.8 µs and 26±2.58 µs was measured for network speeds of 100 Mbps and 1 Gbps, respectively. The performance decreased significantly when the processor was experiencing high load.

As a first assumption, the measured cycle time should include twice (for each computer) the round-trip stack time and the network delay (including the switch). The propagation delay on the cable is very short (below 100 ns for 10 m cable). The switch delay for a 84 byte UDP packet would in theory represent at least 6.72 µs for a 100 Mbps link and 0.672 µs for a 1 Gbps link. In practice the switch delays may be slightly larger. It is seen that the measured cycle times are significantly larger than the contributions from stack traversals and network delay. Most of the additional delay comes from interrupt related latencies in the computers. The different contributions to the measured cycle times are summarized in Figure 2.

One important factor to be aware of is that the measured stack traversal times do not include the hardware, i.e. the packets only go down to the IP layer but not further down. Thus, no interrupt handling from the hardware is included in the measured stack times.

Two hardware-related mechanisms are suggested as significant contributors to the cycle time, namely interrupt latencies and limitations in the network interface card (NIC).

The cycle times measured therefore consist of the following parts:

- **stack traversal times in the two computers**
- **network delays in cables and the switch**
- **interrupt related latencies**
- **network interface card related latencies**

Of these parts, only the stack traversal times were measured. The cable delay is very short as the signals propagate with a speed in the range of one order of magnitude below the speed of light. The delay through the switch is at least one packet delay, which is proportional
to the packet size and inversely proportional to the network speed, as discussed above. In the next section, interrupt and hardware related latencies will be discussed.

### 4.2 Interrupt handling

There are several types of delays that are related to interrupt handling in a computer. This has to do with the way a computer handles an interrupt. Basic interrupt latency is usually defined as the time it takes for a computer system to begin running the interrupt code associated with a particular event. During this time the CPU finishes processing the current instruction, flushes the instruction pipeline, reads the interrupt vector, locates the address of the trap handler and jumps to that address [13].

Then the operating system trap handler records the current computer state and starts an interrupt dispatcher that determines the source of the interrupt and transfers control to an interrupt service routine (ISR), which is provided by the device driver for the particular device that caused the interrupt. Finally, the ISR performs I/O transfer to or from the device that generated the interrupt.

According to Microsoft [13], the chain of events caused by an interrupt typically can take 10-30 μs. The exact timing is difficult to estimate as it will depend on factors like processor clock rate, operating system and driver version and performance.

A modern network interface card (NIC) has a number of parameters that can be tuned to optimize the performance of the device. NICs designed for server systems typically have the most settings. The network interface used in these tests had numerous parameters adjustment possibilities on the Windows XP platform. The two most important parameters was found to be:

- interrupt moderation (on-off)
- maximum number of interrupts per second (1000-30000)

These settings were not available on the Linux platform. Thus, the following discussion of NIC settings is related only to the Windows XP measurements. Interrupt moderation means that the network interface does not necessarily generate an interrupt for every receive and send operation, but waits for more receive/send operations to occur and then generates an interrupt. In this way the number of NIC interrupts in a system will be lower during periods of high packet rates, thereby preventing system interrupt overloading and a situation called receive livelock [14]. Receive livelock occurs when the interrupt rate is so high that the system spends all the time responding to interrupts, resulting in zero system throughput. Interrupt moderation solves this possible problem at the expense of system performance, as the communication packets will be delayed when the system waits for more packets. Interrupt moderation is a so-called dynamic auto-tuning feature [13].

Setting the interrupt moderation parameter on and off did not show any significant effect on the measurements reported for the performance across the network on Windows XP. This presumably indicates that the dynamic auto-tuning of the NICs functioned well and that the interrupt rate was not high enough to create any system throughput problems.

However, as could be expected, the maximum interrupt rate per second proved to be a very important parameter on Windows XP. The parameter had a default value of 5000, which on average would indicate that the time between interrupts was limited at 200 μs. With this setting, the cycle times in the tests never came below 200 μs. However, each node generated 10000 interrupts each second when the cycle time was 200 μs. Detailed studies of the network traffic with these settings (discussed below) showed that the cycle time occasionally switched from approximately 200 μs to approximately 400 μs for

---

**Figure 2.** The different contributions to the measured cycle times are shown. The average stack traversal times are displayed (taking into account the ±3 standard deviations error limit the values for 100 Mbps and 1 Gbps on the same operating system are not significantly different). The contributions from interrupt related latencies are comparable to or even larger than the stack traversal times. The network delay is set equal to the packet delay through the switch.
some seconds before being switched back down again. This phenomenon will be discussed further below.

The measured cycle times seemed to be inversely proportional to the parameter controlling the maximum number of interrupts per second, but this correlation was only found for settings up to approximately 20000 interrupts per seconds. This indicates that in order to utilize the full performance potential of the test network on Windows XP this setting had to be above 20000. In all tests reported here this parameters was thus set to the maximum value of 30000 interrupts per second.

The switch was set up with a mirroring port, outputting all the traffic to and from computers A and B on one port. This port was connected to a computer running the protocol analyzer software Ethereal [3], which captured and time-stamped all the packets. Given the inaccuracy of such a software based protocol analyzer running on a Windows XP platform, the accuracy is still believed to be in the order of tens of microseconds.

The measurements with Ethereal showed that the parameter controlling the maximum number of interrupts per second had a profound effect on the measurements. This is demonstrated in Figure 3, where the cycle times represented by the time differences between each packet from computer A is shown for different interrupt rate parameter settings.

The cycle times shown in Figure 3 display a significant amount of variability. This is presumably due to both Ethereal inaccuracies and network performance variability. The real-time performance results reported in this paper do not depend on Ethereal performance. However, there was a clearly visible variable cycle time behavior for some interrupt rate parameter values.

For interrupt rate settings below approximately 20000 a detailed analysis of the cycle times showed an interesting effect. An unstable cycle time behavior occurred when the maximum allowable interrupt rate per second was set below approximately 20000. The mean cycle times switched between two or more levels. The difference between the two levels was always equal to the inverse of the interrupt rate setting. Furthermore, the lowest level was equal to a multiple of the interrupt rate setting.

4.3 Discussing the results

It is suggested that the reason for the switching is as follows: the network starts up but after some time (seconds) the system notices that the number of interrupts coming from the NIC is larger than the maximum value set in the interrupt rate parameter. Special attention is then put to the NIC. For the next time period (in the order of seconds) the cycle time is increased due to the fact that the system now allows only as many interrupts from this system as the NIC setting. The interrupt rate is now below the maximum allowed level and after a few second the NIC is allowed to run freely again, and the whole behavior repeats itself. For the minimum achieved cycle time of slightly above 100 μs the interrupt rate is slightly below 20000 interrupts per second. This explains why the unstable cycle time behavior was only observed for interrupt rate settings below approximately 20000, as is demonstrated in Figure 3.

An interesting point seen from the results is worth stressing. From the Windows XP results in Table 1 the measured mean cycle time was 106 μs for a network running at 1 Gbps. Assuming that less than 2 μs is spent propagating the cables and in the switch, this means that 104/106=98% of the time is spent inside the end nodes. For 100 Mbps the number is (125-16)/125=87%.

Correspondingly, from the Linux results in Table 2 it is seen that the measured mean cycle time for a network running at 1 Gbps was 51 μs. Again assuming that less than 2 μs is spent propagating the cables and in the switch, 49/51=96% of the time is spent inside the end nodes. For 100 Mbps the corresponding result is (76-16)/76=79%.

This simple calculation clearly shows that the critical performance factor in hard real-time communication networks is the end node performance and the protocol stack implementation. The end nodes are the bottlenecks in the network both for network speeds of 100 Mbps and 1 Gbps. To meet the toughest requirements, the protocol stack must be streamlined and optimized for real-time performance and the operating system must also have real-time performance. Thus, to achieve a higher degree of real-time performance than what UDP provides, deterministic protocols specially constructed for determinism and low jitter should be adopted.

There was a significant difference in real-time performance between Windows XP and Linux, with Linux providing higher real-time performance than Windows XP. The reason for this difference is unclear, but it is supported also by other studies [6], [11]. The difference may have to do with both stack implementation and operating system process handling. No attempts were made to optimize the performance of any of the operating systems by changing interrupt priorities etc.

It is also seen from the measurements that most of the performance difference between networks running at 100 Mbps and 1 Gbps comes from the ten-fold difference in switch delay between these two network speeds. Thus, the performance gain when going from 100 Mbps to 1 Gbps is mostly related to the increased switch propagation speed. This argument is, however, only true as long as the bandwidth is not heavily used, causing switch queuing and possibly switch overflow. For systems with very hard real-time requirements, gigabit Ethernet systems are very interesting since packet delays will be very low. Switched gigabit Ethernet thus provides a very fast, very deterministic communication technology for the automation industry.
Figure 3. Cycle times for different NIC maximum interrupt per second settings (indicated by the number in the boxes). First 50000 cycles with low processor load followed by 50000 cycles with high processor load. For more details see text.
References


