Towards 10Gb/s open-source routing

Olof Hagsand[1]
Robert Olsson[2]
Bengt Gördén[3]

1 Abstract

We present Linux performance results on selected PC hardware for IP packet forwarding in 10Gb/s speeds. In our experiments, we use Bifrost Linux on a multi-core NUMA PC architecture with multiple DMA channels, dual PCIe buses and 10GE network interface cards.

Our experiments were divided into TX and forwarding experiments. The purpose of the TX experiments was to explore hardware capabilities, while the purpose of the forwarding experiments was to give realistic bandwidth and packet rate numbers.

Our results show that 10Gb/s transmission rate is obtainable in wirespeed for down to 250 byte packets. Further, single-flow forwarding experiments using a single CPU core show a forwarding rate of around 900 Kp/s resulting in near wire-speed for larger packets. Traffic with more realistic packet distribution with packet filters and large FIBs enabled shows a forwarding bandwith around 4.5Gb/s.

We also show how multiple queues on the receive side were evenly distributed over multiple CPU cores. We identify a remaining bottleneck in the linux kernel before the full potential for multi-queue and multi-core forwarding can be utilized. When this bottleneck is removed, it would in principle be possible to forward realistic traffic in 10Gb/s wirespeed and beyond.

2 Introduction

The first IP routers were software-based and used off-the shelf computer architectures. As the requirement for throughput increased, applications specific circuits were developed (ASICs) along with high performance switching backplanes(e.g. cross-bars) and advanced memory systems (including TCAMs). This enables current routers to perform wire-speed routing up to Terabit speeds. The commercial high-end routers of today have little in common with a standard desktop.

On commercial high-end routers, the architecture has developed from an integrated routing and forwarding module (e.g. the BSD IP stack [8]) into a separated control-plane and data-plane where the former directs the real-time forwarding in the data-plane using management and signaling software. At the same time the complexity of the forwarding and routing protocols have increased resulting in more hardware, and more complex software modules, up to a point where hardware cost, power consumption and protocol complexity are important limiting factors of network deployment.

Simultaneously, development of routers on general-purpose computer platforms (such as PC's) has developed. In particular, general purpose hardware combined with open-source [11, 10, 9] have the advantages of offering a low-cost and flexible solution that is tractable for several niches of networking deployment. Such a platform is inexpensive since it uses off-the-shelf commodity hardware, and flexible in the sense of its openness of the source software and a potentially large development community.

[1]Royal Institute of Technology (KTH), Sweden `olofh@kth.se`
[2]Uppsala University, Uppsala, Sweden `robert.olsson@its.uu.se`
[3]Royal Institute of Technology (KTH), Sweden `gorden@kth.se`

One can also see the modularization development taking place in standardization forums including IETF ForCES [4] as a further trend that supports the development of open software routers, or possibly as a combination of open modules with efficient forwarding components [7].

However, many previous efforts have been hindered by performance requirements. While it has been possible to deploy open source routers as packet filterers on medium-bandwidth networks it has been difficult to connect them to high-bandwidth uplinks.

Somewhat simplified, performance limitations are dependent on per-packet (transaction) and per-byte (bandwidth) costs. On a PC architecture, per-packet costs have mainly to do with processing in software by CPUs and is therefore dependent on instruction count, I/O and memory latency, cache behaviour and clock frequency.

Per-byte costs are typically coupled to bandwidth limitations of buses and memory. In particular, a PC router has to pay the double bandwidth price of forwarding a packet from one network interface to main memory (via DMA) and then back to an outgoing network interface after being inspected by a CPU.

In particular, the 1Gbps PCI bus used to be a limiting factor during several years but with the advent of PCI Express, the performance has been increased by the use of parallell lanes and a new generation in bus technology with respect to DMA and bus arbitration. One important advantage with PCIe is that interrupts are transferred in-line instead of out-of-band using MSI, which enables a better handling since it allows for multiple queueable interrupts.

Memory cache behaviour is also important and is a crucial issue with the introduction of multi-core architectures. With the advances of efficient forwarding algorithms [2] and small memory footprints [5], IPv4 forwarding itself is seldom a limiting factor.

Other limitations have been advances in protocol development where open source routing have trailed behind commercial software vendors. One particular example of this is the lack of open source MPLS implementations, with associated VPN services using BGP and MPLS. On the other hand, the open source routing community tends to build its solution with simpler and cleaner network architectures often relying on IP-pure networks.

Our claim in this paper is that several current trends combined actually speaks for a renewed attempt of using general-purpose hardware, and we present an approach that we think has a potential for success in using on a larger scale in new application areas. In particular, with 10GE speed and low cost we believe that open source routers can be used by enterprises, small ISPs, and other scenarios where cost-efficiency and clean network design are important. But maybe the most important issue is the ability to participate in the development of new services, which can increase the knowledge and may give competitive advantages.

In this paper we discuss how to exploit the parallellism of multi-core CPU architectures with NUMA architecture and parallell PCIe buses combined with 10G Ethernet interface cards with multiple interrupt and DMA channels. We have chosen an advanced PC platform with large potential for parallellism that we believe will be commonplace very soon in desktop PCs.

3 Experimental platform and setup

For the experiments we use an advanced PC platform and run an experimental variant of the Bifrost Linux distribution.

The interface cards we selected were 10 Gigabit XF SR Dual Port Server Adapter PCI Express x8 lanes based on 82598 chipset from Intel with multiple interrupt and DMA channels. The cards have multiple RX and TX queues. In our experiments we use both two dual and single NICs. The kernel driver was ixgbe. A special ixgbe-1.3.16.1 driver was used for the multi-queue experiments.

The computer platform was an AMD Opteron 2222 with two double-core 3GHz CPUs combined with a Tyan Thunder n6650W(S2915) mother board with double PCIe buses, see Figure 1. The four CPUs are arranged in two double-cores, each having access to local memory, thus forming a simple NUMA architecture. Internal buses are HyperTransport (HT).

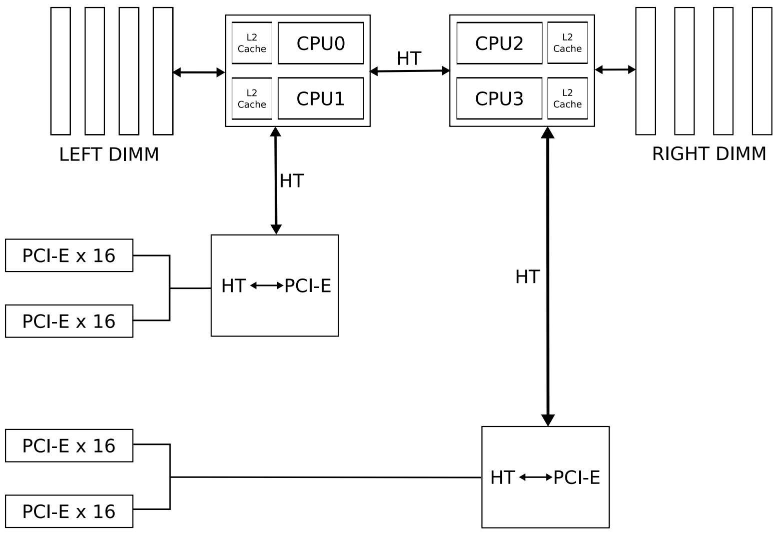

Figure 1: Simplified block structure of the Tyan 2915 board with two AMD 2222 CPUs and double PCIe buses.

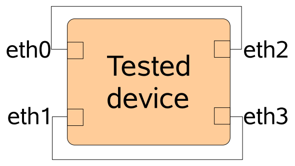

Figure 2: Experimental setups for transmission(TX). Fibers were loopbacked between local interfaces.

In the base configuration, we placed two dual 10GE cards on each of the PCIe buses. This means that we can use four CPUs, two main memories, two PCIe buses and four 10GE interfaces.

The software was Bifrost [9] version 5.9.1 which is a Linux release aimed at providing an open source routing and packet filtering platform. Bifrost includes routing daemons, packet filtering, packet generators, etc.

The Linux kernel was 2.6.24rc7 with the LC-trie forwarding engine, and traffic was generated using a modified version of pktgen [1], a Linux packet generator.

The experiments were conducted using two setups as shown in Figure 2 for transmission and Figure 3 for forwarding. The tested device is the experimental platform as described earlier and the test generator and sink device are similar systems. In all cases, pktgen was used to generate traffic, and interface counters were used to verify their reception. Received packets were only registered by the receiving port and not actually transferred over the bus to main memory. On this particular NIC, interface counters could still be read after configuring the interface as down.

3.1 Hardware selection process

Before selecting this particular hardware setup, we examined several other interface cards. Preliminary experiments performed on those cards were somewhat disappointing in terms of packet-per-second and

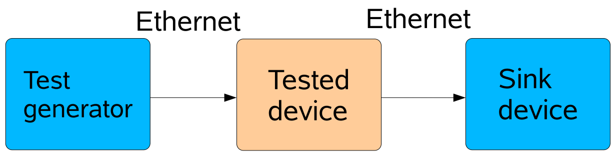

Figure 3: Experimental setups for forwarding. Traffic was generated, forwarded and terminated using three boxes.

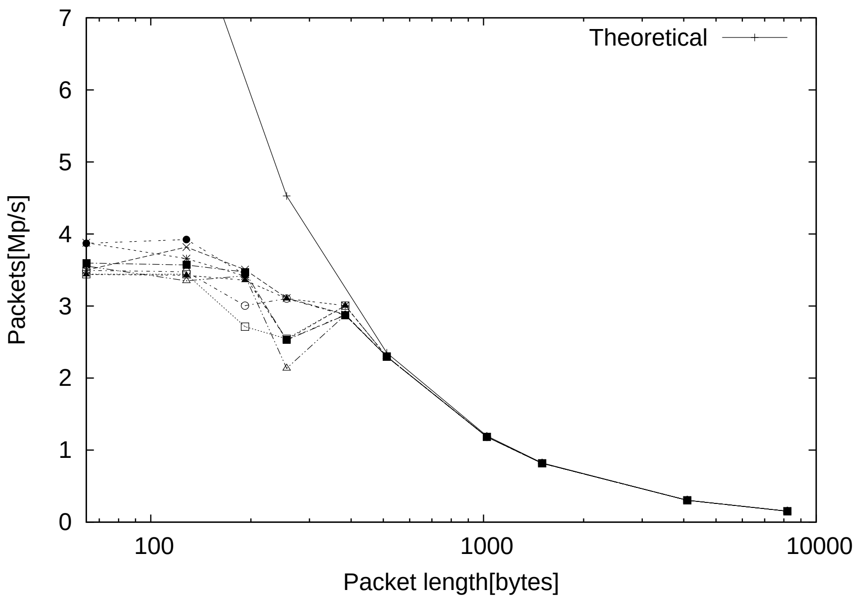

Figure 4: Transmit rate in million packets per second as a function of packet length in bytes. The figure shows the results of eight measurements and the theoretical limit. The x-axis is logarithmic.

bandwidth performance. The selected card matched our requirements and we believe that there are currently new competing cards that also match this performance. In any case, it is essential to pick the correct hardware components in order to get full 10GE performance.

4 Description of experiments

The experiments were divided into two main areas: Transmission (TX) and forwarding.

The purpose of the TX experiments was to explore hardware capabilities, including DMA, bus bandwidth and driver performance. This served as a hardware baseline in preparation for the forwarding experiments. By knowing TX performance, upper limits for IP forwarding are known in principle.

The forwarding tests are more complex and involves many factors that are difficult to study in isolation. First, a single flow was forwarded and the outgoing interface was varied using a single CPU. Thereafter, a realistic multiple-flow stream was forwarded also using a single CPU. In the last experiment, multi-queues on the interface cards were used to dispatch different flows to four different CPUs.

5 TX Result

5.1 Single-sender TX

In the first experiment, IP packets were sent from eth0 to eth2 (see Figure 2). Only a single CPU was used. Packets were not actively received, eth2 was used just to provide link.

In accompanying experiments, not shown here, we noted that sending from different CPUs to different cards on different PCIe buses had little impact on the results, therefore we conclude that the HT buses are not a limiting factor in these experiments, and we abstract away which CPU actually transmitted on each interface.

It is also important in this and all following experiments to assign TX interrupts so that the same CPU that sent the packet makes the buffer cleanup after TX. If not, the CPU gets cache misses when freeing buffers.

Eight sets of one million packets were sent for each packet size.

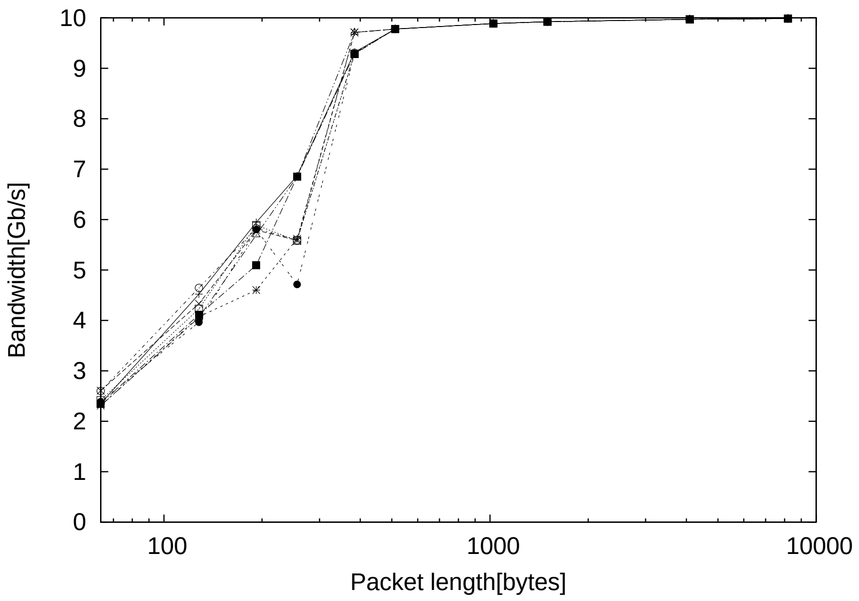

Figure 5: Bandwidth in Gb/s for a single sender. The figure shows the results from the same experiment as Figure 4 but bandwidth instead of pps. The x-axis is logarithmic.

	CPU 1	CPU 2	CPU 3	CPU 4	Total
Rate [pkt/s]	530K	530K	536K	536K	2.135M
Bandwidth [bit/s]	6.45G	6.45G	6.53G	6.53G	25.96G

Table 1: Four CPUs were in parallell transmitting 1500 byte packets using separate 10Gb/s ports. The table shows that the cumulative bandwidth was above 25 Gb/s.

Figure 4 shows packets per second as a function of packets size in bytes. The figure shows that highest pps for a single CPU is above 3.9 million.

Figure 5 shows result from the same experiments where bandwidth in bits per second is plotted against packet size. The figure shows clearly that wirespeed performance is maintained of up to 256 byte packets, and then drops.

Hardly surprising, one can see the limiting factor is per-packet cost for small packets which we believe is probably due to I/O latency.

5.2 Multiple-sender TX

In this experiment, the single sender case was extended to employ all four CPUs. Each CPU transmitted a single flow to a separate 10GE card over two separate PCIe buses. Fibers were loopbacked as shown in the upper part of Figure 2 and the receive side simply counted packets without further processing.

Table 1 shows the result of transmitting 1500-byte packets using all four CPUs in parallell.

From the table it can be seen that the total bandwidth produced is 25.9 Gbps, and there is also an even distribution between the CPUs and interfaces. The limitation is probably in the PCIe buses, since two CPUs try to send 20Gbps in total over a single PCIe bus, see Figure 1.

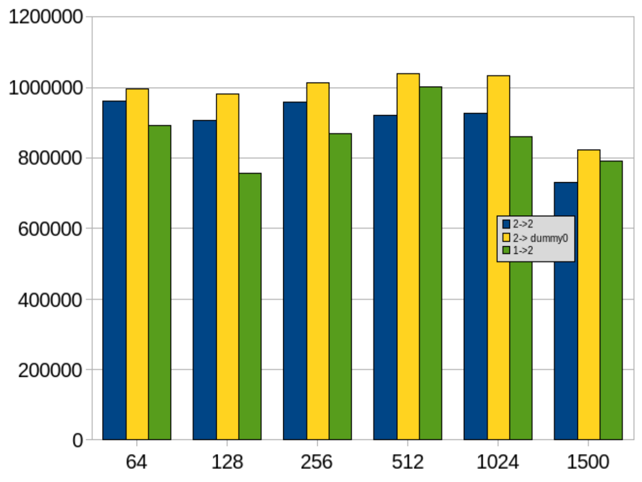

Figure 6: Packets per second as a function of packet length for forwarding of a single flow and single CPU. The left bars show the result when ingress and egress was placed on the same card, the middle when a dummy interface was used, and the right bars show the result when ingress and egress were placed on different cards.

6 Forwarding Result

6.1 Single CPU, single flow

In the following experiments, the setup in Figure 3 was used. A single packet flow was was sent from B to C, with router A receiving packets on one interface and forwarding them on another.

Packets were forwarded by a single CPU, with a single DMA from RX and a single DMA to TX was performed. Since all packets belonged to a single flow, all lookups were made in the destination cache.

The experiment first used three different output interfaces:

1. Same card. The input and output interfaces were on the same card but different port on the dual adaptor. This means that both RX and TX was made over the same PCIe bus.

2. Dummy interface. RX and all software processing was performed, but not the final TX.

3. Different cards. RX and TX on different PCIe buses.

Figure 6 shows the packet-per-second graph comparing these three different strategies.

As can be seen from the figure, there is little variation between the experiments, and the primary limiting factor is packets per second at around 900kpps. This results in near wire-speed for larger packets but degrading bandwidth performance at lower packet sizes.

Profiling was done for each experiment in order to get a detailed understanding of the CPU and code execution. Figure 7 shows the profiling in the different cards case. The figure shows that the CPU spends a large part of its time in buffer handling. Input handling seems also, as expected, to yield more work than forwarding and output.

6.2 Single CPU, multiple flows

In the next set of experiments, pktgen was configured to produce a mix of flows with varying destination address and packet sizes. 8K simultaneous flows were generated, where each flow consisted of 30 packets. Table 2 shows the packet-size distribution which has a relatively high percentage of 64 byte packets. The scheduling of the flows were made using four concurrent CPUs each using round-robin internally.

This resulted in a packet stream of around 31000 new flows per second aimed at representing a realistic packet flow in a relatively loaded network environment.

```
samples  %        symbol name
396100   14.8714  kfree
390230   14.6510  dev_kfree_skb_irq
300715   11.2902  skb_release_data
156310    5.8686  eth_type_trans
142188    5.3384  ip_rcv
106848    4.0116  __alloc_skb
75677     2.8413  raise_softirq_irqoff
69924     2.6253  nf_hook_slow
69547     2.6111  kmem_cache_free
68244     2.5622  netif_receive_skb
59197     2.2225  __netdev_alloc_skb
59179     2.2218  cache_flusharray
53777     2.0190  ip_route_input
49528     1.8595  ip_rcv_finish
48392     1.8169  __qdisc_run
39125     1.4689  ip_forward
36634     1.3754  dev_queue_xmit
33888     1.2723  cache_alloc_refill
33465     1.2564  ip_finish_output
```

Figure 7: Forwarding profiling: Single-CPU.

Packet length[bytes]	Distribution
64	45%
576	25%
1500	30%

Table 2: Packet-size distribution in the forwarding experiments.

Experiment	Mean Bandwidth
Baseline	4.6 Gb/s
Extended	4.1 Gb/s

Table 3: Forwarding bandwidth for multiple flows. The extended experiment includes a large FIB and IP tables.

	1 CPU	4 CPUs
Rate [pkt/s]	582K	521K

Table 4: Forwarding of a multi-flow stream using a single CPU and four CPUs in parallell.

The linux forwarding cache performs well as long as a limited set of new flows arrive per second. In this experiment the 31K new flows per second corresponds to a destination cache miss rate of around 5%.

The FIB was extended to 214K routes and the netfilter modules were loaded - again without actually filtering any packets.

The result of these experiments are shown in Table 3. The mean value of several experiments are shown where the variation is relatively large, on the order of 400Mb/s. In the first configuration (baseline) only forwarding was made, while in the second (extended), IP filters were enabled and the FIB extended. It can be seen that enabling IP filtering and extending the FIB reduces the performance somewhat (4.6 Gb/s and 4.1 Gb/s respectively). The reasons of this is most probably the increased number of instructions per packet that needs to be made by the CPU.

6.3 Multiple queues, multiple CPUs

In the previous experiment, only one CPU was used to forward packets from exactly one interface to another. One way to extend this is to add more CPU cores to handle the forwarding path and thus increase the performance.

In this experiment the novel multi-queue functionality of the Intel NICs was tested. This means that for a single input interface, four interrupt channels were allocated, one for each CPU, corresponding to four different queues (ring-buffers) on the interface card. Dispatching between packets use a hashing algorithms so that flows are evenly distributed between CPUs - at least for multi-flow traffic.

We generated an input stream consisting of 8096 simulataneous 64-byte flows, where each flow was 30 packets long. The approximate sending rate was 820K packets per second. First, we let a single CPU forward this stream, and then let all four CPUs forward this stream in parallell. The number of packets that were forwarded were then recorded.

It can be seen from the Table 4 that the total forwarding capacity of one CPU (581.5Kpps) surpasses the forwarding of the four CPUs in parallell (520.6Kpps). That is, using four parallell CPUs actually leads to lower performance than using a single CPU. The second observation visile in Table 5 is that the load is evenly distributed between the CPU cores.

The profiling of the forwarding code in Figure 8 shows that a large part of the CPU time is spent in the "dev_queue_xmit" and "__qdisc_run" code, much more than in previous experiments (see Figure 7). It turns out that this piece of code is a serialization point where common data-structures are manipulated

	CPU 1	CPU 2	CPU 3	CPU 4
Rate [pkt/s]	129K	131K	131 K	130K

Table 5: Individual forwarding performance of the four CPUs forwarding in parallell.

```
samples  %        symbol name
1087576  22.0815  dev_queue_xmit
651777   13.2333  __qdisc_run
234205    4.7552  eth_type_trans
204177    4.1455  dev_kfree_skb_irq
174442    3.5418  kfree
158693    3.2220  netif_receive_skb
149875    3.0430  pfifo_fast_enqueue
116842    2.3723  ip_finish_output
114529    2.3253  __netdev_alloc_skb
110495    2.2434  cache_alloc_refill
```

Figure 8: Forwarding profiling: Multi-queue, multi-CPU

that lead to cache misses that effects the performance.

We have taken a multiple flow stream and distributed the traffic evenly between the multiple CPUs. With the hardware classifiers on the network interface cards and the MSI interrupts and the driver code network traffic is evenly distributed among the CPU cores. The network stack runs in parallell between the CPUs and we can note that the TX interrupts are assigned.

However, we have identified a bottleneck in the linux forwarding code that need to be addressed before we can continue and further increase the forwarding capacity by adding more CPUs. The TX and the qdisc code needs to be adapted so that its performance can scale up in the case of multiple CPUs.

7 Discussion

We have shown how we to make efficient forwarding up to 10Gb/s speeds using the Linux kernel and PC hardware. When obtaining such results, it is important to tune software, configuring interrupt affinity and allocating DMA channels adequately. It is also important to carefully select CPU cores, interface NICs, memory and buses to obtain a system with suitable performance. A large issue has to do with avoiding cache misses by constructing highly local code that is independent of shared data structures.

A key issue is how to distribute input load over several memories and CPU cores. An important building stone is the multi-queue and hardware classification support provided by many modern interface cards, for example as specified by the Receiver Side Scaling [6]. With multi-queue, input is distributed via separate DMA channels to different memories and CPU cores using hashing of the packets headers. This provides basic support for virtualization but is also essential for forwarding performance since it provides a mechanism to parallelize the packet processing.

With hardware classification, it is possible to go a step further in functionality and performance by off-loading parts of the forwarding decisions to hardware. The Intel cards used in these experiments supports hashing while other cards, including NICs from SUN microsystems, implement TCAMs which enables a finer granularity classifications to be made.

Other fields where hardware classification is useful include quality-of-service, fast filtering, packet capture and even stateful networking and flow lookup.

We even call for more advanced requirements than what the RSS defines to more challenging and valuable classifier functions. This to improve the forwarding in open source routing further and to challenge hardware vendors. Minimal function should be required and standardized to start and support software development. In this way we believe that open source routers can truly challenge the high-end router vendors with open and low-cost solutions.

8 Conclusions

There is no doubt that open source routing has entered the 10Gb/s arena. Network interface cards, CPUs and buses can do 10Gb/s with relatively small packet sizes. Forwarding is possible at 10G/s speed with large packets, and around 5Gb/s in a mixed flow environment. Just using a single CPU core.

To utilize multi-queues and load-balancing of forwarding among several CPU cores, we identified an issue with the last part of the TX qdisc code that is not fully ready for being used in such a parallellized environment. When this remaining bottleneck is removed, we can fully utilize the full potential of multi-core, multi-queue forwarding. In fact, while finishing this paper, kernel patches for an improved multiqueue TX implementation with less lock contention has been implemented.

Acknowledgements

This paper is a result of a study sponsored by IIS. Interface boards were provided by Intel. Thanks also to several helpful reviewers.

References

[1] R.Olsson. *Pktgen the linux packet generator*. In Proceedings of the Linux Symposium, Ottawa, Canada, volume 2, pages 11 - 24, 2005.

[2] S. Nilsson, and G. Karlsson, *Fast address look-up for Internet routers*, In Proc. IFIP 4th International Conference on Broadband Communications, pp. 11-22, 1998.

[3] R. Olsson, and Stefan Nilsson, *TRASH A dynamic LC-trie and hash data structure*, Workshop on High Performance Switching and Routing, (HPSR '07) 2007.

[4] L. Yang, R. Dantu, T. Anderson, and R. Gopal, *Forwarding and Control Element Separation (ForCES) framework*, IETF RFC 3746, April, 2004

[5] M. Degermark et al., "Small Forwarding Tables for Fast Routing Lookups". in Proc. *ACM SIGCOMM Conference'97*, pages 3-14, Oct. 1997

[6] Microsoft Corporation, "Scalable Networking: Eliminating the Receive Processing Bottleneck-Introducing RSS", WinHEC 2004 Version - April 14, 2004

[7] M. Hidell, P. Sjödin, and O. Hagsand, "Control and Forwarding Plane Interaction in Distributed Routers", in Proc. of the 4th IFIP-TC6 Networking Conference: Networking 2005, Waterloo, Canada, May 2005.

[8] S. J. Leffler, M. K. McKusick, M. J. Karels, and J. S. Quarterman, "The Design and Implementation of the 4.3 BSD Operating System". Addison-Wesley, May 1989.

[9] R. Olsson, H. Wassen, E. Pedersen, "Open Source Routing in High-Speed Production Use", Linux Kongress, October 2008.

[10] Kunihiro Ishiguro, et al, "Quagga, A routing software package for TCP/IP networks", July 2006

[11] M. Handley, E. Kohler, A. Ghosh, O. Hodson, P. Radoslavov, "Designing Extensible IP Router Software", in Proc of the 2nd USENIX Symposium on Networked Systems Design and Implementation (NSDI) 2005.