40 Gigabit Ethernet: Prototyping Transparent End-to-End Connectivity

Cosmin Dumitru **Ralph Koning**
Cees De Laat

University of Amsterdam, Science Park 904, 1098XH
{c.dumitru,r.koning,delaat}@uva.nl

Abstract

The ever increasing demands of data intensive eScience applications have pushed the limits of computer networks. With the launch of the new 40 Gigabit Ethernet(40GE) standard, 802.3ba, applications can go beyond the common 10 Gigabit/s per data stream barrier for both local area, and as demonstrated in the GLIF 2010 and Supercomputing 2010 demos[3], wide area setups. In this article we profile the performance of state-of-the-art server hardware combined with 40GE technology. We give an insight on the issues involved with ultra high performance network adapters and suggest optimization approaches.

Keywords: 40G Ethernet, Optical Networks, Computer Architecture

1 INTRODUCTION

The dual 40/100 Gigabit Standard 802.3ba [5] is the next evolution of the Ethernet standard. It increases the maximum speed at which Ethernet frames can be transmitted and defines new physical (PHY) standards for the transport of data over copper or optical media. It was was approved in its final state in July 2010 with vendors announcing hardware that implements it (and its drafts) in September 2009. In Q3 of 2010, 40GE client adapters became available in limited supply. At the same time several vendors started offering optical switch modules or development platforms.

Given the promised four fold increase from 10Gigabit Ethernet (10GE), 40GE puts extra stress on the underling hardware because it is able, as it will be presented in this article, to saturate the PCI Express computer interface. Other technologies, like Infiniband Quad Data Rate(QDR), already provide comparable speeds but they are limited in use to the datacenter premises. Multiple 10GE network interface cards can be bonded in a single logical Ethernet interface using the IEEE 802.3ad Link Aggregation standard but even so, the single stream maximum rate is limited to 10Gbit/s. This is because, in order to avoid frame reordering, frames intended for one receiver are always sent on the same physical interface.

In the last years, computer architectures shifted from the single core CPUs to multicore. Together with this shift, CPU manufacturers started integrating a local memory controller on the CPU die in order to increase the available bandwidth to the memory. This led to the situation where each CPU has its own local memory and any access outside the local memory is done through another CPU. Essentially, this means that any modern multi socket x86 machine uses the Non Uniform Memory Architecture (NUMA).

2 EXPERIMENTAL SETUP

Figure 1: Server Hardware Specifications

Server Model	Supermicro H8DTT-HIBQF	Dell R815
CPU Model	Intel Xeon E5620 2.4GHz	AMD Opteron6172 2.1GHz
Core Count	2 x 4 cores	4 x 12 cores
RAM	24GB	128 GB

In this section we will describe the setup used during our experiments. Two models of servers were used, both employing a form of NUMA: an OEM server based on the Supermicro H8DTT-HIBQF motherboard and a Dell R815. The hardware specifications of the servers are displayed in figure 1. In the rest of this article we will refer to the the platforms as Intel Nehalem and AMD Magny Cours.

Server Architecture Both server models support the PCI-E Gen 2.0 peripheral interface. Besides the CPU frequency, core count and RAM memory, one very important architectural aspect is the way the CPUs are connected to each other and to the I/O Hubs or PCI-E bridges. While the Intel Nehalem motherboard uses only one chip to connect the CPUs to the peripherals, the AMD Magny Cours uses two chips allowing more

Figure 2: Simplified view of the AMD Magny Cours IO Architecture - Dual HexaCore Quad Socket

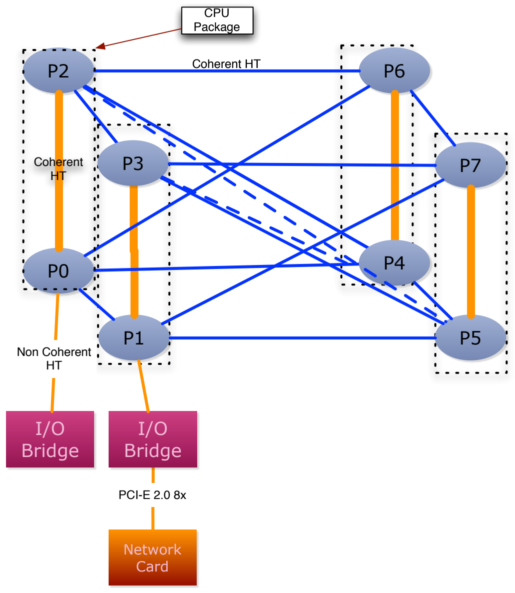
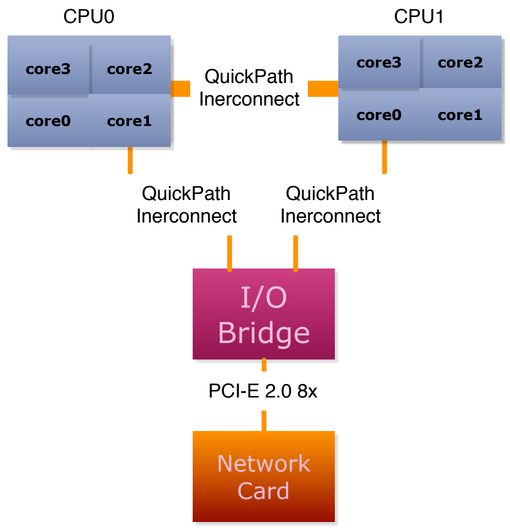


Figure 3: Simplified view of the Intel Nehalem IO Architecture - Quad Core Dual Socket



PCI-E devices to be connected to the machine. The dual chip approach increases the available bandwidth, yet depending on the chosen topology, it extends the path from the NUMA nodes to the peripherals.

The AMD Magny Cours platform employes 48 CPU cores. The cores are grouped into 4 CPUs or packages, each package having two six-core CPU modules that share a common L3 cache. From an NUMA point of view, a package is equivalent to two NUMA nodes. The HyperTransport interconnect enables communication between the nodes or PCI-E bridges. A more in-depth description of the platform can be found in [2], where the authors present possible CPU I/O topologies, each suited for either I/O performance or low latency (small diameter of the CPU network) the latter being the recommended setup for server manufacturers. We assume that the I/O topology of the AMD Magny Cours platform used is the low latency variant presented in figure 2 . Two of the CPU packages are not connected directly to the PCI-E bridges but through the other two CPU packages - in figure 2 P5-P7 and P4-P6.

The Intel Nehalem platform[1] used had a lower core density, only 8 cores, provided by two quad-core CPUs, each CPU representing one NUMA node . Although the HyperThreading feature was available, it was disabled during all the tests. Inter-node (CPU) communication and PCI-E connectivity to the CPUs is provided by the the Quick Path Interconnect (QPI) interface (figure 3).

Connectivity For the connectivity we used two Mellanox ConnectX2 40GE NICs (firmware version 2.7.700), connected back to back with an optical fiber cable. The cable is comprised of two QSPF+ optical modules and a 12 pair MPO fiber (SR4 standard). In the SR4 standard, four multimode fibers are used for the RX side and 4 multimode fibers for the TX side. While on the logical level there is only one 40GE interface available to the server, the QSPF+ optical module has the function to convert the electrical signal sent by the network card to four 10Gbit/s optical signals which are sent over the four fiber strands. The PHY component of the 40GE standard specifies the way in which the four signals are synchronized and demultiplexed on the receiving side. A detailed description of the PHY component of the 802.3ba is presented in [5] . The cards use the PCI-Express Gen 2.0 x8 peripheral interface which supports a maximum of 40Gigabit/s raw transfer rate. Due to the 8/10bit encoding used for the PCI-E gen 2.0 protocol, the available data bandwidth is 32Gbit/s. The real available bandwidth is further decreased by the overhead induced by the PCI-E protocol.

Multi queue support [10], was enabled on the Mellanox cards by default. The mlx4_en driver creates a number of receive (RX) queues or rings that is equal to the number of online CPUs (or cores). When a network packet is received it is placed in a RX queue and the network card triggers an interrupt signaling the corresponding CPU to handle the incoming data. Each incoming IP packet is placed in the appropriate RX queue according to an algorithm that hashes the IP information in the header of the packet. The result of this is that IP flow will be always handled in-order and by the same core and also multiple flows will be distributed in a fair manner to the available cores. A similar feature

Figure 4: Local iperf tests - single and multiflow

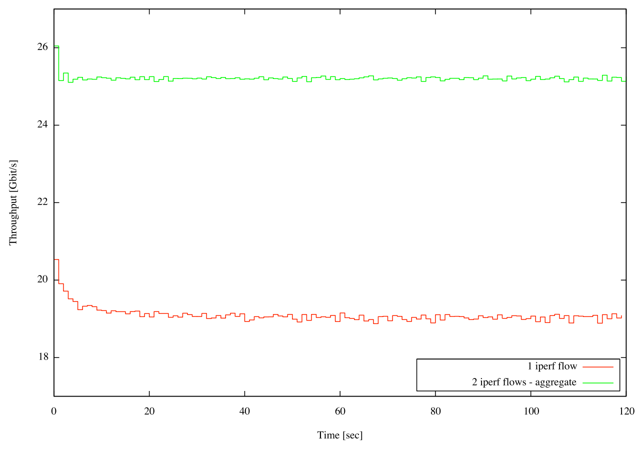
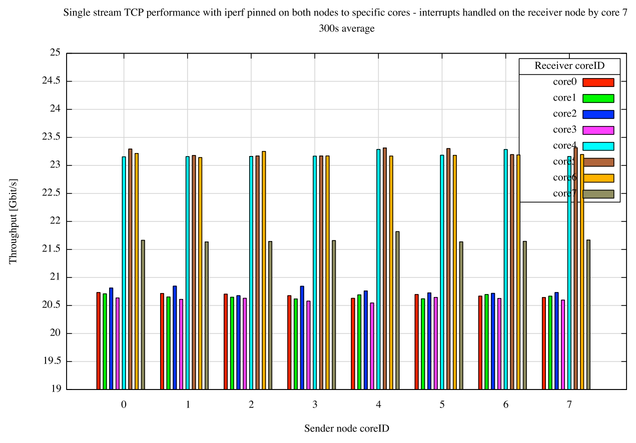


Figure 5: Average throughput of iperf with core pinning



is used for the transmission part, the driver creating a number of transmission(TX) queues that each handle one IP flow at the sender side.

Software All experiments were performed using iperf 2.0.5, a popular and well established bandwidth testing tool. The kernel used was vanilla Linux 2.6.38 compiled with the same options for both platforms, the only difference being the 'Processor Family' option which was set according to the underlying hardware.

The observed latency between the sender and receiver was approximately 0.15ms as reported by the ICMP ping utility.

3 MEASUREMENTS

First we focused on the performance of the network cards. In figure 4 we present the results of measurements using two Intel Nehalem machines connected by 40GE. We present single and multi TCP flow aggregate. As mentioned in section 1, the maximum theoretical 32Gbit/s data rate is further decreased by the inherent overhead of the PCI-E protocol and of the network protocols (Ethernet, TCP/IP) that support the communication. Therefore, the application transfer rate of 25.3Gbit/s observed using two TCP streams and comes close to the theoretical maximum. We conclude that in our tests, the PCI-E interface becomes saturated . This is the maximum rate at which the card can send or receive data.

The focus of our measurements was single TCP flow performance. If the source port is not specified by an application, the Linux kernel picks a random one from the ephemeral port range. This is the case of the `iperf` utility. Because the multi queue receive and transmit (also known as Linux scalable I/O[8]) mechanism uses a hash based on the IP header (source and destination, IP and port) we patched the `iperf` source code so that for any single flow run, the same source port would be used. This assured that a TCP flow would always be handled by the same core.

Intel Nehalem The single stream TCP performance throughput for the Intel Nehalem server oscillated during the tests between 20.7Gbit/s and 23.3Gbit/s. During consecutive runs the parameters of the setup didn't change, yet the results of the measurements varied considerably. We suspected that the cause of this behavior was due to the NUMA characteristics of the machine. The asymmetric network performance of NUMA machines has been investigated in the past, yet no research exists on newer technology. We connected two Intel servers back to back using 40GE. Using the `numactl` utility we pinned the patched `iperf` process on each of the 8 cores on the sender and receiver. This resulted in 64 measurements that cover all the possible combinations of pinned sender and receiving core. Each measurement was done over 5 minutes. In figure 5 we present the averaged results of these measurement. We inspected the interrupt counters, exposed by the Linux kernel via the `procfs` interface and we determined that for our given setup, the interrupts were handled by core 7. Because of this, we observe slightly lower throughput when the `iperf` server process is pinned to this core. The core to CPU mapping is more simpler and more intuitive than in the AMD case: the first four cores belong to CPU0 while the remaining four to CPU1. When the application runs on the same CPU as the core that handles the interrupts the performance increases by approximately 2Gigbit/s. Not surprisingly, the same performance increase is not achieved when `iperf` and the interrupts are handled by the same core, the throughput increasing with only 1Gigabit/s. The location of the application on the sender node does not influence the performance in a very significant way and therefore

Figure 6: Simple experimental setup to determine I/O Performance - Dell AMD

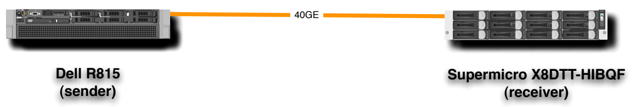
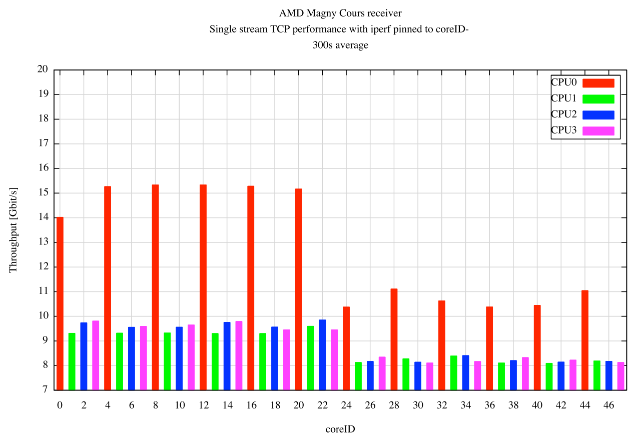


Figure 7: Average throughput of iperf with core pinning - AMD receiver



we conclude that the limitation is at the receiver sides.

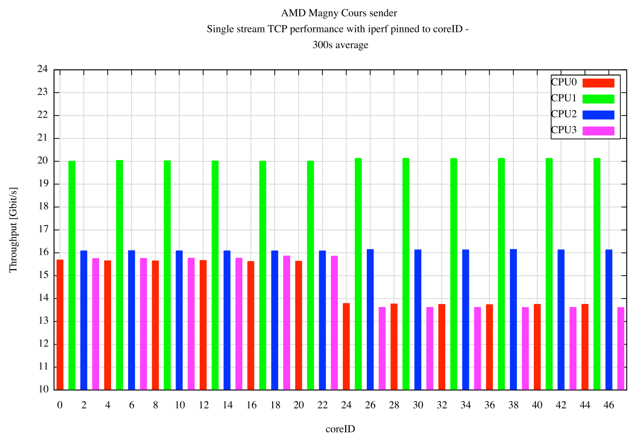
AMD Magny Cours In our tests the AMD Magny Cours was unable to reach the same I/O performance as the Intel Nehalem server even when using `iperf` with multiple threads.

In comparison with the Intel Nehalem server, the AMD Magny Cours has a more complicated topology and even more it is asymmetric, not all CPU sockets being directly connected to the I/O bridges. The mapping of the cores to CPU packages on our setup followed the rule: if one core has id i then it is located on package $i \pmod 4$, so for example core 12 is located on package 0. The first six cores from one CPU formed the first NUMA node while the remaining six formed the second node.

To explore the asymmetry and see its impact on I/O performance, we performed a similar experiment where an Intel Nehalem server and an AMD Magnuy Cours server were connected back to back using 40GE. The servers were configured both as an `iperf` server (receiver) and as a single threaded sender with the `iperf` process forced to run on a specific core (figure 6). The fact that the Intel server was able to send and receive receive data at rates higher than the AMD server, allowed us to change parameters on the AMD server without worrying that the other party would not be able to cope with the incoming data. The iperf process on the Intel Nehalem server was pinned to the same core for all the tests.

The measurements were repeated for each of the 48 cores available for a period of 5 minutes with the AMD server being in turns, sender and receiver. By analyzing the results of the average per core throughput(figures 7, 8) we can observe a pattern that indicates that some cores perform significantly better than others. When mapping the coreID over the given CPU topology, a correlation between the core location, relative to the CPU package, and I/O performance stands out. The graph clearly shows that even for single threaded applications which do heavy I/O traffic, the placement of the process on specific cores heavily impacts the overall performance. In a multithreaded setting this affinity issue is overcome by the overall interconnect bandwidth to the PCI-E bridge. Our tests showed a maximum of 20Gbit/s when running `iperf` with multiple threads(TCP streams). In our setup CPU1 has the best connection to the I/O bridge. In the case of the sender there is almost no difference between cores located on CPU1 but on different nodes, the average throughput being close to 20Gbit/s. This changes dramatically in the case when the AMD Magny Cours is the receiver. The interrupts are handled by core0, and the cores located on the same node as core0, exhibit the best performance. Unlike in the case of the sender, the cores located on the other NUMA node, but still on the same package, suffer from a performance hit, losing 4Gbit/s in throughput.

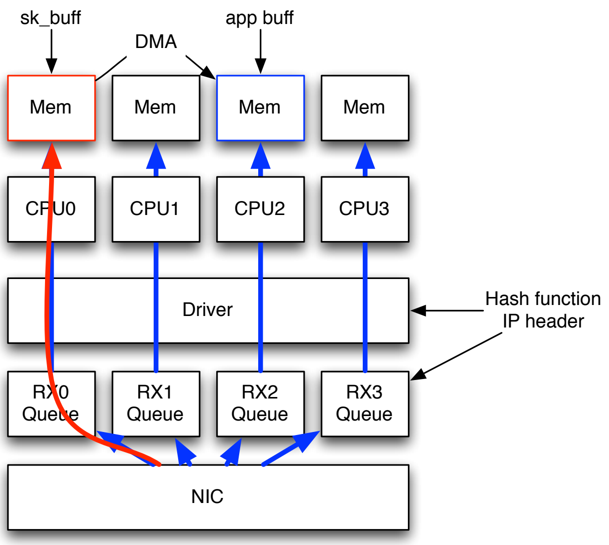
Figure 8: Average throughput of iperf with core pinning - AMD sender



4 DISCUSSION

In the case of the AMD server, the receiver can handle throughput higher than 20Gbit/s, as shown in section 3 and therefore, we believe that the inconsistent performance is caused by the limitations of the memory architecture used. This means that for single threaded

Figure 9: Simplified view of the Linux network stack - receiver side



`iperf` process, the bottle neck lies closer to the CPU cores. In terms of interconnect bandwidth, the Hypertransport links are not equally shared between the cores[2] and even if the AMD Magny Cours has a novel cache coherency mechanism, we can expect that still some of the traffic is cache coherency related, leaving less available bandwidth for data intensive applications. Unfortunately, proving this hypothesis is beyond the scope of this paper yet in [7] the authors suggest that cache coherency can have an important impact on network traffic.

A deeper understanding of the processes that take place in the kernel on the receiver side was needed and therefore, we decided to inspect the mlx4_en driver source code [6]. As mentioned before the driver has multiqueue support and by default it creates a number of receive queues equal to the number of online CPUs, as seen by the kernel. While the driver is more complex and has support for advanced features like TCP offloading and Generic Segmentation Offload (GSO), we will not go into the details of these. The receive procedure follows a standard NAPI network driver approach[4]: a socket buffer structure, also named `skb`, is pre allocated in order to handle the incoming data from the network. When the data is received from the network, it is buffered on the network card and the CPU is notified via an interrupt about its arrival. The CPU then polls the network card and issues a DMA transfer from the network card to the memory address stored in the `skb` structure. From here on, the kernel decides what to do with the received packet, as it gets passed to the upper layers of the networking stack. In the case of a packet destined to a local application, the payload is copied to the socket buffer belonging to the application (figure 9). This means that the data is copied at least twice from the moment it is received by the network adapter, once to from the network card to kernel space and from there to user space (for the sake of simplicity we do not we do not take into consideration any fragmentation or retransmit TCP/IP buffers, even though those exist). In the case of a NUMA machine the `skb` structure is allocated on the local node, the one that handles the receive queue from which the data originates. Obviously, if the application is running on a different node an inter-node transfer is needed and the topology, the memory and interconnect bandwidth and latency affect the overall performance. Our measurements clearly show degraded performance when internode transfer is involved.

In our tests we assumed that the Mellanox ConnectX2 40GE drivers have an optimal behavior and their performance is not affected in any way by the underlying architecture. Interestingly, changing the kernel from the 2.6.18 (shipped by default with Centos 5.5, the Linux distributuion used during the measurements, driver version 1.4.2.2) to 2.6.38 (driver version 1.5.1.6) increased the throughput by almost 2Gbit/s in the case of the AMD Magny Cours server. This shows that there is some room for improvement in driver performance.

There is no standard way to map a TCP stream to a core close to an application. While tools that provide this feature exist, like Intel Flow Director, they are specific to certain type of hardware[9]. The network stack is currently unable to derive the best core-to-receive queue mapping. The `mlx4_en` driver offers the `rss_mask` parameter that can alter the parameters used for the hashing functions. In this way either of the components of the IP headers (source& destination IP and port) can be ignored during the hash calculation. Even so, this only option offers a very crude way of traffic steering because the outcome of the mapping is still not straight forward.

We believe that our measurements give a good insight on the capabilities of the current server hardware when faced with network I/O intensive applications. Given the current state of the networking stack on the Linux OS, it is very probable that similar performance inconsistencies will occur with future multicore NUMA architectures and next-gen network adapters and the approach presented in this paper could be used to investigate them.

5 CONCLUSIONS

In this report we have presented an in-depth analysis of the performance of 40GE when combined current state-of-the-art server hardware. We conclude that given the current PCI-E 2.0 interface and CPU microachitectures the 40GE standard is not yet used to its full potential. A modern server can not fully utilize the available bandwidth and while it can saturate the I/O bus this leaves little room for a real application running on the machine. When using a more capable machine the I/O limitations stand out even more giving 40GE little advantage over other existing technologies. With the introduction of the new PCI-E 3.0 computer interface in late 2010 together with faster CPUs, we expect that the full potential of 40GE will be unleashed in terms of aggregated throughput. With the current available technology, the bottleneck has moved one level higher, from the network to the computer's internal interconnect. At the moment, the number of applications that can leverage this bandwidth increase is still limited. New research and applications are needed to promote 40GE from an exotic network protocol to a commodity interconnect.

Acknowledgments This work was made possible by financial support from the GigaPort 2011 Research on Networks project. The CineGrid Amsterdam consortium gratefully acknowledges the generous financial support from the European Fund for Regional Development (EFRO), Pieken in de Delta, the Province of Noord-Holland and the City of Amsterdam.

This article would have not been possible without the support of : *Erik-Jan Bos, Gerben van Malenstein, Roeland Nuijts* - **SURFnet**, *David Yeung, Jan-Willem Elion, Harry Peng, Kevin McKernan, Martin Bluethner, Rodney Wilson* - **CIENA**, *Kees Verstoep, Henri Bal* -**Vrije Universiteit Amsterdam**, *Erez Cohen, Bill Lee* - **Mellanox**.

References

[1] Casazza, J. First the tick, now the tock: Intel microarchitecture (nehalem). *Intel Corporation* (2009).

[2] Conway, P., Kalyanasundharam, N., Donley, G., Lepak, K., and Hughes, B. Cache Hierarchy and Memory Subsystem of the AMD Opteron Processor. *IEEE Micro 30*, 2 (Mar. 2010), 16–29.

[3] Dumitru, C., Koning, R., and de Laat, C. Clearstream: Prototyping 40 gbps transparent end-to-end connectivity.

[4] Foundation, L. napi article on linuxfoundation.org. `http://www.linuxfoundation.org/collaborate/workgroups/networking/napi`. [Online; accessed 20-February-2011].

[5] IEEE. Ieee std 802.3ba-2010 (amendment to ieee standard 802.3-2008). 1 –457.

[6] Inc., M. Mellanox mlx4 en driver source code hosted on lxr.linux.no. `http://lxr.linux.no/linux+v2.6.37.1/drivers/net/mlx4/`. [Online; accessed 20-February-2011].

[7] Kumar, A., and Huggahalli, R. Impact of cache coherence protocols on the processing of network traffic. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture* (Washington, DC, USA, 2007), MICRO 40, IEEE Computer Society, pp. 161–171.

[8] Provos, N., Lever, C., and Alliance, S. Scalable network I/O in Linux. In *Proceedings of the USENIX Annual Technical Conference, FREENIX Track* (2000), vol. 19.

[9] Wu, W., DeMar, P., and Crawford, M. Why can some advanced ethernet nics cause packet reordering? *Communications Letters, IEEE 15*, 2 (2011), 253 –255.

[10] Yi, Z., and Waskiewicz, P. Enabling Linux network support of hardware multiqueue devices. In *Proc. of the 2007 Linux Symposium*, pp. 305–310.