Improving Packet Processing Efficiency on Multi-core Architectures with Single Input Queue

Péter Orosz
University of Debrecen/Dept. of Informatics Systems and Networks, Debrecen, Hungary
orosz.peter@inf.unideb.hu

***Abstract*—Generic purpose multi-core PC architectures are facing performance challenges of high rate packet reception on gigabit per second and higher speed network interfaces. In order to assign a CPU core to a networking softIRQ, the single input queue design of the low-level packet processing subsystem relies on the kernel's Symmetric Multiprocessing (SMP) scheduler, which does not perform load balancing of the softIRQ instances between the CPU cores. In practice, when single receive queue is used all of the softIRQs are assigned to a single CPU core. This typical arrangement could easily drive to CPU resource exhaustion and high packet loss ratio on high bandwidth interfaces. The non-steady state of the system is triggered by the high arrival rate of the packets. This work presents a proposal for improving the packet processing efficiency in single input queue multi-core systems.**

I. INTRODUCTION

Both the CPU architectures and the networking technologies passed through a dramatic development in the last decade. By the advent of x86-based multi-core CPU architectures, the processing power of the generic purpose desktop and server PCs was significantly increased. The multi-core design replaced the frequency rush previously generated between the CPU vendors in the single core era. Concurrently, the Local Area Network (LAN) technologies have been enhanced up to gigabit per second and higher bandwidths. The fast evolution of the transmission technologies pushed the PCI bus to legacy state to give place to the novel PCI Express technology. The heaviest network load easily generates an excessive CPU load on a single core system due to the high number of interrupts per second [1][2][3]. New packet processing techniques were developed focusing on adaptive processing mechanisms, which provide low latency for real-time traffic and high processing capacity with more efficient CPU utilization for bulk type traffic [4]. In contrast, any execution of the packet processing bottom-half handler (softIRQ) in a multi-core design can be scheduled to and performed on an arbitrary CPU core, theoretically [4]. Since the softIRQ design is reentrant, several instances of a single softIRQ (*NET_RX_SOFTIRQ*, notably) can be concurrently executed even on different cores [1]. Modern operating systems also require significant optimization in this field. In practice, most of the high speed NIC hardware and driver implementations do not support multiple input queues, which is the most important prerequisite of parallel processing of the incoming traffic (Fig. 1). In a single input queue system, the packet processing softIRQ could not be executed concurrently on different cores due to locking problems. The execution of the *NET_RX_SOFTIRQ* is therefore serialized to one CPU core [5]. By default, the scheduler of the operating system assigns *core0* to all of the implemented softIRQ types. Accordingly, all of the softIRQ types and all of their instances are sharing the same core. In an optimal case, the NIC design implements multiple input queues to achieve the dedicated queue to each CPU core layout. Multiple input queues enable to efficiently process incoming packets on per-flow basis. However, when one large data flow is present in the networking subsystem only, the same limitation appears that previously described on the single core design.

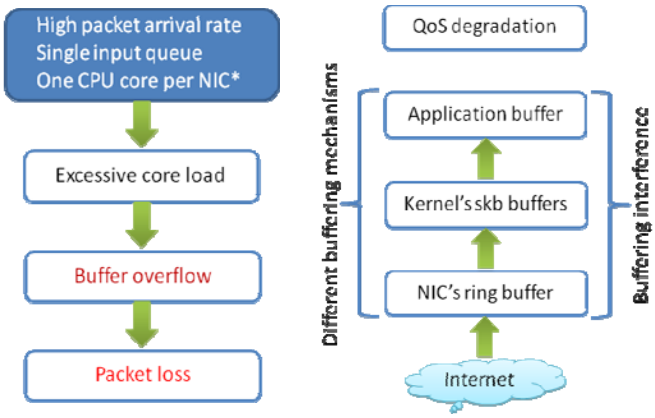

Figure 1. Bottlenecks of packet processing performance at high traffic intensity.

Using a single input queue softIRQ instances are not distributable between CPU cores (Table 1).

TABLE I.
SOFTIRQ PARALLELIZATION

Single input queue	Multiple input queue
one CPU core per NIC	one CPU core per process

II. RELATED WORK

Yunchun et al. proposed a packet processing model to analyze processing characteristics in multi-core systems [6]. They also studied how the load-balancing of the computing resources affect the performance. Chun-Ying et al. investigated the performance bottleneck of the in-line packet processing and proposed

new software architecture called Fast Queue to improve the overall system throughput [7]. An extensive investigation was executed by Hyun-Wook et al. to show the impact of system configurations on the network performance [8]. The series of experiments was performed with multi-port 1 Gbps and single port 10 Gbps comparisons. They revealed that processing the packets on multiple cores can result in higher resource consumption without any important benefit. The root cause of the performance degradation is the lower efficiency of L1 and L2 caching. To answer this problem Hye-Churn et al. proposed a new scheduling scheme called MiAMI, which provides multi-core aware process scheduling over multiple network interfaces [9]. MiAMI determines the optimal core affinity based on the processor cache layout, traffic intensity, and CPU core loads. Nevertheless, none of the mentioned proposals give solution to improve performance in multi-core systems with single input queue.

III. The Proposed Method

A. Overview of the packet processing method

The primary goal of proposed method is to optimize the packet processing performance and improve application QoS on generic purpose x64-based multi-core Linux systems with high performance NIC and a single input queue, when large, intensive traffic is present.

By continuously monitoring the core level CPU load, the statistical analysis of the measured data assists to determine the lowest loaded core, which will be dedicated to *NET_RX_SOFTIRQ* execution, exclusively. This step affects the core affinity of *NET_RX_SOFTIRQ*, which is set to 0x00FF for all softIRQ types by default. In the default configuration, the SMP scheduler of the kernel will not distribute the execution of the different softIRQ types between the cores. Since the execution of a softIRQ is interruptible and multiple softIRQ types share the same core, the execution time may provide a large variance, which results on poor jitter performance with real-time applications. The purpose of the load analysis is to set optimal CPU affinity to the NIC's softIRQ and to minimize the variance of the execution time of the *NET_RX_SOFTIRQ*. Further degradation of this variance can be reached by introducing another optimization step: by disabling hardware interrupts on the dedicated core, the execution of the softIRQ will not be suspended (Fig. 2).

B. Optimization steps

a) Optimization of the buffers within the packet processing subsystem

This step involves a series of calculation for optimal buffer sizes in order to avoid buffering interference and packet loss within the processing path. NIC's ring buffer length has been recalculated based on the arrival rate of the incoming packets. Kernel's *net.core.wmem_** parameters have been determined according to (1).

$$2^k x(BxD_{oneway}) \quad (1)$$

where $k \geq 1$ and B denotes the physical bandwidth of the link and D is the one-way delay between the communication endpoints.

b) CPU core affinity for the NET_RX_SOFTIRQ

Determining and setting the optimal affinity of the *NET_RX_SOFTIRQ* packet processing softIRQ is the second phase of the initial setup, which has been performed by monitoring the CPU usage for a predefined time on per-core basis. A zero loaded core will be selected and assigned to the softIRQ. The monitoring is implemented via a shell script that retrieves the per-core load parameters in every second. The affinity of the *NET_RX_SOFTIRQ* is set by

```
# echo {8-bit core mask} >
/proc/irq/XX/smp_affinity
```

c) Dynamically disabling interrupts on the selected core

The purpose of this step is to eliminate large overhead and its large variance of the softIRQ execution. By disabling interrupts on the selected core, the execution of the softIRQ will be not suspended, which enables the CPU to load the instructions directly from its own L1 cache. Since the softIRQ is serialized to one core, this method results on lower execution overhead and variance as shown in Section IV. The logical diagram of the dynamic interrupt disabling is presented on Figure 2.

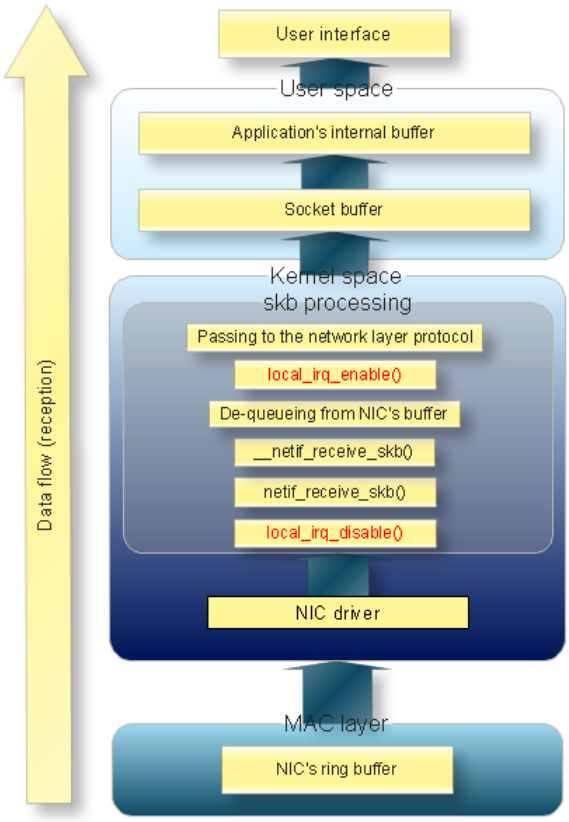

Figure 2. Interrupts are disabled on the dedicated CPU core while it executes the packet processing bottom-half handler.

IV. EVALUATION OF THE METHOD

A. Measurement setup

Synthesized traffic flows were generated with very high precision by a custom device based on a Xilinx Virtex-4 FX12 FPGA and transmitted on a Gigabit Ethernet link directly towards the measurement workstation (see Fig. 3). The proposed optimization method, as described in Section III, was implemented within preemptive as well as non-preemptive Linux kernels (v2.6.39.2). The hardware architecture of the measurement PC was based on an Intel Core i7 870 processor with four hyper-threaded cores. Each core has its own L1 and L2 caches, while the processor cores share the same L3 cache. This layout is a key factor when the softIRQ execution times are analyzed on different logical configurations [8]. An Intel 1000/PRO Gigabit Ethernet NIC was connected to the system via the PCI Express subsystem. For kernel-device communication the stock Intel e1000e NIC device driver was used with default parameters except the size of the DMA ring buffer, which has been altered according to Section III.B.

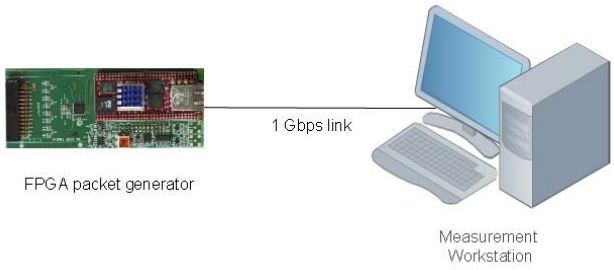

Figure 3. Custom FPGA-based packet generator is directly connected to the measurement workstation via a twisted pair Gigabit Ethernet link.

Each measurement scenario involved a series of measurements, each of them performed with fixed packet size and fixed inter-frame gap (IFG). The packet size is increased from 72 bytes up to 1200 bytes, while the IFG was constant 12 bytes within one measurement scenario. During the transmission, the following performance related system parameters were monitored:

- Aggregated CPU softIRQ load
- Per core softIRQ load
- Kernel level packet loss ratio
- Application level packet loss ratio

B. Statistical analyzis of the measurement data

The measurement data of each scenario is presented in an aggregated format on the following graphs. Figure 4 represents the softIRQ load of the dedicated core as function of the size of the generated packets for both preemptive and non-preemptive kernel variants, while Figure 5, 6 and 7 introduce the empirical cumulative distributions (ECDF) and the empirical complementary cumulative distributions (CCDF) for the measurement datasets.

Figure 4 represents the CPU core load that decreased to 1/10th of the original one for packet sizes from 500 to 1200 bytes using the proposed method with both preemptive and non-preemptive kernel variants. Furthermore, the non-preemptive kernel combined with the proposal provides significant improvement in CPU load with small packet sizes as well.

Lower CPU utilization derived from the lower softIRQ execution overhead, which improves packet loss ratio as presented on Figure 7. The improvements have been statistically analyzed.

Let's consider the lower end of the x-axis (core load) with the non-preemptive kernel version on Fig. 5. In case of the original non-modified variant (continuous blue line), CPU load is under 10 percent with approx. 0.12 of probability. In contrast, the application of the new method within the kernel (purple line) results on a significantly lower CPU utilization: CPU load is less than 10 percent with approx. 0.75 of probability.

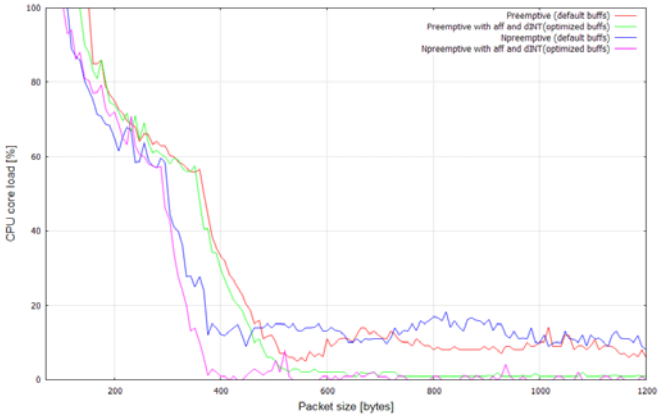

Figure 4. The CPU load of the softIRQ at various packet sizes from 72 to 1200 bytes. Measurements were done using both preemptive and non-preemptive kernels with and without the optimization method.

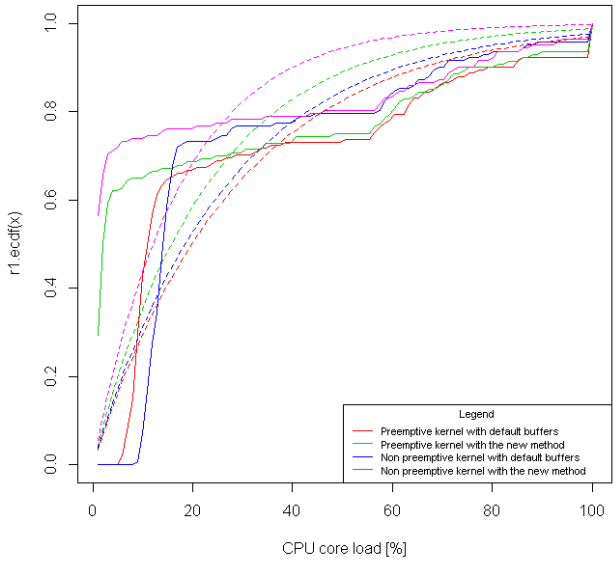

Figure 5. Empirical CDFs for the measurements presented on Fig. 4 and their datasets fitted to the exponential distribution.

The complementary CDF helps us to better observe the shape of the upper tail that describes the probability of large value [10]. Beyond the calculation of the empirical distributions, the datasets have been fitted to the exponential distribution in order to determine the

behavior of their upper tail (Fig. 6 and 7). Applying CCDF to CPU load datasets of these measurements, the large values dominates packet processing performance.

The empirical complementary cumulative distributions of the core load datasets have been calculated according (2).

$$\overline{F} = P(X > x) = 1 - F(x) \qquad (2)$$

All of the datasets have been fitted to the exponential distribution using Maximum Likelihood Estimation (MLE). In the next step, I determined the characteristics of the upper tail (Fig. 6). Since the probability of large values (near 100) is non-zero for all measurement scenarios and the tails decline slower than exponentially, all of the distributions has long-tail property, which is defined as (3). For a long-tail distribution, the shape of its upper tail looks like a straight line in log-log scale (Fig. 7).

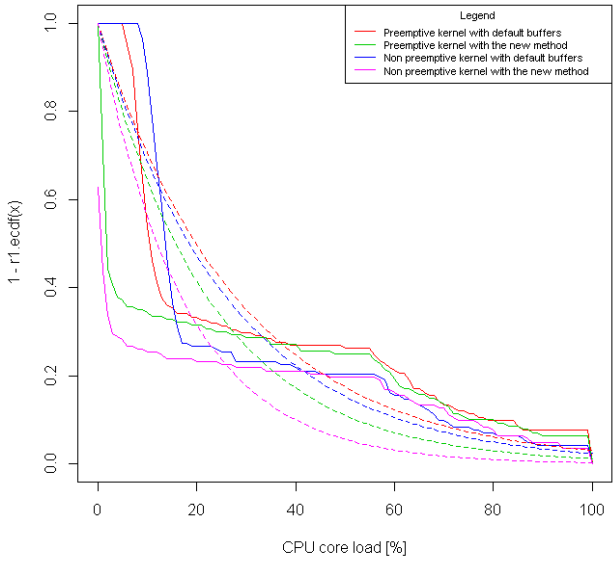

Figure 6. Empirical complementary CDFs for the measurements presented on Fig. 4 and their datasets fitted to the exponential distribution.

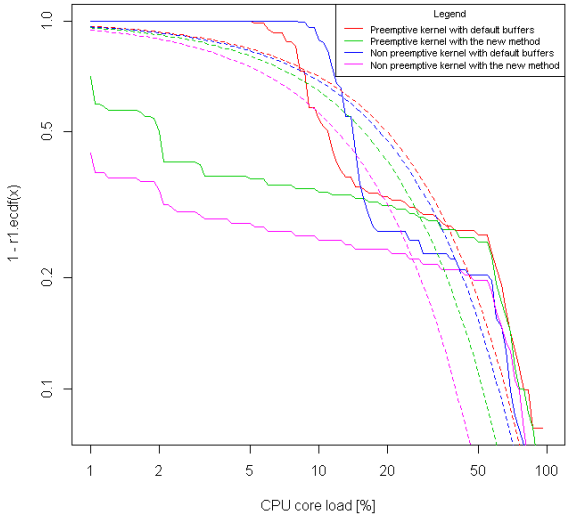

Figure 7. Empirical complementary CDFs with log-log axis

The long-tailed distribution has a polynomial tail behavior that is

$$P(X > x) \sim cx^{-\alpha} \text{ as } x \to \infty \qquad (3)$$

where X is a random variable, c is a location parameter, and α is a shape parameter [11].

Additionally, by disabling hardware interrupts on the core during the execution of *NET_RX_SOFTIRQ*, the mean of the execution overhead decreased from 28.65 % to 17.39 % of CPU load.

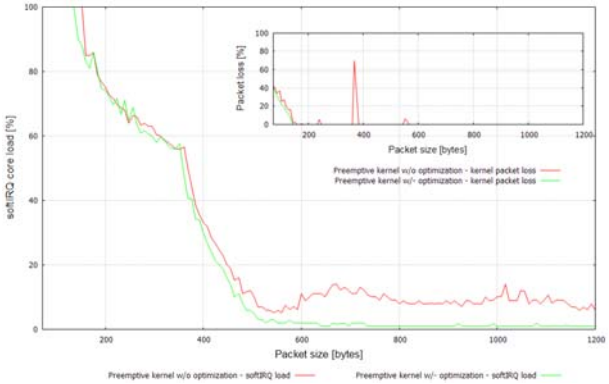

Figure 8. The CPU load of the softIRQ and the packet loss ratio in percentage at various packet sizes from 72 to 1200 bytes. Measurement was performed using preemptive kernel with and without the optimization method.

The incoming data is buffered at several layers within the system as presented in Section I. The logical data path is: NIC's ring buffer, sk_buffers, socket buffer and internal application buffer. Each layer implements its own buffer handling mechanism, which therefore could trigger buffering interference within the data processing path (Fig. 9). Application level buffer overflow was occurred from packet size of 380 bytes and the loss increases down to approx. 140 bytes. Then the trend was turning back due to the kernel level buffer overflow. Since the kernel buffers logically precede the application buffer in the data path, this low level loss decreases the amount of data the kernel has to pass up to the application and therefore the latter's processing load is decreasing, which results on lower loss ratio at the application level that turns down near to zero percent at very small packet sizes.

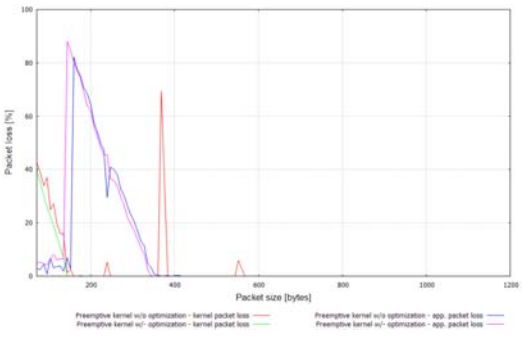

Figure 9. Packet loss ratio in percentage at various packet sizes from 72 to 1200 bytes at kernel and application levels. Measurement was performed with preemptive kernel with and without the optimization method.

Applying the proposed buffer adjustment, the kernel level loss ratio became zero for packet sizes from 200 to 1200 bytes. In contrast, without optimal buffers, packet loss can occur within the kernel at packet sizes up to approx. 500 bytes.

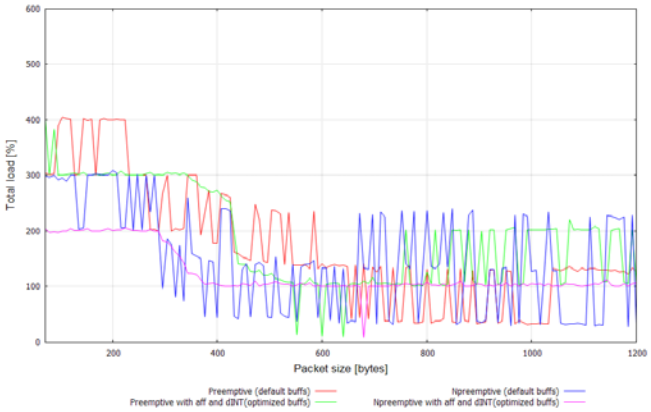

Figure 10. Aggregated CPU load in percentage at various packet sizes from 72 to 1200 bytes. Measurements were performed using both preemptive and non-preemptive kernels with and without the optimization method.

A further benefit of the method is that the aggregated CPU load is also decreased as presented on Figure 10. The proposed method has decreased the mean and variance of the total CPU load as function of the packet size.

V. Conclusion

An optimization method has been developed and implemented in order to improve packet processing efficiency of intensive ingress traffic in multi-core Linux systems with single input queue. A proposal of the method has been presented and its effectiveness was evaluated using its Linux based implementation in a real laboratory network environment and performing statistical analysis of the measurement data. According to the results of the analysis, the CPU load decreased to approx. $1/10^{th}$ of the original one for packet sizes from 500 to 1200 bytes using both preemptive and non-preemptive kernels. However, the non-preemptive kernel combined with the proposed method provided further improvement in CPU load with small packet sizes as well. With the proposed method, the execution overhead of the softIRQ decreased and CPU load is therefore less than 10 percent with approx. 0.75 of probability. Disabling interrupts on the core and thus not suspending the softIRQ execution produced very small amount of variance and therefore does not bring down packet jitter within a real-time data flow. Considering the large values, the empirical distributions of the CPU load, as showed in the paper, have long-tail property.

Acknowledgment

The work was supported by the TÁMOP 4.2.2.C-11/1/KONV-2012-0001 project. The project was implemented through the New Széchenyi Plan, co-financed by the European Social Fund.

References

[1] Christian Benvenuti, "Understanding Linux Network Internals," *O'Reilly*, 2006

[2] Peter Orosz; Tamas Skopko; Jozsef Imrek; , "Performance Evaluation of the Nanosecond Resolution Timestamping Feature of the Enhanced Libpcap," *6th International Conference on Systems and Networks Communications, ICSNC 2011*, October 23-28, 2011, Barcelona, Spain, ISBN 978-1-61208-166-3, Proceeding p. 220-225.

[3] Faulkner, M.; Brampton, A.; Pink, S.; , "Evaluating the Performance of Network Protocol Processing on Multi-core Systems," *Advanced Information Networking and Applications, 2009. AINA '09. International Conference on* , vol., no., pp.16-23, 26-29 May 2009

[4] Interrupt Moderation Using Intel® GbE Controllers, *Intel*

[5] Matthew Wilcox, "I'll Do It Later: Softirqs, Tasklets, Bottom Halves, Task Queues, Work Queues and Timers," *Hewlett-Packard Company*

[6] Yunchun Li; Xinxin Qiao; , "A Parallel Packet Processing Method on Multi-core Systems," *Distributed Computing and Applications to Business, Engineering and Science (DCABES), 2011 Tenth International Symposium on* , vol., no., pp.78-81, 14-17 Oct. 2011

[7] Chun-Ying Huang; Chi-Ming Chen; Shu-Ping Yu; Sheng-Yao Hsu; Chih-Hung Lin; , "Accelerate in-line packet processing using fast queue," *TENCON 2010 - 2010 IEEE Region 10 Conference* , vol., no., pp.1048-1052, 21-24 Nov. 2010

[8] Hyun-Wook Jin; Yeon-Ji Yun; Hye-Churn Jang; , "TCP/IP Performance Near I/O Bus Bandwidth on Multi-Core Systems: 10-Gigabit Ethernet vs. Multi-Port Gigabit Ethernet," *Parallel Processing - Workshops, 2008. ICPP-W '08. International Conference on* , vol., no., pp.87-94, 8-12 Sept. 2008

[9] Hye-Churn Jang; Hyun-Wook Jin; , "MiAMI: Multi-core Aware Processor Affinity for TCP/IP over Multiple Network Interfaces," *High Performance Interconnects, 2009. HOTI 2009. 17th IEEE Symposium on* , vol., no., pp.73-82, 25-27 Aug. 2009

[10] Mark Crovella; Balachander Krishnamurty;, "Internet Measurement, " Wiley, 2006

[11] Allen B. Downey, "Evidence for long-tailed distributions in the Internet, " *IMW '01 Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pp. 229 - 241