# Open-source routing at 10Gb/s

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

## I. Abstract

We present throughput measurements using the Bifrost Linux open source router on selected PC hardware. The hardware consists of eight CPU cores, NUMA architecture, double PCIe buses and Intel and SUN 10Gb/s interface cards. These cards are equipped with hardware classifiers that dispatch packets to multiple DMA queues which enables parallel processing of packet forwarding and load-balancing between the multiple CPUs.

In our experiments, we send a multiflow, simplex packet stream through an open-source router. We measure the throughput and vary packet size, active CPUs, and router configuration. In the experiments, we use an IP flow and packet-length distribution that we claim is realistic for many network scenarios. Using these realistic traffic streams, we show how speeds close to 10Gb/s is achievable for normal Internet traffic.

In particular, we show how the use of multiple CPUs increases the throughput up to a breakpoint which in our setting is at four CPUs. Further, we show that adding filters and full BGP tables have minor effects in the performance.

With these results, we claim that open source routers using new PC architectures are a viable option for use in 10Gb/s networks for many network scenarios.

## II. Introduction

Although the first IP routers were software-based, the forwarding in modern commercial routers are primarily hardware-based, containing applications specific circuits (ASICs), 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.

However, 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 [8], [7], [6] 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.

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.

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 [3] and small memory footprints [4], IPv4 forwarding itself is seldom a limiting factor.

We believe that several current trends combined actually speaks for a renewed attempt of using general-purpose hardware, and we have shown 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 previous work [1], we showed how multi-core CPU architectures with NUMA architecture and parallell PCIe buses combined with 10G Ethernet interface cards with classifiers and multiple interrupt and DMA channels could be used to speed up the packet forwarding of a Linux-based router. We identified several bottlenecks that had to do with locking mechanisms in different parts of the code that caused cache-misses. This was particualry evident in the so-called

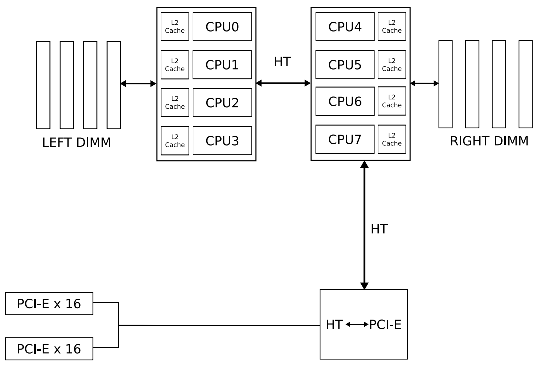

Figure 1. Simplified block structure of the Tyan 2927 board with two AMD 2382 CPUs and a single PCIe bus.

"qdisc" part of the kernel code, where output queueing was managed. During the last year, many of these bottlenecks have been removed in the Linux kernel, thus paving the way for full usage of the parallellism provided by the new hardware.

## III. Experimental platform and setup

The experiments us a prerelease of Bifrost [6] 6.0 based on Linux kernel 2.6.29-rc2 64-bit with NUMA support. Bifrost 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 had the LC-trie forwarding engine [3], and traffic was generated using a modified version of pktgen [2], a Linux packet generator.

The motherboard is a TYAN Thunder 2927 with two Quad-Core AMD Opteron(tm) Processor 2382 with 2.6 GHz, 8 CPUs in total, see Figure 1. The eight CPUs are arranged in two quad-cores, each having access to local memory, thus forming a simple NUMA architecture. Internal buses are HyperTransport (HT).

We used two network interface cards:

- *Intel ixgbe*. A 10 Gigabit XF SR Dual Port Server Adapter PCI Express x8 lanes based on the 82598 chipset 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 is ixgbe. The card unfortunately has fixed opto-modules.
- *SUN niu*. Sun Neptune is a dual 10 Gbit/s, PCIe x8-based network card. The NIC has 8 receive and 12 transmit DMA channels per port. The board supports different physical connections via XFP modules and can do programmable receive packet classification in hardware using TCAMs.

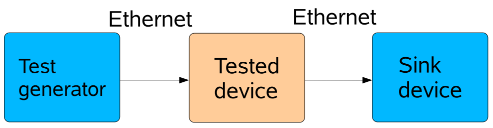

Figure 2. Experimental setups for forwarding. Traffic was generated, forwarded and terminated using three computers.

The Linux driver is niu, named after the board (Network Interface Unit). The niu driver has addtional patches to improve multi- queue handling

Both network interface cards have a hardware classifier that computes a hash-value of the packet header. The hash-value is then used to select receive queue, and thus selects which CPU receives the packet. Load-balancing between CPUs therefore require the traffic header to consist of several flows, enough to make the hash-function evenly distribute the traffic over the different CPUs.

The SUN niu card also has a TCAM classifier for more finer-grained classification, but it was not used in these experiments.

## IV. Description of experiments

Three experiments were made where in each experiment throughput was measured as packets per second (pps) and bits per second (bps). The first measure is an indication of per-packet performance and usually reveals latencies and bottlenecks in CPU and software, while the second is often associated with limits in bus and memory bandwidth.

1) *Throughput vs packet length*: Packet lengths varied between 64 and 1500 bytes.
2) *Throughput vs number of CPU cores*: The number of CPU cores varied between 1 and 8.
3) *Throughput vs functionality*: Router functionality varied in terms of filtering and routing table size.

The experiments were used the setup shown in Figure 2. Traffic was sent from the generator via the router to the sink device. The tested device was the experimental platform described in Section III, the test generator and sink device being similar systems.

In all cases, pktgen was used to generate traffic, and interface counters were used to verify their reception. On the sink device, 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.

A reference measurement was made with an IXIA traffic analysator which showed that our figures are within 5% of the IXIA measurements. A weakness of our setup is that the traffic generator can not generated mixed flow traffic in a rate higher than 2.5 Mpps. This means that the generator is saturated at this point, and that also the router forwarding is limited at this level.

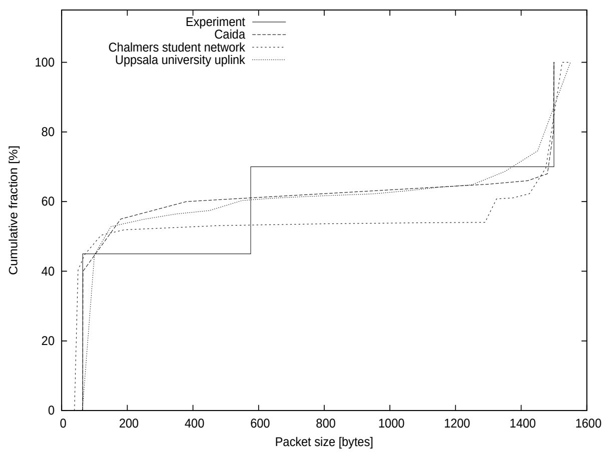

Figure 3. Packet length distribution

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

Table I
PACKET LENGTH DISTRIBUTION USED.

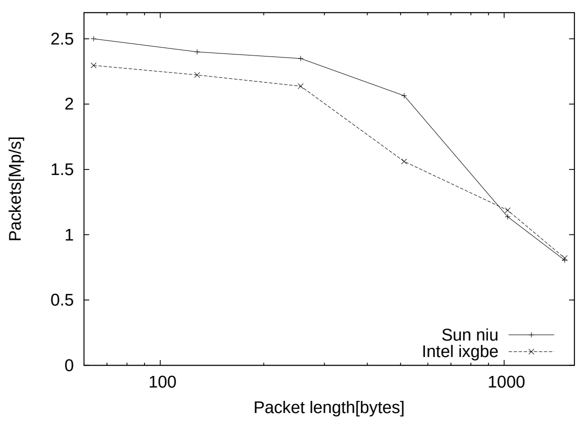

Figure 4. Throughput in million packets per second as a function of packet length in bytes. Note that the x-axis is logarithmic.

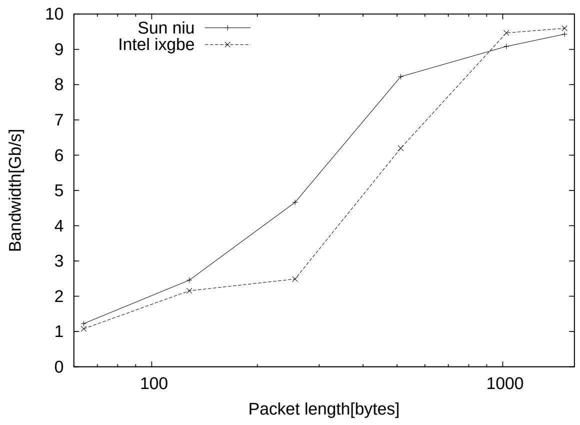

Figure 5. Throughput bandwidth in Gb/s as a function of packet length. The x-axis is logarithmic.

However, separate tests with an IXIA, not presented here, show that the router can forward packets up to 3.4Mpps, which means that the router under-performs in these experiements, at least for small packet sizes.

### A. *Traffic distribution*

All traffic used an even random distribution of destination IPv4 addresses in the $11/8$ range. There were 8000 such flows at one given time, and every flow had a duration of 30 packets. The flows were scheduled using a round-robin scheduler. This resulted in ca 31K new flows per second.

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

There were two packet length distributions:

1) *Fixed*. In experiment 1, the packet length was varied between 64 and 1500 bytes.
2) *Mixed*. In experiments 2 and 3, a packet length distribution shown in Table I was used. Figure 3 compares this distribution with three realistic distributions: the WIDE MAWI transpacific link in march 2008 [9], one taken from the Chalmers student network in December 2008 [10] and one from the Uppsala university uplink router [11].

### B. *Router functionality*

There were four router functionality configurations:

1) *Basic*. A small routing routing table and no modules loaded.
2) The *Netfilter* module was loaded, but without performing actual filtering.
3) Netfilter and *connection tracking* module loaded
4) Netfilter was loaded and *full BGP* table (280 thousand routes).

## V. RESULTS

### A. *Experiment 1: Throughput vs packet length*

In the first experiment, the traffic flow mix was sent from the source, via the router and received by the sink. The router had basic functionality and used all eight CPUs. The packet length was varied between 64 and 1500 bytes.

The results are shown in Figures 4 and 5. It can be seen that a maximum packet rate of 2.5 Mb/s is achieved at low packet rates. It can also be seen that the router shows close to wire-speed performance at large packet lengths.

The figures also show that in these experiments, the Sun niu card performs better than the Intel ixgbe.

Unfortunately, due to the limitation of the source, rates higher than 2.5Mpps can not be generated, the limitations shown in the figures for small packets are therefore probably a limitation of the sender, not the router itself.

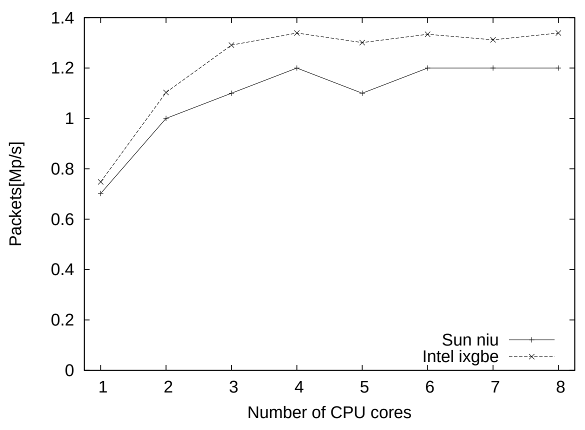

Figure 6. Throughput in million packets per second as a function of active CPUs.

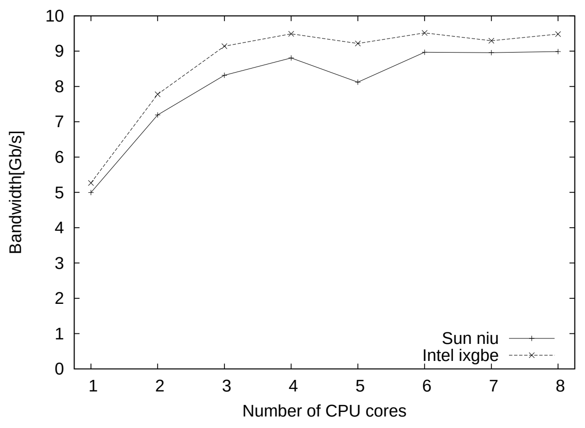

Figure 7. Throughput bandwidth in Gb/s as a function of number of active CPUs

## B. Experiment 2: Throughput vs number of CPU cores

In the second experiment, the traffic flow mix was sent using the packet length distribution as shown in Table I. The funtionality of the router was basic, but the number of CPU cores varied. The results are shown in Figures 6 and 7.

It can be seen that the performance starts at approximate 5Gb/s and then increases to a point at around 3-4 CPUs where the performance levels off at around 9 Gb/s. One could see this as an indication of detoriating utilization of the CPUs. However, this probably depends on the traffic mix and the results may change with changing traffic distribution.

There is also a question on what hinders the last gap to 10Gb/s wirespeed. This is an important question that requires further study. Preliminary tests with an IXIA packet generator indicates that here again the traffic generator may underperform. It may also be that packets are lost in transit, also at lower utilization, a fact which would be a negative result.

It should be noted that at this traffic distribution, the Intel ixgbe performs better than the SUN niu.

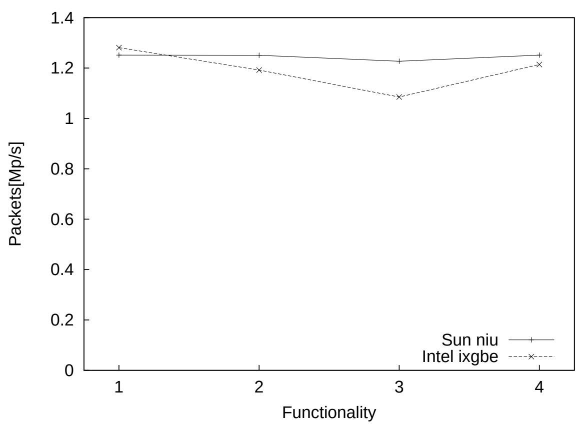

Figure 8. Throughput in million packets per second with different routing configurations.

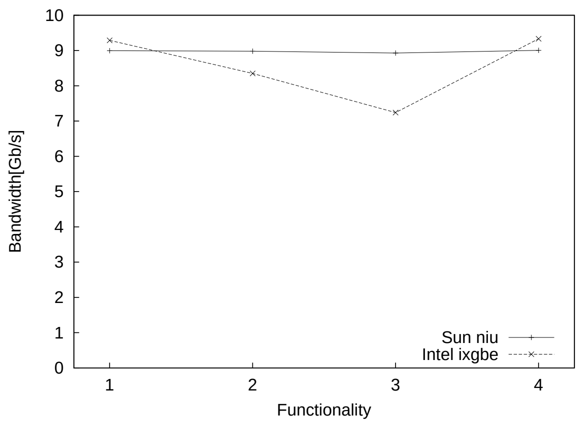

Figure 9. Throughput bandwidth in Gb/s with different routing configurations as specified in Section IV-B.

## C. Experiment 3: Throughput vs functionality

In the third experiment, the functionalityof the router was varied. The netfilter and connection tracking modules were loaded and a full BGP routing table was loaded with 280 thousand routes. The results are shown in Figures 8 and 9.

It can be seen that the performance is reduced somewhat, the results indicate that the performance is maintained, also in the presence of large routing tables and filter modules. The SUN and Intel cards behave somewhat differently, it seems that the ixgbe card, for example, is affected more when loading the modules.

## VI. 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 very close to 10G/s with packet distributions that is close to what is seen on many Internet links.

In particular, our results show realistic Internet traffic being forwarded in close to wirespeed by load-balancing the traffic over multiple CPU cores. New advances in network interface cards, multi-queue technology, multi-core CPUs and high-speed buses combined with new software has made this development possible. Software advances include parallelizing of the forwarding code for use in multi-core CPUs, as well as support for interrupt affinity and queue handling.

When obtaining such results, it is important to tune software, configuring interrupt affinity to avoid cache misses 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.

During the process of this project, the Linux kernel has improved its performance with respect to parallelized forwarding quite drastically, both in general forwarding as well as driver specific code. This effort has been made by the Linux community as a whole, where we have contributed partly by code patches, partly by measurements and profiling of the source code.

In particular, support for the new multiqueue capable cards have been implemented in the Linux networking stack. We are now at a point where additional CPUs visibly improves performance, which is a new property of forwarding since additional CPUs used to result in decreased network performance.

This is a continuing process, with hardware vendours constantly developing new and more efficient classifiers. With this paper we also see new challenges, including closing the last 10% gap between the measured 9 Gb/s and full wirespeed forwarding. With more detailed experiments, we plan to identify the reasons for this and try to remove them if possible.

We also hope that the performance breakpoint that we identified at four CPU cores can be raised, maybe by allocatating cores in a more fine-grained manner, so that, for example, a subset of CPU cores can be allocated for traffic in one direction, and another subset of CPUs in another.

A key issue to achieve high performance is how to distribute input load over several memories and CPU cores. The cards used in this paper implements the Receiver Side Scaling proposal [5] where input is distributed via separate DMA channels to different memories and CPU cores using hashing of the packet 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 more powerful hardware classification, it is possible to go a step further in functionality and performance by off-loading parts of the forwarding decisions to hardware. Some cards provide more powerful classification mechanisms, such as the TCAM on the SUN niu. This shows an interesting path for future work, and we plan to continue our work by looking at classification using such more powerful classification devices, in particular how the Linux kernel and its associated software can benefit from such usage. By using a TCAM, packets can be dropped or tagged by hardware thus enabling wire-speed filtering and prioritization, for example.

A specific point is how to best serialise several transmit queues while causing minimal lock contention, cache misses and packet reordering. The receiver side is relatively simple in this respect as the classifier splits the incoming traffic. The transmit side is by design a congestion point since the packets comes from several interfaces that may potentially cause blocking.

*Acknowledgements*

This paper is a result of a study sponsored by IIS. Interface boards were provided by Intel and SUN, CPUs by AMD. Uppsala university and KTH for providing experimental networking infrastructures.

## REFERENCES

[1] O Hagsand, R.Olsson., B. Gorden *Towards 10Gb/s open source routing*. In Proceedings of the Linux Symposium, Hamburg, October, 2008

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

[3] 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.

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

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

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

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

[8] 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.

[9] , "Packet size distribution comparison between Internet links in 1998 and 2008: WIDE MAWI", http://www.caida.org/research/traffic-analysis/pkt_size_distribution/graphs.xml, Caida, 2008.

[10] Milnert, "Packet size distribution in Chalmers student network", Bifrost mailing list, Dec, 2008.

[11] Pedersen, "Packet size distribution of Uppsala university uplink", Bifrost mailing list, Dec, 2008.