ETHOM, an Ethernet over SCI and DX Driver for Linux

Rainer Finocchiaro, Lukas Razik, Stefan Lankes, Thomas Bemmerl*

Abstract—**Nowadays, high computing demands are often tackled by clusters of computers, each of which is basically an assembly of a growing number of CPU cores and main memory; these *nodes* are connected by some kind of communication network. With the growing speed and number of CPU cores, the network becomes a severe bottleneck limiting overall cluster performance. High-speed interconnects like SCI and Dolphin DX are good for alleviating this communication bottleneck, when the communication software is either based on IP or specifically adapted to the interconnect. Software written to communicate directly via Ethernet frames can not be used this way, though. In this article, we present ETHOM, a driver that implements an Ethernet interface on top of the Dolphin Express software stack. It enables the Dolphin networks SCI and DX to be used as high-speed replacement for Ethernet. Offering an Ethernet and with that an IP interface, it enhances their functionality and allows usage of layer-2 kernel functionality like interface bonding and bridging. This driver is an improvement of ETHOS, the Ethernet over Sockets driver, delivering lower latencies at the cost of higher CPU load. By means of various measurements, we show that ETHOM with SCI or DX offers a twofold increase in communication performance over Gigabit Ethernet.**

Keywords: Ethernet, SCI, Dolphin DX, Linux, TIPC

1 Introduction

Computational power has always been a scarce resource and prognoses predict that this situation will not change any time soon. While Computer performance increases, the demand for more power increases at least at the same pace.

Until very recently, CPUs as the main component of a computing system grew more powerful by raising the clock frequency. Today parallelism in the form of additional cores per die adds to the performance increase. From a hardware point of view, the next level of parallelism is the gathering of single computers to form a cluster. Traditionally, the single computers – called nodes – in these clusters were connected by Ethernet in one of its incarnations, as it is cheap, stable, and well supported. Concerning the software, the predominant protocol used on top of Ethernet is the TCP/IP stack. With software running on the cluster that communicates intensively, the network is more and more the real bottleneck that limits cluster performance.

So, there are two problems to cope with: (1) First of all, networking hardware in the form of Gigabit Ethernet is too slow for several purposes, 10G Ethernet is still in the beginnings, not yet very wide-spread and rather expensive. (2) Then, TCP/IP is a protocol suite designed for communication in wide area networks, offering elaborate mechanisms for routing, to deal with even extensive packet loss, etc. It is not so well suited for clusters.

To cope with these problems, there are mainly two approaches in order to allow faster[1] communication: (1) The first approach is to use high-speed networks, each having their own low-level programming interface (API), most providing an implementation of the POSIX socket API, and some offering an IP interface. Examples of these networks include InfiniBand[2], Myrinet[3], QsNet[4], SCI[5], and Dolphin DX[6]. An IP interface for Dolphin DX has been presented in [1]. (2) The second approach is to replace the software layer TCP/UDP – and sometimes IP as well – while keeping the Ethernet hardware. Examples of these replacement protocols include SCTP (Stream Control Transmission Protocol [2]), DCCP (Datagram Congestion Control Protocol [3]), UDP-Lite [4], AoE (ATA over Ethernet [5]), and TIPC (Transparent Interprocess Communication Protocol [6][7]).

Being developed originally at Ericsson, TIPC has its origin in the telecommunication sector, but provides some characteristics making it suitable for high performance computing (HPC) with clusters, such as an addressing scheme supporting failover mechanisms and less overhead for exchanging data within a cluster. TIPC is the transport layer of choice of the Kerrighed project [8]. It is used for kernel to kernel commnunication, but cannot currently make use of high-speed networks like InfiniBand

*Chair for Operating Systems, RWTH Aachen University, Kopernikusstr. 16, 52056 Aachen, Germany,
E-mail: {rainer,razik,stefan,thomas}@lfbs.rwth-aachen.de

[1] latency and bandwidth wise

[2] http://www.infinibandta.org/events/past/it_roadshow/overview.pdf

[3] http://www.myri.com/myrinet/product_list.html

[4] http://www.quadrics.com

[5] http://www.dolphinics.com/products/pent-dseries-d350.html

[6] http://www.dolphinics.com/products/pent-dxseries-dxh510.html

or SCI, as they both do not provide an Ethernet interface, nor does TIPC provide a specialised "bearer", which is the adaptation layer between TIPC and a network interface.

A previous article [9] described ETHOS, the first approach of a driver offering an Ethernet interface utilising high-speed networks for communication. This driver was designed to use kernel-level UDP sockets to deliver data to peers (see Figure 2); it enables any network providing kernel-space UDP sockets to be used as Ethernet replacement. Measurements with ETHOS on top of SCI and InfiniBand showed significantly higher bandwidth and lower latency than Gigabit Ethernet.

In order to further reduce communication latency, we decided to abandon compatibility with other high-speed interconnects and use the next lower software layer available in the Dolphin Express stack, the Message Queue Interface. Using this interface, *ETHOM (ETHernet Over Message-Queue driver)* provides an Ethernet interface for SCI and Dolphin DX hardware. Therefore, in addition to the TCP/UDP-Sockets already provided by the Dolphin Express software stack, ETHOM offers an Ethernet interface, enabling interface bonding, bridging and other layer 2 kernel features, as well as (IP-)Routing for the SCI and Dolphin DX interconnects. Furthermore, TIPC is enabled to make use of these two network technologies leveraging its Ethernet bearer, just like any other software that is stacked on top of an Ethernet interface.

The rest of this article is organised in the following way: In section 2, we shortly present the Linux network architecture, serving as a background for understanding where the presented Ethernet interface resides. Furthermore, some details about design decisions are given. Section 3 gives an overview of the performance of ETHOM measured with popular microbenchmarks. Finally, in section 4, we conclude with the current status and plans for further improvements.

2 Architecture

In order to describe the architecture of our Ethernet over Message Queues driver, an overview of the different hardware and software layers is given in section 2.1. Building on that, we shortly describe our first attempt of an Ethernet replacement driver, which uses sockets as "communication medium", in section 2.2. With this background, we go into some detail about the implementation and design decisions in section 2.3.

2.1 Linux Ethernet Network Architecture

Seen from a rather high level of abstraction, the Linux Ethernet network architecture consists of three layers that are used by an application to communicate with a counterpart on another node (see Figure 1). The first layer consists of a programming interface like for example

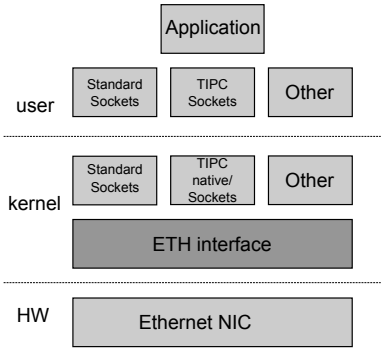

Figure 1: Linux Network Architecture

the well-known and wide-spread *sockets interface*. This layer is available for user-space application processes as well as for the kernel space. It expects user data in any form from the application and builds Ethernet frames which are passed to the next lower layer. The layer below is represented by the *Ethernet interface layer*. This layer takes Ethernet frames from above and passes them on to the hardware. The third and lowest is the *hardware layer*, as represented by the Ethernet NIC, which cares for the physical transmission of data from a sender to its peer.

2.2 Architecture of ETHOS

The article presented in [9] describes previous work on ETHOS, an Ethernet driver built using the Linux kernel sockets API to send and receive data. The Ethernet interface layer no longer passes Ethernet frames directly to the hardware. Instead it makes use of kernel-space UDP sockets to pass on the Ethernet frames to the hardware layer. If a network device does not offer native UDP kernel sockets but an IP driver, standard UDP kernel sockets on top of this IP driver are deployed. (Note that the Ethernet NIC shown in Figure 2 is used for demonstration purposes only. It is in general not reasonable to use it for applications, as performance should by principle always lag behind the native interface.)

2.3 Architecture of ETHOM

Within the work presented in this paper, a thin layer of indirection is inserted below the Ethernet interface (see Figure 3). This layer passes the Ethernet frames to the SCI Message Queues, which represent the lowest message passing layer of the Dolphin software stack. At the lowest level, SCI or DX cards physically deliver the data to the peer nodes.

As depicted in Figure 4, on each node of the cluster there are two message queues for every peer. Each message queue is identified by a cluster-wide unique ID.

The memory requirements for each message queue are dominated by the buffer, which is currently hard-coded to ˜13 KB. As the SCI network allows a maximum of 512

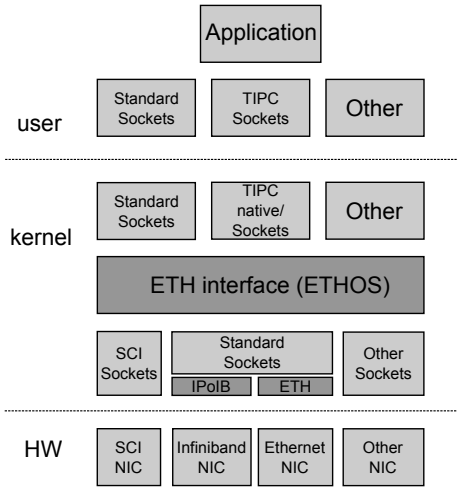

Figure 2: Network Architecture with ETHOS

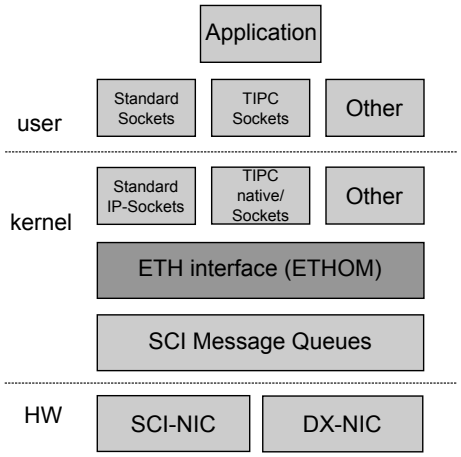

Figure 3: Network Architecture with ETHOM

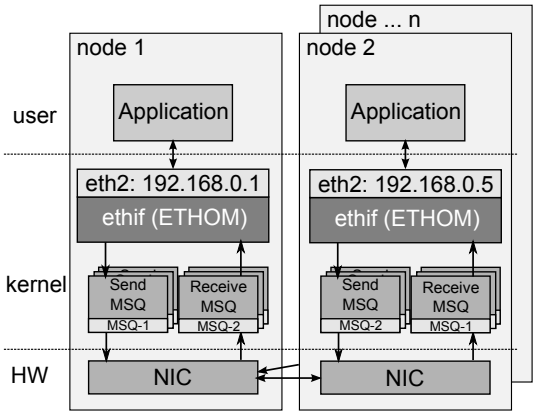

Figure 4: Implementation of ETHOM

nodes when setup as a 3D torus, total memory requirement per node would amount to: number of peers $\times$ 2 queues $\times$ size of queue = 511 $\times$ 2 $\times$ 13 KB = 13 MB. So, concerning memory consumption, our driver should be quite scalable.

As it is very improbable to have more than one card of either SCI or Dolphin DX inside of one node, currently only one Ethernet interface is supported by ETHOM.

Connection Establishment ETHOM is configured in two phases: (1) First, at load time of the driver, the number of nodes in the network are specified, (2) then at runtime, an IP address, MTU[7], broadcast address, and network mask are assigned with *ifconfig* just like with any other ethernet interface.

After loading of the driver, on each node, two unidirectional message queues are created for every peer node in the network (e.g. 14 message queues on each node in case of 7 peer nodes). Message queue IDs are calculated from the local and the peer node number as

$$\begin{aligned} \mathrm{ID}_{\mathrm{ReceiveQueue}} &= \#\mathrm{hosts} \times \mathrm{peer} + \mathrm{local} \\ \mathrm{ID}_{\mathrm{SendQueue}} &= \#\mathrm{hosts} \times \mathrm{local} + \mathrm{peer} \end{aligned}$$

This way they are guaranteed to be unique throughout the cluster.

For each peer node, two threads are started (e.g. 14 threads on each node in case of 7 peers), one trying to connect the local send to the distant receive queue and one waiting for a connection on the local receive queue. As soon as the first of the threads waiting on the local receive queue has accomplished its connection, this thread becomes the *master thread* that polls on all connected receive queues. All the other send and receive threads terminate as soon as their connection is established, effectively reducing the number of remaining threads to one. In case the peer node does not connect directly, a new connection attempt is made periodically.

In case of IP communication on top of ETHOM, IP addresses can be specified arbitrarily, they do not have to correspond to node numbers. Just like with hardware Ethernet devices, ARP[8] is used at first contact to find the node that provides the sought-after IP address. The ARP broadcast request is sent to each connected node sequentially and answered by the node in question. From that point on, communication over IP is possible.

Communication Phase When the application on node 1 (compare Figure 4) sends a message to the application on node 2, this message is passed to the kernel

[7]Maximum Transmission Unit
[8]Address Resolution Protocol

networking stack. The kernel then splits it into packets fitting into the previously specified MTU (Fragmentation) – if necessary – and equips each packet with an Ethernet header. This newly constructed *Ethernet frame* is passed to ETHOM, which hands it over to the send message queue and directly flushes the queue. On the receiving node 2, the Ethernet frame's arrival is detected by the polling thread and immediately forwarded to the kernel networking stack. The kernel reassembles the upper layer message – if necessary – and hands it over to the application.

By exchanging Ethernet frames, ETHOM – unlike IP interfaces like IPoIB[9] – directly supports all protocols that rely on Ethernet (like e.g. TIPC) in addition to IP.

As the Dolphin message queue API does not support broadcasting of data to all connected peers, broadcast is implemented in ETHOM. Currently, data is sent to each peer sequentially in a simple Round-Robin fashion.

In case of a node failure or shutdown, all other nodes continue working as before. Reconnection of message queues as soon as a node comes up again is not yet implemented, though.

3 Experimental Results

Measurements were performed on two clusters as SCI and DX cards are built into separate clusters:

(1) The first cluster, called Xeon throughout this paper, consists of two nodes equipped with two Intel Xeon X5355 four-core CPUs running at 2.66 GHz. The mainboard is an Intel S5000PSL with two on-board Gigabit Ethernet controllers (Intel 82563EB). Each node is equipped with a DX adapter from Dolphin (DX510H, 16 Gb/s) in a PCIe x8 slot, and an InfiniBand adapter from Mellanox (MHGS18-XTC DDR, 20 Gb/s), which is plugged into a PCIe x8 slot, too. The DX cards are connected directly without an intermediate switch, while Gigabit Ethernet and InfiniBand use a switch; Cisco Catalyst 2960G-24TC-L (Ethernet) and Mellanox MTS-2400-DDR (InfiniBand).

(2) The second cluster, called PD, consists of 16 nodes, two of wich were used for measurements. Each node features a PentiumD 820 dual-core CPU running at 2.8 GHz. The mainboard is from ASUSTek (P5MT-M). It is equipped with two on-board Gigabit Ethernet controllers (Broadcom BCM5721). In addition to that, each node is equipped with an SCI card from Dolphin (D352, 10 Gb/s), which resides in a PCIe x4 slot, and with an InfiniBand adapter from Mellanox (MHGS18-XTC DDR, 20 Gb/s), which is plugged into a PCIe x8 slot. The SCI cards are connected in a 4x4 torus topology.

All nodes run an unpatched kernel 2.6.22, 64-bit on Xeon and 32-bit on PD. Kernel preemption was enabled. For InfiniBand, we used the drivers included in kernel 2.6.22, for SCI and DX, version 3.3.1d of Dolphin's software stack.

In order to measure the performance of our driver in comparison with hardware drivers for Ethernet and InfiniBand, we measured TCP socket performance, concentrating on latency (see section 3.1) and bandwidth (see section 3.2) for various message sizes. In addition to that, we performed two measurements with TIPC replacing TCP/IP (see section 3.4) in order to see what performance to expect from our approach to enable TIPC over high-speed interconnects.

For the TCP experiments, we used NPtcp from the NetPIPE[10] suite in version 3.7.1, which is described in [10] and Dolphin's sockperf[11] in version 3.3.1d as it records information about interrupt and CPU usage. TIPC performance was measured with tipcbench[12], which is part of the "TIPC demo v1.15 package".

After testing with several different MTU settings, we chose to use the biggest possible MTU for ETHOM, as it does not have a negative effect on latency but proved positive for bandwidth. Apart from that, we did not touch the default settings for the other parameters of our Ethernet, SCI, DX, and InfiniBand NICs (like interrupt coalescing, message coalescing, and any other).

Currently, we use a protocol of the message queues which is restricted to 8 KB, therefore the largest MTU for ETHOM is 8 KB minus header length at the moment. We expect greatly improved bandwidth when usage of a protocol supporting larger messages is implemented.

3.1 Latency

In Figure 5 the round-trip latency (RTT/2) for messages of varying sizes measured with NPtcp is shown.

The green curve represents Gigabit Ethernet, the reference that ETHOM competes with. The lowest latencies are delivered by ETHOM on SCI, followed by ETHOM on DX; the highest times are the Ethernet times. A dramatic decrease in latency can be seen for Ethernet with message sizes between 16 and 48 B, which indicates polling for new messages on the receiving side. For larger messages, the high raw bandwidths of InfiniBand and DX lead to lower latency as for SCI. Comparing ETHOM on SCI with ETHOS on SCI, an improvement in latency of around 10 μs for small messages and around 15 μs for larger ones can be observed.

Summarising, ETHOM on SCI provides an improvement in latency by a factor of two and above on our measure-

[9] IP over InfiniBand

[10] http://www.scl.ameslab.gov/netpipe/

[11] sockperf is part of the Dolphin Driver Package available from http://www.dolphinics.com

[12] http://tipc.sourceforge.net/tipc_linux.html

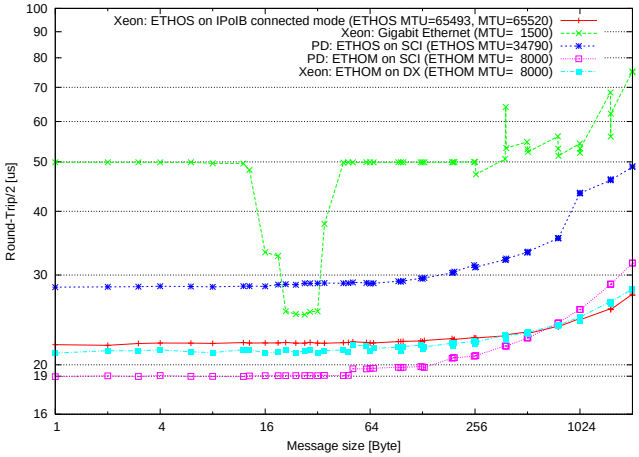

Figure 5: Latency measured with NPtcp

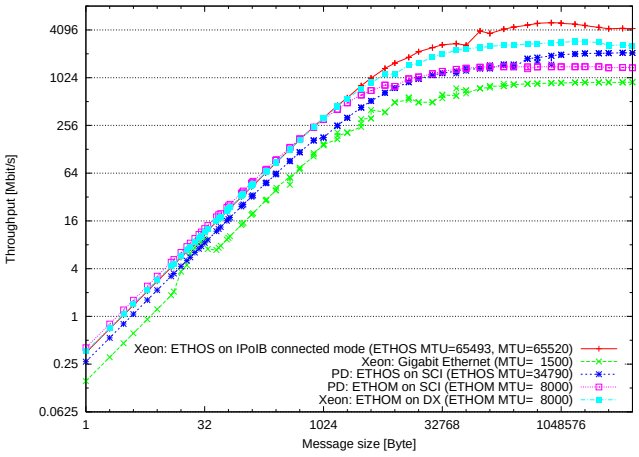

Figure 6: Throughput measured with NPtcp

ment platform over Gigabit Ethernet and about a 30% improvement over its companion ETHOS.

3.2 Bandwidth

Figure 6 shows the bandwidth for varying message sizes measured with NPtcp.

Gigabit Ethernet delivers for all message sizes the lowest bandwidth (excluding the aforementioned interval between 16 and 48 B). For small messages, ETHOM on SCI performs best with ETHOM on DX and ETHOS on IPoIB close by. At about 1 KB, the three curves split again each gradually approaching its maximum, which is at 1.5 Gb/s for SCI and 3 Gb/s for DX (with their current limitation to an MTU of 8 KB) and about 5 Gb/s for ETHOS on InfiniBand. Comparing ETHOM with ETHOS on SCI, it can be noticed that for small messages (until 8 KB) ETHOM provides a 50% increase in bandwidth. For large messages (256 KB and above) ETHOS benefits from the support for larger low-level packets and maybe additional buffering in the sockets layer.

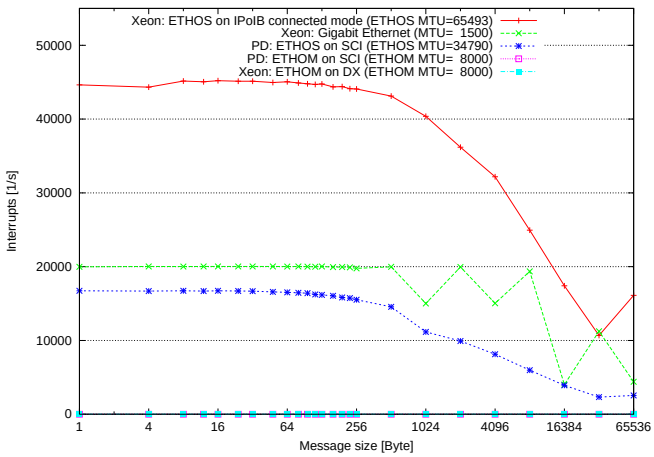

Figure 7: Interrupts measured with sockperf

To sum up, ETHOM on SCI exhibits a twofold increase in bandwidth for messages up to 1 KB over Gigabit Ethernet and about a 50% increase over ETHOS.

3.3 Interrupts and CPU Utilisation

In this section we show the drawbacks of ETHOM, which uses polling and therefore a higher system load to achieve its high performance. Figure 7 and Figure 8 have to be examined together in order to get some meaningful statement. In Figure 7 the number of interrupts triggered by each device are recorded over the message size.

Two points are immediately eye-catching: (1) ETHOM on SCI and DX does not trigger any interrupts, which is obvious considering that ETHOM uses polling mechanisms instead of relying on an interrupt. (2) With 45000 IRQs/s, ETHOS on InfiniBand puts by far the highest load onto the IRQ-processing routines, sharply decreasing with messages bigger than 512 Byte. At a second glance, it can be seen that our Ethernet adapter is limited to 20000 IRQs/s, which is a clear indication of coalescing intermediate interrupts.

As mentioned before, we have to keep the interrupts in mind when discussing the system load. Figure 8 shows only the system time and not the time needed for IRQ processing. Several aspects are worth mentioning here: First of all, ETHOM on SCI has a very high system load, as one thread is constantly polling for new messages, effectively occupying one of the 2 cores. The curve for ETHOM on DX, which is measured on the 8-core Xeon system, indicates that a multi-core platform is a much better basis for our polling approach and alleviates the high load. The system load amounts to 12.5%, which again reflects one core fully occupied with polling, and there is very low IRQ load to be expected as no interrupt is triggered by the DX adapter.

The next point worth mentioning is the very low system time for Gigabit Ethernet and ETHOS on InfiniBand. On

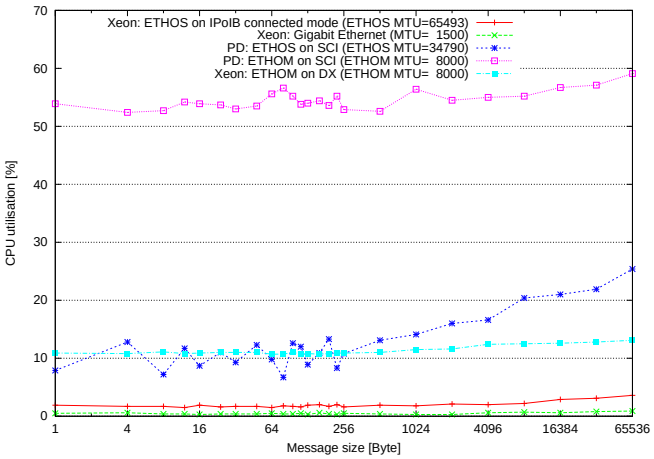

Figure 8: Systime measured with sockperf

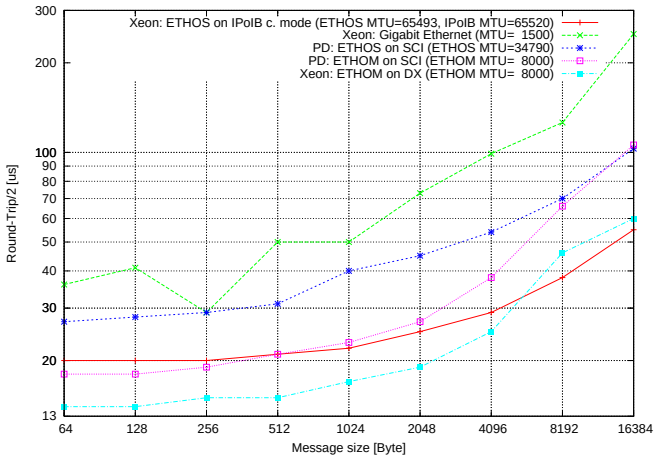

Figure 9: Latency measured with tipcbench

InfiniBand, due to the high interrupt rate, non-negligible time will be spend in the interrupt routines, but on Ethernet, the overall system load in general is very low.

The unsteadiness of the ETHOS on SCI curve seems to stem from changing between polling and interrupt mode in Dolphin's SCI sockets, which ETHOS uses.

System utilisation can be summarised with very low load for Gigabit Ethernet, moderate load for ETHOS on SCI and ETHOS on InfiniBand. At the other end is ETHOM, which causes a very high load. ETHOM will clearly benefit very much from upcoming *many cores*[13].

3.4 TIPC Benchmarks

As one of the main points in our motivation was to speed up communication over TIPC, we finally measured performance of Ethernet, SCI, DX, and InfiniBand with the TIPC protocol replacing TCP/IP.

Latency In Figure 9, the latency for varying message sizes measured with tipcbench is depicted.

With the TIPC protocol, ETHOM on DX reaches the lowest latencies we ever measured, amounting to 14 μs. ETHOM on SCI lies at 19 μs for small messages. Gigabit Ethernet has the highest latencies for all message sizes. ETHOS on InfiniBand has a latency of 20 μs for small messages, starting from 8 KB it has the lowest latency of the measured interconnects.

Comparing ETHOM and ETHOS on SCI, a decrease in latency of between 10 and 20 μs can be observed for messages smaller than 8 KB. For 8 KB and above the bigger packet size supported by ETHOS leads to a latency which is on par with ETHOM.

[13]the term used for high number of cores per CPU die

Bandwidth Figure 10 shows the bandwidth for varying message sizes measured with tipcbench. Several facts are interesting about this measurement: (1) Gigabit Ethernet provides the best throughput for small messages, has a significant decrease at a message size of 4 KB and reaches a maximum of clearly below 500 Mb/s. It is not clear to us why Ethernet performs so well for small messages in this test, but we suspect that this is influenced by polling and very efficient buffering. The fact that the maximum throughput is rather low indicates that the strength of the TIPC protocol lies in low latency rather than a high bandwidth. (2) The second point that catches the eye is the severe decrease in bandwidth of ETHOM on DX for messages of 8 KB and above. Part of the problem should be related to the limitation of 8 KB for packets that can in our implementation currently be sent via message queues. Another part seems to be the way that ETHOM deals with send failures. As this problem does not appear with NPtcp, we suspect part of the problem in the way TIPC uses the ETH interface or in the way tipcbench works. We are still investigating this issue and hope for improvement when we get rid of the "8 KB limit".

Comparing ETHOM and ETHOS on SCI, we see an improvement in bandwidth for practically all message sizes. An increase for message sizes above 8 KB is expected after modification of ETHOM.

Summarising the TIPC benchmarks, we record that ETHOM on SCI and DX shows a very low latency. Compared with Ethernet, it provides a decrease by the factor 2 to 3 for small messages and up to 4 at 4 KB. Concerning bandwidth, Ethernet performs surprisingly well for small messages. Tipcbench and the effects with ETHOM on DX have to be studied more thoroughly.

ISBN: 978-988-17012-5-1

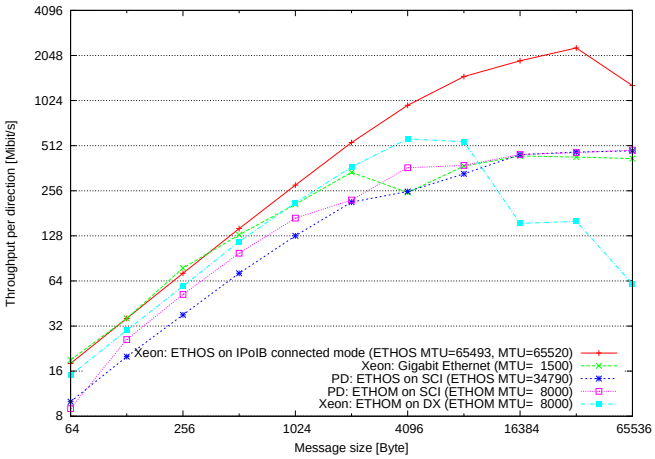

Figure 10: Bandwidth measured with tipcbench

4 Conclusions and Outlook

The tests performed within the scope of this article show that ETHOM – making use of a high-speed interconnect like either SCI or Dolphin DX – is a solution that offers better performance than Gigabit Ethernet, latency-wise and bandwidth-wise. Regarding the completely different price range of Gigabit Ethernet and high-speed interconnects, this comparison is only reasonable, when an Ethernet interface is required.

Comparing the results with ETHOS [9], which implemented an Ethernet interface using kernel-level sockets as its lower interface, we observe a 30%-70% improvement in bandwidth for small to medium packages and about a 30% decrease in latency, when SCI is used.

The advent of *many cores* should have a twofold positive effect: (1) The network should become an even bigger bottleneck for communicating applications, as the connection is shared by a bigger number of cores, so better communication performance is highly appreciated. (2) Having a smaller ratio between the one core sacrificed for communication and the number of cores still available for computation reduces the relative communication overhead.

With ETHOM, we present a driver for Linux, that makes effective use of the lowest message passing layer of the Dolphin software stack. Processing Ethernet frames from the layer above, it enables a potentially wide range of software to make use of Dolphin's high-speed networks. It has a low overhead and is small with about 1000 lines of code.

Currently, ETHOM fulfills our main aim to enable TIPC – and any other software communicating via Ethernet frames – to use SCI and DX. Besides ETHOS, it provides the only Ethernet interface for SCI and DX; as a side effect, support for IP-routing is now offered using the standard kernel IP stack on top of ETHOM.

On the other hand side, porting software to the native interfaces of high-speed interconnects almost always provides better performance and efficiency at runtime – obviously at the cost of porting effort. As usual, it remains to the user to balance the pros and cons.

Having succeeded to let TIPC run on top of SCI and DX, our next goal is to sacrifice the compatibility to Ethernet and design a native TIPC bearer for SCI and DX. This way, we hope to further improve performance. Apart from that, the effect of the performance improvement onto applications and higher-level functionality will be studied.

References

[1] V. Krishnan, "Towards an Integrated IO and Clustering Solution using PCI Express," in *IEEE Cluster*, (Austin, Texas), Sept. 2007.

[2] S. Fu and M. Atiquzzaman, "SCTP: state of the art in research, products, and technical challenges," in *Computer Communications, 2003. CCW 2003. Proceedings. 2003 IEEE 18th Annual Workshop on*, pp. 85–91, 2003.

[3] E. Kohler, M. Handley, and S. Floyd, "Datagram Congestion Control Protocol (DCCP)." `http://ietfreport.isoc.org/rfc/PDF/rfc4340.pdf`, 2006.

[4] L.-A. Larzon, M. Degermark, S. Pink, L.-E. Jonsson, and G. Fairhurst, "The Lightweight User Datagram Protocol (UDP-Lite)." `http://ietfreport.isoc.org/rfc/PDF/rfc3828.pdf`, 2004.

[5] S. Hopkins and B. Coile, "AoE (ATA over Ethernet)." `http://www.coraid.com/site/co-pdfs/AoEr10.pdf`, 2006.

[6] J. Maloy, "TIPC: Providing Communication for Linux Clusters," in *Proceedings of the Ottawa Linux Symposium*, pp. 347–356, 2004. `http://www.linuxsymposium.org/proceedings/LinuxSymposium2004_V2.pdf`.

[7] A. Stephens, J. Maloy, and E. Horvath, "TIPC Programmer's Guide." `http://tipc.sourceforge.net/doc/tipc_1.7_prog_guide.pdf`, 2008.

[8] The Kerrighed Team, "Kerrighed: a Single System Image operating system for clusters." `http://www.kerrighed.org`, 2008.

[9] R. Finocchiaro, L. Razik, S. Lankes, and T. Bemmerl, "ETHOS, a generic Ethernet over Sockets Driver for Linux," in *Proceedings of the 20th International Conference on Parallel and Distributed Computing and Systems (PDCS)*, 2008.

[10] Q. Snell, A. Mikler, and J. Gustafson, "Netpipe: A network protocol independent performace evaluator," in *In Proceedings of the IASTED International Conference on Intelligent Information Management and Systems*, 1996. `http://www.scl.ameslab.gov/netpipe/paper/full.html`.