10Gbps Open Source Routing

Projektdeltagare: Bengt Gördén KTH, Olof Hagsand KTH, Robert Olsson UU

Projekttid: 20071001 - 20080331

Resultatet av projektet använder PC-hårdvara och öppen källkod för IP paketforwardering i 10Gbps hastighet. Vi använder PC med multicore-arkitektur och NUMA-noder med multipla DMA-kanaler, dubbla PCIe bussar och state-of-the-art 10GE nätverkskort. I dokumentet fokuserar vi på transmission av paket och studier av genomströmning och multicore-prestanda. Våra preliminära resultat visar att man kan skicka 250 byte stora paket eller större i full hastighet (wirespeed 10Gbps). Vi visar också hur multipla köer på inkommande interface distribueras jämt över multipla CPU-kärnor. Vi identifierar även en flaskhals i linuxkärnan som, innan den åtgärdats, inte möjliggör full multikö och multicore forwarding.

Vi vill i och med detta dokument också uppmärksamma IIS och Intel för deras välvilliga inställning till detta projekt och sponsrat oss med medel och hårdvara.

.se

intel

KTH
VETENSKAP
OCH KONST

UPPSALA
UNIVERSITET

Towards 10Gbps open-source routing

Bengt Gördén*, Olof Hagsand* and Robert Olsson†
* Royal Institute of Technology (KTH), Sweden
Email: {gorden, olofh}@kth.se
† Uppsala University, Uppsala, Sweden
Email: robert.olsson@its.uu.se

***Abstract*—We present performance results using selected PC hardware and open source software for IP packet forwarding in 10Gbps speeds. In our experiments, we use a multi-core NUMA PC architecture with multiple DMA channels, dual PCIe buses and state-of-the art 10GE network interface cards. In this paper we focus on packet transmission and study throughput and multi-core performance.**

Our primary results are that 10Gbps tranmission rate is obtainable in wirespeed for down to 250 byte packets. We also show how multiple queues on the receive side were evenly distributed over multiple CPU cores. We also identify a remaining bottleneck in the linux kernel before the full potential for multi-queue and mult-core forwarding can be utilized.

I. 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 [9]–[11] 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.

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 tranferreis 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

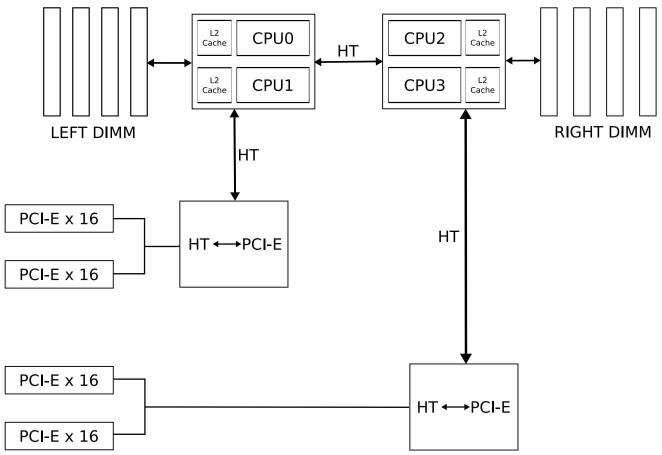

Fig. 1. Simplified block structure of the Tyan 2915 board with two AMD 2222 CPUs and double PCIe buses.

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.

II. 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 computer platform was an AMD Barcelona 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).

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 pktgen [1], a Linux packet generator.

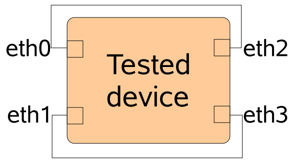

Fig. 2. Experimental setups for transmission(TX). Fibers were loopbacked between local interfaces.

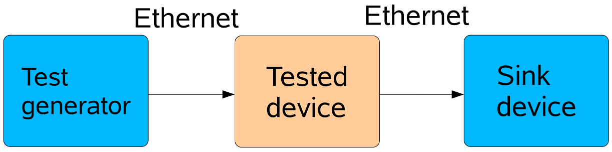

Fig. 3. Experimental setups for forwarding. Traffic was generated, forwarded and terminated using three boxes.

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.

A. 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 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.

III. 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.

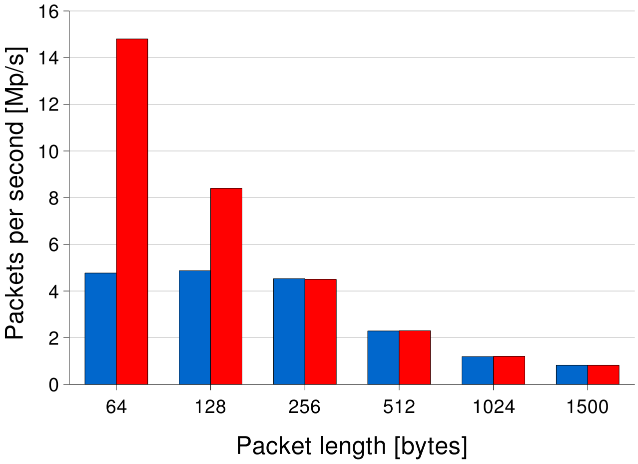

Fig. 4. TX single sender: Packets per second.

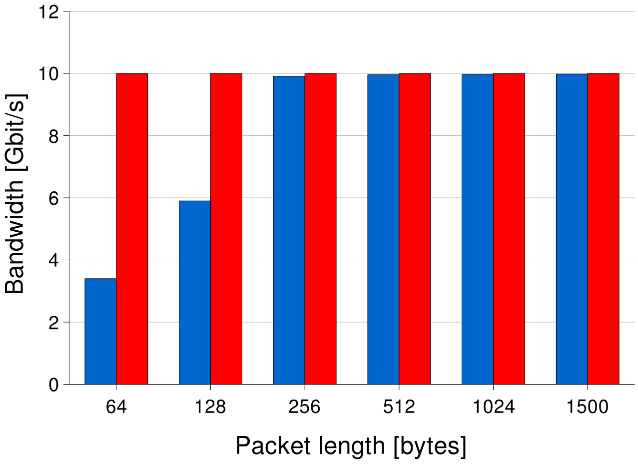

Fig. 5. TX single sender: Bandwidth

IV. TX Result

A. 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 a cache misses when freeing buffers.

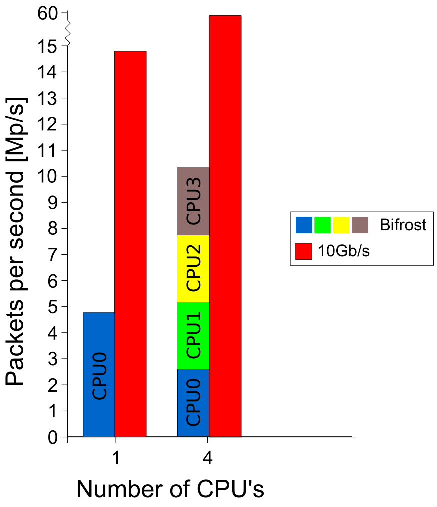

Fig. 6. TX multiple senders: Packets per second for 64 byte packets shown for one and four CPUs. Note that the y-axis is broken between 15 and 60 Mp/s.

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

Figure 4 shows millions of packets per second as a function of packets size in bytes. The ideal graph for perfect 10Gbps forwarding is also shown. The figure shows that maximum pps for a single CPU is above 4 million.

Figure 5 shows result from the same experiment where bandwidth in Gbps 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.

B. 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.

Figure 6 shows the result of transmitting 64-byte packets. The x-axis represents the different senders while the y-axis are packets per second. In the figure, the left columns show the experimental values while the right columns show the theoretical limit. Note that the y-axis is broken to fit the theoretical packets per second which reaches approximately 60Mp/s for 4x10Gbps.

Note also that the left column in the case of four CPUs has been partitioned showing how much each CPU contributes.

Two things can be noted from this experiment: The total transmission rate is around 10Mpps and the distribution between the CPUs is even. However, the packet rate is lower for a single sender (2.5 vs 4.7 Mpps).

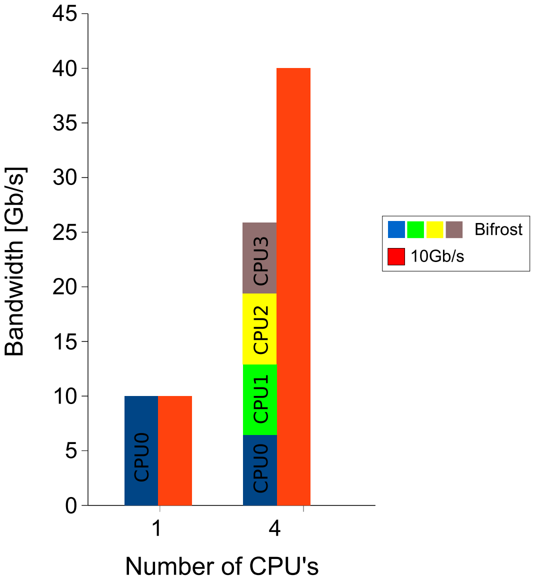

Fig. 7. TX multiple senders: Bandwidth for 1500 byte packets shown for one and four CPUs.

Figure 7, shows the resulting bandwidth when each of the four streams transmitted 1500 byte packets. The y-axis represents Gbps, and the left columns show experimental results.

In the bandwidth plot, it can be seen that the total bandwidth produced is 25.8 Gbps, and there is also a fair 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.

V. FORWARDING RESULT

A. 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.

Figure 9

The experiment first used three different output interfaces:

- 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.
- Different cards. RX and TX on different PCIe buses.
- Dummy interface. RX and all software processing was performed, but not the final TX.

Figure 8 shows the packet-per-second graph. In the graphs comparing these three different strategies.

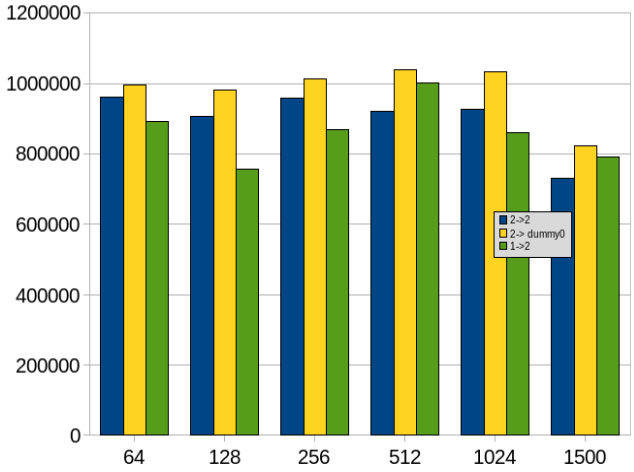

Fig. 8. Forwarding in PPS: Single CPU, single flow

```
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
```

Fig. 9. Forwarding profiling: single-CPU

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.

Figure 11 shows a similar experiment where the netfilter module has been loaded, but without any packet filters. One can see a minor performance degradation due to the increased per-packet cycle cost.

Profiling was done for each experiment in order to get a detailed understanding of the CPU and code execution. Figure 9 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.

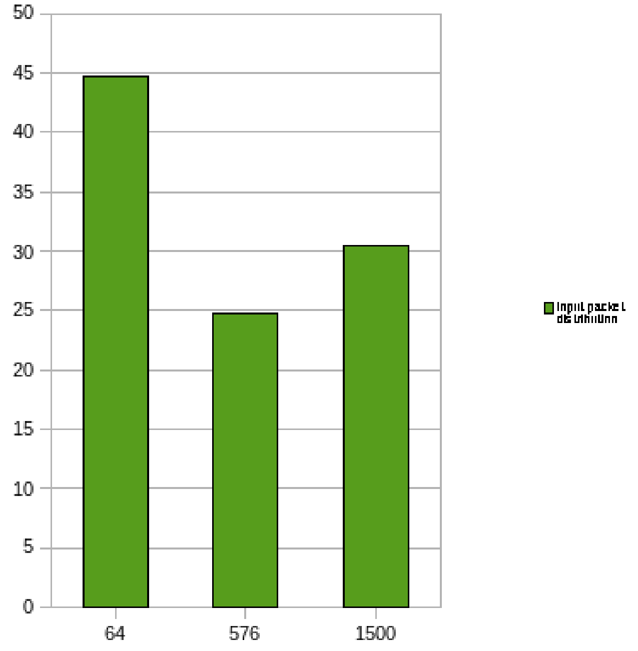

Fig. 10. Packet distribution

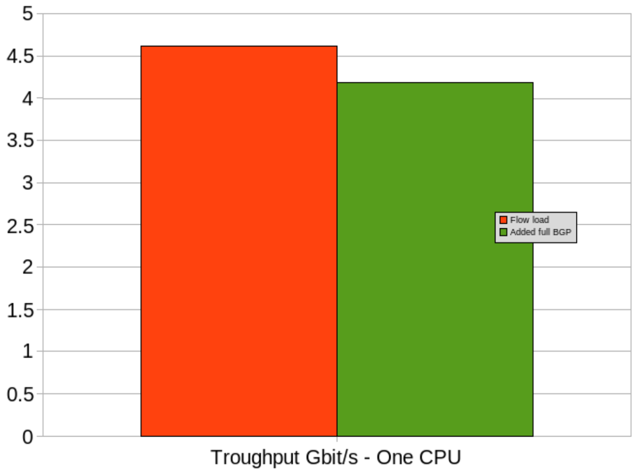

Fig. 11. Forwarding multiple flow PPS

B. *Single CPU, multiple flows*

In the next set of experiments, pktgen was modified 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. Figure 10 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 normal network environment.

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%.

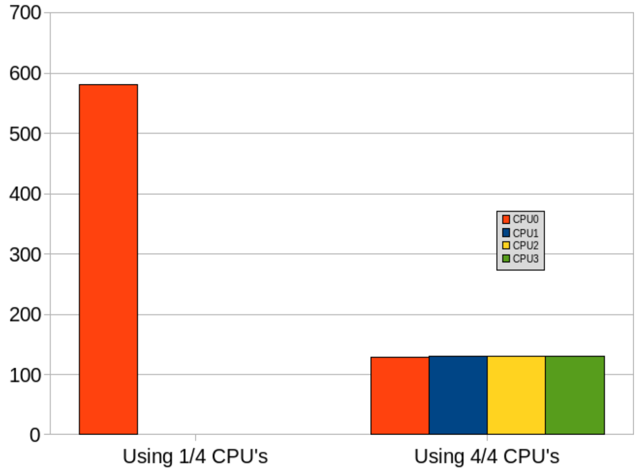

Fig. 12. Multi-queue forwarding

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 Figure 11. It can be seen from the figure that enabling IP filtering and extending the FIB reduces the performance somewhat. The reasons of this is most probably the increased number of instructions per packet that needds to be made by the CPU.

C. *Multiple queues, multiple CPUs*

Only 64 bytes pakets

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 CPUs to handle the forwarding path and thus increase the performance. It should be noted however that forwarding packets in this simplex way is not a realistic scenario. More realistic would be to forward between several interfaces and increasing the performance by letting different CPUs handle different incoming interfaces, for example.

In this experiment the novel multi-queue functionality of the Intel NICs was tested. This means that for a single input interface, four DMA 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.

In our case, the four different CPUs forward packets in parallell and the transmit the packets to the same output interface.

Figure 12 shows the behaviour of multi-queue forwarding. The case with four parallell CPUs actually leads to lower performance than the single CPU case.

The profiling of the forwarding code in Figure 13 shows that a large part of the CPU time is spent in the ”dev_queue_xmit”

```
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
```

Fig. 13. Forwarding profiling: Multi-queue, multi-CPU

and ”__qdisc_run” code, much more than in previous experiments (see Figure 9). It turns out that this piece of code is a serialization point where common data-structures are manipulated that lead to cache misses that effects the performance.

We have taken a 10GE stream and we have 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.

VI. 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.

VII. CONCLUSIONS

There is little 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.

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.

Acknowledgements

This paper is a result of a study sponsored by IIS. Interface boards were provided by Intel.

REFERENCES

[1] R.Olsson. *Pktgen the linux packet generator*. In Proceedings of the Linux Symposium, Ottawa, Canada, volume 2, pages 1124, 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”, Uppsala University, work-in-progress, 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.