

Performance Study of Winsock Direct with 10 Gb Ethernet RDMA-Enabled NIC

Brian Hausauer

Chief Architect - NetEffect, Inc.

9211 Waterford Centre Blvd, Suite 100, Austin, Texas 78758

bh2006@neteffect.com

Abstract

This paper describes a performance study conducted with the new 10 Gb Ethernet iWARP RDMA-enabled adapter from NetEffect, using Microsoft Winsock Direct Server Area Network technology. The paper starts with a brief summary of the Winsock Direct software architecture and the NetEffect hardware architecture. It provides initial performance results for bandwidth, CPU utilization, and latency. It analyzes the performance results, and concludes with some discussion on future work that can lead to further performance improvements.

1. Introduction

Microsoft introduced the Winsock Direct architecture in 2001 with Windows 2000 Datacenter Server. The goal of Winsock Direct (WSD) is to enable unmodified sockets applications to take advantage of the network stack offloads available in modern Server Area Network (SAN) adapters. While this paper does not provide a survey of past work in SAN network stack offloads, it is worthwhile to note that U-Net [10] represents early work in this area, and that Virtual Interface Architecture [11], InfiniBand [12], and iWARP [13] represent significant advances. Network stack offloads move network processing from the host CPU onto the network adapter. WSD utilizes all of the following network stack offloads: Reliable Connection transport offload, Remote Direct Memory Access (RDMA) capability, and OS bypass, which eliminates OS intervention between user-level applications and network adapter hardware.

Over the years, several types of SAN adapters have supported WSD, including Compaq ServerNet, Giganet CLAN, and InfiniBand. None of these adapters support the common datacenter networking technologies, Ethernet and TCP/IP. A new class of network adapters has recently come on the market, that *do* support Ethernet and TCP/IP, and also support the iWARP RDMA standard. Such adapters can be appealing to datacenter IT managers and may lead to more pervasive deployment of WSD.

2. Winsock Direct Architecture Summary

The WSD architecture is shown in Figure 1. Applications talk to the Windows Sockets layer in an unmodified fashion. The Windows Sockets Switch keeps a list of IP subnets that are network offload-

accelerated. When both endnodes in a sockets session are network offload-accelerated, and on the same IP subnet, and when the session was created to use TCP transport, then the Windows Sockets Switch makes the connection using the WSD Provider path. Otherwise, the Windows Sockets Switch makes the connection using the Host TCP/IP stack. There can be a combination of network offload-accelerated and unaccelerated traffic on the IP subnets of the SAN. The SAN NIC supports this mixture of traffic by providing a normal NDIS driver interface for connection to the Host TCP/IP stack, and by providing a proprietary interface to the WSD Provider and the WSD Proxy Driver for SAN services.

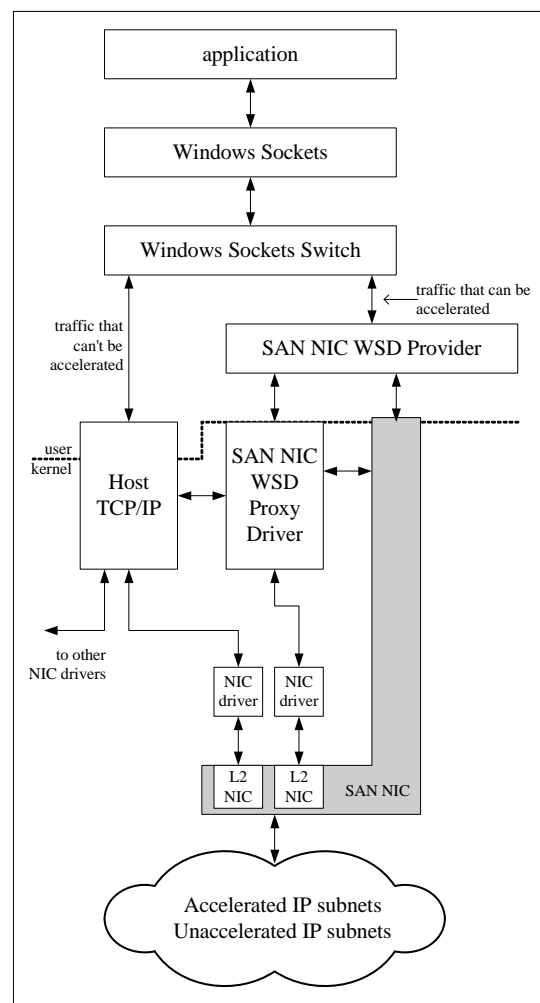


Figure 1. Winsock Direct Architecture

Note that while the WSD Provider communicates to the WSD Proxy Driver for things like SAN connection management and Memory Registration, steady-state data transfer operations are typically conducted directly with the SAN NIC, enabling OS bypass as described earlier.

[1] [2] and [3] provide more details on WSD architecture.

Other commercially available network stack offload solutions designed to work with unmodified sockets applications include the Microsoft TCP Chimney [14] and Sockets Direct Protocol (SDP) running on Linux. TCP Chimney utilizes Reliable Connection transport offload, but not RDMA or OS bypass. It has the advantage of being completely compatible with legacy TCP/IP solutions and can be used even when only one of the endnodes in a sockets session is equipped with a SAN adapter. SDP on Linux utilizes Reliable Connection transport offload and RDMA, but not OS bypass. More on SDP in later sections.

3. NetEffect Adapter Hardware Architecture

Figure 2 is a high-level block diagram of NetEffect's NE010 Ethernet Channel Adapter (ECA). The ECA is a single chip that integrates basic Layer 2 NICs, a TCP Offload Engine, and iWARP RDMA capability. Here is a brief description of the blocks : The *Host Interface* is PCI-X 64/133, having a peak burst bandwidth of ~8 Gb/s. PCI-X can not support the full throughput of a 10 Gb Ethernet port, which is capable of 10 Gb/s unidirectional (and 20 Gb/s bidirectional) peak bandwidth. PCI Express is a newer host interface technology supporting peak bandwidth of 16 Gb/s unidirectional (32 Gb/s bidirectional) or more depending on link width. NetEffect chose PCI-X for the NE010 over the more performant PCI Express based on limitations in the ASIC technology used to implement the NE010. A follow-on chip with PCI Express x8 link is in the works. The *MAC* is a standard Layer 2 Ethernet MAC that connects to 10 Gb Ethernet (10GBASE-CX4, 10GBASE-SR, etc) or to 1 Gb Ethernet. The *Protocol Engine* performs all of the complex network protocol processing. It contains 20 sub-blocks, including three Tensilica Xtensa 32-bit CPUs. The sub-blocks perform functions such as work scheduling, IP, TCP and iWARP header processing, connection context management and caching, memory protection and translation, transport timers, completion processing and associated event generation, window buffer management, DMA operations to Host Memory to fetch work requests or source/sink data payloads, and Layer 2 through 4 statistics counters. Much of the steady-state packet processing is performed with hardware state machines, however exceptional conditions such as packet retransmission are performed by CPUs. Each

CPU is optimized for a specific function, with custom instructions designed by NetEffect, and with additional integrated hardware assists. For both ingress and egress, protocol processing is broken into a series of pipe stages, where each stage is separated by an elastic FIFO. The stages are designed so that requests for off-chip information (e.g. connection context or Host DMAs) are made at the end of a stage, so that information is local on-chip at the start of the next stage. Such pipelining allows the NE010 to achieve a very high steady-state packet processing rate of well over one million packets per second unidirectional, a difficult achievement for a network transport termination device.

The *Transaction Switch* is a memory-based crossbar for short-term storage of packets. Typically, packets experience a single store-and-forward in the Transaction Switch when traversing the chip.

The *DDR2 SDRAM Controller* provides on-adapter storage for a variety of network-related information, including: connection context, memory region tables and virtual-to-physical address translation tables required for RDMA, receive window buffers (typically only used on non-RDMA accelerated connections), etc. The *DDR2 SDRAM Controller* peak throughput is 3.2 GB/s, and it supports connection of up to 4 GB RAM (256 MB typical).

For more information on the NE010, including details on OSes and middleware supported, see [4].

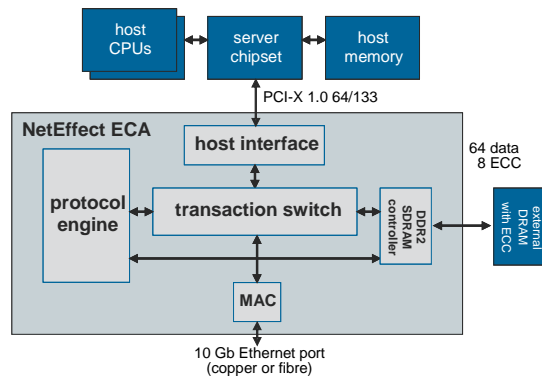


Figure 2. NetEffect Adapter Hardware Architecture

4. Experimental Results

All test results reported herein were obtained using the following setup: Two servers are connected back-to-back with NE010 ECAs, or alternatively with conventional state-of-the-art 10 Gb Ethernet NICs. Each server has two 2.8 GHz Opterons with 1 MB L2 Cache, and a Supermicro H8DC8 motherboard with 2 GB 128-bit wide DDR400 SDRAM installed. Each adapter is installed in a PCI-X 64/133 slot. OS is Microsoft Windows Server 2003.

4.1. Throughput

WSD throughput testing was performed using NTttcp, the Windows-specific version of the popular ttcp benchmark. NTttcp is distributed in the Microsoft Windows 2000 Driver Development Kit (DDK). Its use is summarized in [5]. NTttcp represents a level of sockets performance optimization perhaps not achieved by many real-world sockets applications, including the ability to aggressively pre-post socket receive buffers and the ability to have many socket sends simultaneously outstanding. These abilities are realized using overlapped I/O (OIO), supported by the Microsoft Winsock2 API [6], but not widely supported by other OSes. OIO allows sockets applications to conduct network communications in a fully asynchronous fashion. For example, the application can post a series of messages for transmit, and be notified later of transmit completion allowing buffer reuse. Similarly, the application can post a series of messages for receive (empty buffers), and be notified later of receive completion allowing received data payload processing. OIO has the potential to significantly increase the performance of sockets applications, because it enables pipelined network communications, while minimizing the need to buffer copy messages within the networking stack.

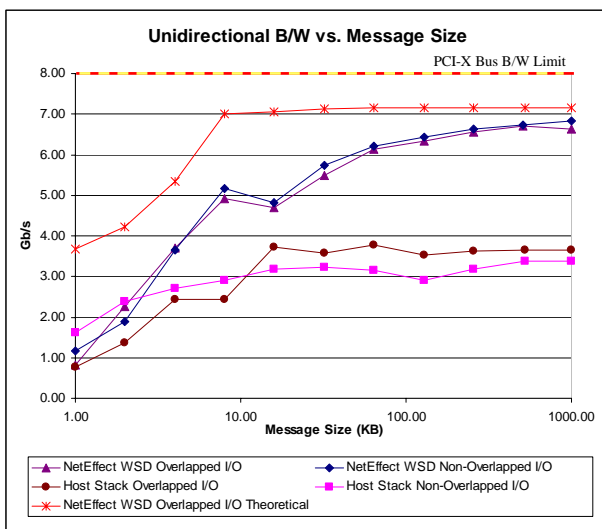


Figure 3. Throughput Results

Figure 3 shows throughput results running NTttcp for a range of message sizes from 1 KB to 1 MB. The x axis is shown on a log scale to highlight the small-message performance. All results were obtained using standard-sized 1500 byte Ethernet frames. The curves labeled *Host Stack Overlapped I/O* and *Host Stack Non-Overlapped I/O* show results using said conventional 10 Gb Ethernet NIC with the Host OS performing TCP/IP processing. All of the stateless

NIC offloads implemented by Windows Server 2003 are enabled (checksum offload, LSO, etc). For *Host Stack Overlapped I/O* the transmitter was configured for 8 OIOs and the receiver for 24 OIOs. For large message sizes (e.g. 128 KB), OIO provides about 20% performance boost over non-OIO. For smaller message sizes, OIO performs worse than non-OIO. This is possibly due to the transmitter being limited to only 8 OIOs, which may not be enough to achieve good pipelined throughput on the network for message sizes less than about 8 KB (8 OIOs x 8 KB = 64 KB outstanding). Then again, OIO requires the Host OS to perform extra bookkeeping, and perhaps at the smaller message sizes, this overhead does not pay off.

Refer now to the curves labeled *NetEffect WSD Overlapped I/O* and *NetEffect WSD Non-Overlapped I/O*. These results were obtained using the NE010 ECAs and WSD. For *NetEffect WSD Overlapped I/O* both the transmitter and receiver were configured for four OIOs (more on this later). For large message sizes, throughput is almost double that of the conventional NIC, approaching the 8 Gb/s PCI-X bus bandwidth limit. There are three perceived anomalies in this data :

- For messages smaller than ~2 KB WSD throughput drops below that of the conventional NIC using non-OIO.
- The transition from 8 KB to 16 KB message size shows a consistent dip in throughput.
- The expected performance benefit of OIO does not exhibit.

We believe many of these anomalies can be explained as follows: WSD uses proprietary data transfer mechanisms which appear to be similar in many respects to the mechanisms defined by Sockets Direct Protocol [7]. SDP defines a *Bcopy* mechanism using iWARP Send operations designed to transfer data between OS-owned buffers. SDP defines *Read Zcopy* and *Write Zcopy* mechanisms using iWARP RDMA Read or RDMA Write operations designed to transfer data directly between application-owned buffers. SDP discusses the use of a *Bcopy Threshold* used to determine which data transfer mechanism to use. The idea is to use *Bcopy* for small messages, and either *Read Zcopy* or *Write Zcopy* for larger messages.

Analyzing wire traffic, we observe that WSD uses *Bcopy* mechanism for message sizes up to 8 KB. We observe a maximum of four *Bcopy*-related iWARP Send operations simultaneously outstanding. For message sizes >8 KB we observe that WSD switches to *Read Zcopy* mechanism. We observe a maximum of one *Read Zcopy*-related iWARP RDMA Read operation outstanding. With this information, we draw the following conclusions:

At message size of 2 KB, the *Bcopy* behavior of WSD enables only 8 KB outstanding network data payload. As alluded to earlier, a simple bandwidth-delay analysis of this network will show that 8 KB is not enough outstanding data payload to achieve good pipelined throughput at 10 Gb bitrate. Something like 64 KB outstanding data payload would be a more comfortable number, which would require WSD to support at least 16 *Bcopy*-related iWARP Send operations simultaneously outstanding at 2 KB message size.

The transition from 8 KB to 16 KB message size provides another interesting example of the effect outstanding network data payload can have on throughput. At message size of 8 KB, the *Bcopy* behavior of WSD enables 32 KB outstanding network data payload. At message size of 16 KB, the *Read Zcopy* behavior of WSD enables only 16 KB outstanding network data payload. Again, this is not enough outstanding data payload to achieve good pipelined throughput at 10 Gb bitrate, and hence the dip in throughput.

As mentioned earlier, for *NetEffect WSD Overlapped I/O* both the transmitter and receiver were configured for only four OIOs. Increasing the number of OIOs did not improve performance. This is almost certainly because WSD allows only four *Bcopy*-related iWARP Send operations simultaneously outstanding. This is likely the main reason OIO does not exhibit a performance benefit over non-OIO in our WSD data.

The Figure 3 curve labeled *NetEffect WSD Overlapped I/O Theoretical* represents our attempt to simulate the throughput potential of WSD using the NE010 ECAs, assuming WSD is enhanced to allow more outstanding network data payload. More details on this in section 5.

4.2. CPU Utilization

While collecting the throughput data from section 4.1 we also captured the CPU utilization reported by NTttcp. This data is presented in Figure 4 for the OIO test cases only. The figure shows Gb/s per CPU GHz for the same range of message sizes from 1 KB to 1 MB. This style of depicting CPU utilization has become popular in networking papers (e.g. [8]). The curves represent average CPU utilization from the two servers. For WSD, CPU utilization between the transmit server and the receive server differs by at most 10%. For conventional NIC, CPU utilization between the transmit server and the receive server can differ widely. This is because stateless NIC offloads are much more effective on the transmit server. For example, at 512 KB message size, the transmit server CPU utilization is 9.5%, whereas on the receive server CPU utilization is 51.9%.

Refer now to the curve labeled *Host Stack Overlapped I/O*, which describes the conventional 10

Gb Ethernet NIC with the Host OS performing TCP/IP processing. Conventional wisdom says that 1 GHz of CPU horsepower is required per Gb of network throughput. Our data supports this notion almost exactly at 8 KB message size. At 1 KB message size, the NIC solution achieves 0.21 Gb/GHz, while at 1 MB message size it improves to 4.09.

The curve labeled *NetEffect WSD Overlapped I/O* shows the results obtained using the NE010 ECAs and WSD. The curve shows a significant improvement over the conventional NIC starting at 64 KB message size (3.89 vs. 6.43). At large message size, the benefit of the *Read Zcopy* data transfer mechanism is striking (4.09 vs. 51.11).

The Figure 4 curve labeled *NetEffect WSD Overlapped I/O Theoretical* represents our attempt to simulate the CPU efficiency potential of WSD using the NE010 ECAs, assuming WSD is enhanced to allow more outstanding network data payload. More details on this in section 5.

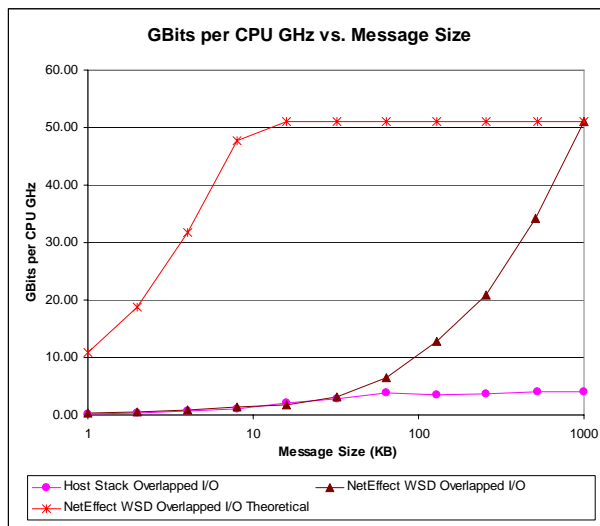


Figure 4. CPU Utilization Results

4.3. Latency

For latency testing, NetEffect used its own internally developed application, *Nespong*. *Nespong* uses non-overlapped, blocking sockets to perform a ping-pong sequence between the transmitter and receiver. The *Nespong* transmitter starts a fine-grained timer, transmits a message of user-specified size and then waits for the receiver to echo the message. It repeats this sequence a user-specified *iteration count* number of times (in this case 4000) and then stops the timer. The reported latency is (timer delta) / (2 * *iteration count*).

Results are reported below in Figure 5, for both conventional NICs and for WSD using the NE010

ECAs, and for a range of message sizes from 1 to 1000 bytes.

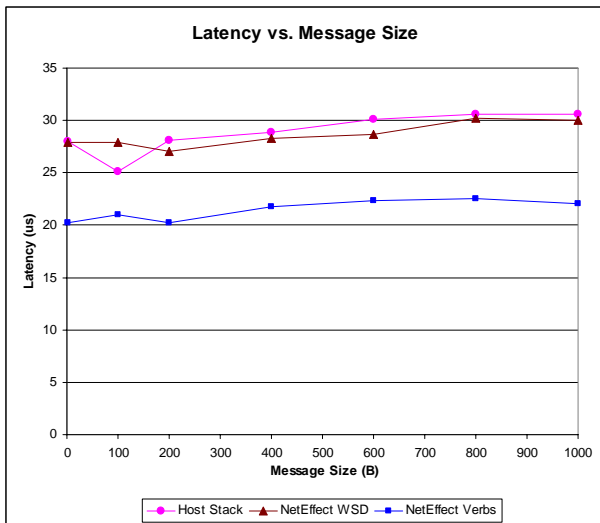


Figure 5. Latency Results

Note that for these latency tests, NetEffect’s own 10 Gb conventional NIC was used, a different NIC than used in sections 4.1 and 4.2. This was done only out of convenience, and should not appreciably change the results.

The results show that latency under WSD using the NE010 ECAs is roughly the same as for the conventional NICs. These results are unsatisfactory, given that under WSD, *Nespong* should be capable to perform the latency loop directly from user space with little OS intervention, and given that NetEffect has measured one-way polling latency of $\sim 10\mu$ using NE010 adapters with its verbs API.

As a comparison test, NetEffect implemented a derivative of *Nespong* implemented directly on top of its verbs API. This derivative uses the same ping-pong algorithm described earlier, conveys messages using iWARP Send operations, and uses interrupts to signal message completions (rather than using polling as is more common when measuring SAN adapter latency). This conveyance model is meant to mimic that of *Nespong* over WSD, but of course there is no WSD-related software overhead included in this derivative test. The Figure 5 curve labeled *NetEffect Verbs* shows the results. Latency measured using verbs is consistently $\sim 7\mu$ s less than when using WSD. As mentioned earlier NetEffect has measured one-way polling latency using verbs of $\sim 10\mu$ s. Comparing this to Figure 5 suggests that using interrupts adds about 10μ s over polling. Additionally, the 7μ s delta between verbs and WSD can not be conclusively accounted for at this time. We are interacting with Microsoft to conduct a more detailed analysis.

5. Future Work

Our future work involves close interaction with Microsoft to explore the perceived WSD bottleneck limiting outstanding network data payload as discussed in 4.1 and to reduce the *Nespong* latency delta between WSD and verbs as described in 4.3.

The NE010 can support several orders of magnitude more Send, RDMA Write and RDMA Read work requests outstanding per TCP/iWARP connection, than currently enabled by WSD. If we somewhat optimistically assume that all of these resources can be effectively applied to WSD *Bcopy*, *Read Zcopy* or *Write Zcopy* data transfer, then we believe we can reach a theoretical WSD throughput as described in the Figure 3 curve labeled *NetEffect WSD Overlapped I/O Theoretical*. This curve represents measured throughput under our verbs API, where all of the NE010 hardware resources are fully utilized.

Another area we plan to explore is more aggressive interrupt moderation at small message sizes. We believe one reason WSD is not showing better CPU utilization at small messages sizes, is due to interrupt rate. iWARP verbs-compliant adapters such as the NE010 support numerous standards-based techniques to moderate interrupts, but it is unclear whether WSD is utilizing these. Some of these techniques can be applied from within the WSD Provider and/or WSD Proxy Driver supplied by NetEffect. The techniques become much more effective after we eliminate the WSD bottleneck limiting outstanding network data payload. Here again, we have conjectured a theoretical WSD CPU utilization as described in the Figure 4 curve labeled *NetEffect WSD Overlapped I/O Theoretical*. This curve represents measured CPU utilization under our verbs API, with traffic pattern similar to that seen under WSD with NTttcp. This curve provides a fairly crude upper boundary that will motivate our next round of performance and efficiency investigations.

We are working on performance comparisons between WSD, SDP on Linux, and TCP Chimney.

Finally, we are carefully watching the research of others [9] experimenting to advance the performance of SDP, for techniques that can also apply to WSD.

6. References

- [1] Pinkerton, Jim, “Winsock Direct: The Value of System Area Networks”, Microsoft Corporation, www.microsoft.com/windows2000/techinfo/howitworks/communications/winsock.asp
- [2] “Winsock Direct and Protocol Offload on SANs”, Microsoft Corporation, www.microsoft.com/whdc/device/network/san/WSD-SAN.msp

- [3] Speight, Evan, Shafi, Hazim, and Bennett, John, "WSDLite: A Lightweight Alternative to Windows Sockets Direct Path", *Proc. 4th USENIX Windows Systems Symposium*, Usenix Assoc., 2000, pp. 113-124.
- [4] "NetEffect NE010 Product Brief", *NetEffect Inc*, www.neteffect.com/documents/NE010_Product_Brief_041906.pdf
- [5] "Performance Tuning Guidelines for Windows Server 2003", *Microsoft Corporation*, Oct. 2003, download.microsoft.com/download/2/8/0/2800a518-7ac6-4aac-bd85-74d2c52e1ec6/tuning.doc
- [6] "Windows Sockets 2 Application Programming Interface", revision 2.2.2, August 7, 1997, ftp.microsoft.com/bussys/winsock/winsock2/WS_API22.DOC
- [7] Pinkerton, James, Delegates, Ellen, Krause, Michael, "Sockets Direct Protocol (SDP) for iWARP over TCP (v1.0)", *RDMA Consortium*, October 31, 2003.
- [8] Regnier, Greg, Makineni, Srihari, Illikkal, Ramesh, et al, "TCP Onloading for Data Center Servers", *IEEE Computer Magazine*, November 2004, pp. 46-56.
- [9] Balaji, P, Bhagvat, S, Jin, H.W., Panda, D.K., "Asynchronous Zero-copy Communication for Synchronous Sockets in the Sockets Direct Protocol (SDP) over InfiniBand", *Workshop on Communication Architecture for Clusters*, April, 2006.
- [10] von Eicken, Thorsten, Basu, Anindya, Buch, Vineet, Vogels, Werner, "U-Net: A User-level Network Interface for Parallel and Distributed Computing", *ACM Symposium on Operating System Principles*, December, 1995.
- [11] "Virtual Interface Architecture Specification v1.0", *Compaq Computer Corp, Intel Corp, Microsoft Corp*, December 16, 1997.
- [12] InfiniBand Trade Association, www.infinibandta.org/
- [13] RDMA Consortium www.rdmaconsortium.org/
- [14] "Scalable Networking: Network Protocol Offload – Introducing TCP Chimney", *Microsoft Corporation*, April 9, 2004.