Abstract—High speed routing with 10Gbps link speed is still very expensive in the commercial world. Network equipment producers such as Cisco, Juniper, etc. provide high performance 10Gbps routing equipment with high price equally. If you need to push large amounts of traffic among networks, but only have a small budget, you will be out of interest in those commercial products. However, there is still alternative based on the fact that network interfaces, I/O bus and processor architecture of end node have reached theoretical bandwidth to achieve 10Gbps throughput. By fully utilizing available system resources, the goals we are trying to achieve is to build a low-cost Linux based software router with 10GE network cards and probe performance of such equipment under different experiments. The issues related can be summarized into three aspects: i) Utilizing 10GE network interface cards mounted on I/O bus of commodity PC to build a routing equipment. ii) Installing latest version of Linux operating system kernel and network interface driver to set up software routing environment. iii) Tuning optimal oriented performance options from kernel to interface card to show difference between theoretical ideas and practical results.

I. INTRODUCTION

Nowadays 10-GigE technology are becoming a well established Ethernet standard, while 100-GigE solutions are already being actively developed [1]. This rapid trend on developing high speed bandwidth network devices is boosted by the drastic demand of large scale data communication in an exploding modern network. For example, front-end servers of data-centers are equipped with 10 Gigabit Ethernet (10GigE) to support large number of clients and provide high bandwidth connection to clients [2].

Therefore, it is possible to build a low cost and efficient software router with open source OS kernel and full network stack. Of course, the routing capability will be decided by the fact that how many 10-GigE network interface cards would be enabled on a host. However, with the introduction of 10-GigE, network I/O re-entered the fast network, slow host scenario that occurred with both the transitions to Fast Ethernet and Gigabit Ethernet [3]. Specifically, three major system bottlenecks may limit the efficiency of high performance I/O adapters [4], e.g. our 10-GigE network interface cards on a host: the PCI-X bus bandwidth, the CPU utilization and the memory bandwidth. In the last years a rapid development involved these items. The performance of the PCI-X bus [5] operating at 133 MHz, which has a peak bandwidth of 8.5 Gb/s, has been overcome by that of the PCI-Express (PCIe) bus [6], that has a peak bandwidth of 20 Gb/s using 8 lanes. CPUs have definitely entered the multi-core era and the memory data rate has increased from e.g. 3.2 GB/s (single channel DDR-400 memory) of the peak transferring rate running at 400 MHz to 10.7GB/s (dual channel DDR-667 memory) running at 667 MHz from the AMD integrated DDR2 memory controller technology.

After a brief description of CPU, PCI-Express, on-board chip-set hardwares and overview of how the Linux kernel handles the transmission and reception of network packets as background in Section II, we will describe the setup of the testbed and measurement result in Section III. Then, the rest of this project report will show the discussion of related work in Section IV. Finally, we will conclude our achievement and discuss about work progress in the future.

II. BACKGROUND

In this section, we describe modern hardware and software technologies on high bandwidth low latency communication such as CPU, I/O bus, 10-GigE network interface and network stack in Linux kernel.

A. Multi-Core AMD Opteron Processor

What we are interested is bandwidth, so the throughput of CPUs is the main concern. The second generation of AMD Opteron multi-core processor [7] at our hands provides features and benefits like:

- AMD64 technology. It lets 64-bit operating systems provide full, transparent, and simultaneous 64-bit platform application multitasking. This means network stack would escape from the memory utiliza-
tion constraint of 32-bit platform (less than 4 GB memory) to theoretically endless.

- Direct Connect Architecture. It helps reduce the bottlenecks inherent in traditional front-side bus architectures. This means: Memory is directly connected to the processor, optimizing memory performance; I/O is directly connected to the processor, for more balanced throughput and I/O; processors are directly connected to other processors, allowing for more linear symmetrical multiprocessing.

- Integrated DDR2 memory controller. A 144-bit wide, on-chip DDR2 memory controller provides 128 bits for data and 16 bits for ECC and Enhanced ECC technologies, while providing low-latency memory bandwidth that scales as processors are added. With dual channel DDR-667 memory, the bandwidth is 10.7GB/s.

- HyperTransport Dual Link. A HyperTransport path that runs at speeds up to 1 GHz for up to 8 GB/s of theoretical bandwidth between each processor and each processor’s attached controllers. This means the bandwidth between each multi-core CPU is 18GB/s.

B. On-board I/O bus

As shown in figure 1, the motherboard in the routing host machine is a Sun Fire X4240 performance server. It illustrates the system-level architecture, which provides up to 4 available PCI Express slots with at least 8 lanes each. This means 20Gbps bandwidth guaranteed per slot theoretically.

C. Myricom 10-GigE Network Adapter

The hardware specification about Myri-10G Ethernet network interface adapter at our hands shows this product:

- 10+10 Gbit/s data rate, full-duplex transmission
- supports Ethernet flow control as defined by IEEE 802.3x
- fits mechanically in x8 or x16 physical slots with auto-negotiates operation in the widest available mode

Additional special options will not be shown fully here, but some of them are really inspiring us on tuning performance on a software router.

- Available bus bandwidth (read DMA, write DMA, and simultaneous read and write DMA) calculation when loaded with interface driver.

D. Network Stack in Linux Kernel

The Linux kernel splits the process of sending and receiving data packets through the network stack into different tasks, which are differently scheduled and accounted, and which can be partially distributed over several CPU cores on multi-processor or multi-core architectures. The statistics of the kernel accounting partitions relevant to network processing—in short User, System, IRQ and SoftIRQ—relative to each CPU core are accessible, as the number of jiffies (1/1000th of a second) that the CPU core has spent in each different mode.

The network adapter driver transmit function is in charge of transferring the packet from the kernel memory to the adapter buffer, which we call DMA mapping to help reduce overhead. As the DMA transfer has completed, the network adapter informs the kernel through a hardware interrupt (DMAdone) so that the kernel can
free the buffer used for that packet by a SoftIRQ.

Vice versa, upon packet reception, once network adapter received a bunch of Ethernet frames and stored them in a buffer—a DMA transfer from the buffer to a reserved space in the kernel memory, it will knock at the door as the DMA transfer has terminated. Then, adapter notifies the kernel of the new available packets by means of an interrupt signal raised on a dedicated IRQ line, so that the Interrupt Controller issues an interrupt to the dedicated processor. The kernel reacts to the IRQ by executing a hardware interrupt handler.

The above described mechanism, known as part of NAPI (New Network Application Program Interface) [8] has been introduced in the 2.6 kernel series. Its main feature is to converge to an interrupt-driven mechanism under light network traffic (reducing both latency and CPU load) and to a poll mechanism under high network traffic (avoiding livelock conditions: packets are accepted only as fast as the system is able to process them). Which inspires a clue that the system could be in principle handling packets faster than the device, somehow [9]. And this is why we are going to focus on testing the impact of batch mode in both Linux kernel and network adapter.

### III. MEASUREMENT AND OPTIMIZATION

In this section, we present the process for improving the performance for the Linux-based router on the CRON Testbed. First, we measure the performance for the current router as a basis for comparison. Second, we measure the influences for individual optimization options in the NIC driver, including packet batching, interrupt coalescing, and packet pacing. Finally, we try to find a combination of the optimization options to improve the performance for the router.

#### A. Environment Configuration

We create an experiment on the CRON testbed as shown in Figure 2. The configuration for the experiment can be stated in three major aspects, the hardware for each node, the software on each node, and the network between each node.

For the hardware for each node, all PCs and the router are Sun Fire X4240 Servers with two AMD Opteron Model 2384 (2.7GHz/512KB AMD64) quad-core processor 8 GB/s bus speed and 8 GB (4 x 2GB) DDR2-667 single rank memory. Each PC has one Myricom Myri-10G 10G-PCIE-8AL-S/10G-PCIE-8BL-S PCI Express NIC while the router has four [10].

For the software on each node, we focus on the kernel for the OS, the driver for the Myricom NIC, and the measurement software for manipulating the network traffic. The operating systems on each machine is the Ubuntu 10.04.01 LTS Server 64-bit version with the latest Linux Kernel Version 2.6.36. The driver for the Myricom NIC is the Myri-10G Driver Version 1.5.3 with the Ethernet Maximum Transmission Unit (MTU) set to 9000 bytes. The measurement software is the IPerf Version 2.0.4 from the Ubuntu repository.

For the network between each node, we care about the topology for the network and the data flow in it. The topology for the network is a star-shape topology. Four Sun Servers, the PCs, as end nodes connect to one Sun Server, the Router, as a software router in the center. In order to measure maximum performance of the router, we generate multiple 10Gbps UDP traffic to saturate all its NICs as shown in Figure 2. All segments with the same color represent one unique data flow. In other words, PC1 and PC2 send one 10 Gbps flow to each other through the router respectively. PC3 and PC4 send one 10 Gbps flow to each other through the router respectively. The reason for using UDP traffic is that it can keep sending data at the constant rate of 10 Gbps because it does not have any congestion control mechanisms for automatic rate adjustment.

#### B. Measurement And Analysis

Prior to enabling any optimization options, we measure the performance for the router with the default parameter settings in Table I [11] by varying the Transport Layer Maximum Segment Size (MSS). The MSS values used are 64 bytes, 256 bytes, 1024 bytes, 1500 bytes,
4096 bytes, 6500 bytes, and 8972 bytes. The reason for choosing 8972 bytes as the maximum MSS is that each Transport Layer Segment is prefixed with a 28 bytes Network Layer Header before arriving the Link Layer, Ethernet. The total size for the Network Layer Datagram, 9000 bytes, just fits the Ethernet MTU so that there is no fragmentation which introduces additional overhead. The measurement result is shown in Figure 3. This result is a basis for future comparisons.

<table>
<thead>
<tr>
<th>net.core.rmem_max</th>
<th>16777216 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>net.core.wmem_max</td>
<td>16777216 bytes</td>
</tr>
<tr>
<td>net.core.netdevmax_backlog</td>
<td>250000 bytes</td>
</tr>
</tbody>
</table>

**TABLE I**

**DEFAULT PARAMETERS**

In the following three sections, we individually test the three optimization options, packet batching, interrupt coalescing, and packet pacing, for the NIC driver to find out how each of them affects the performance of the router.

### C. Packet Batching

Packet Batching allows the CPU cores to postpone the packet processing until the amount of packets in the kernel exceeds a certain value, N, instead of processing any number of packets per interrupt. N is adjustable in both the NIC driver and the Linux kernel. The two values should be the same. The default N for the Linux-based router on CRON Testbed is 64. In other words, the CPU on the router does not process the incoming packets until there are at least 64 packets received by the kernel. We double the N to 128 and measure the performance of the router. The new router outperforms the default one in most cases. We observe that the each CPU core involving in packet forwarding has 100% utilization rate. Batching more packets alleviates the overhead of alternatively processing the interrupt and received packets repetitively, which is a possible reason for packet loss. The result is shown is Figure 4.

![Fig. 4. Packet Batching Performance](image)

### D. Interrupt Coalescing

The Interrupt Coalescing prevents the NIC from interrupting the CPU cores in an unnecessarily high frequency, which is another possible cause for packet loss and performance degradation. The default value for the interval between consecutive interrupts is 75 μs. We change the interval to some values, 50 μs and 100 μs, recommended by the Myricom but cannot see any apparent variation for the router performance. The utilization rate for each CPU core for packet forwarding is still around 100%. While we set the interval to 1000 μs, the CPU utilization drops to 60%, but the performance for the router also dramatically decreases. Only tuning this option does not improve the performance. We also find a variation for Interrupt Coalescing called “Adaptive Interrupt Coalescing”. It enables the driver to search for the most suitable interval between the 0 μs and the default interval. This function is especially helpful for optimization.

### E. Packet Pacing

The Packet Pacing is the mechanism at an end system which accepts arbitrary traffic with given delay constraints and smoothly releases it into network satisfying the constraints [12]. It prevents end systems from
overwhelming the network with enormous traffic at an instant. The Packet Pacing function in the Myri-10G NIC driver is called "Throttle". A user limits the maximum transmission rate, \(tx\_speed\), by specifying a Throttle Factor. The relationship between the transmission rate and the Throttle Factor is illustrated in Equation 1.

\[
tx\_speed = 256 \times \left(\frac{\text{RawPCIeBandwidth}}{\text{ThrottleFactor}}\right)
\]

The Raw PCIe Bandwidth on all machines is 2100 MB/s. The available Throttle Factor ranges from 416 through 8191. Derived from the equation, the transmission rate ranges from 525 Mbps through 10 Gbps.

We measure the impact of the Packet Pacing by setting the Throttle Factor on each sender PC to 416, 512, 1024, 2048, 2094, and 8191. The MSS on each sender PC is fixed to 64 bytes. We expect that fewer packets are injected into the experimental network so that the contention in the router is alleviated and the performance for the router gets better. The result is shown in Figure 5. However, with each Throttle Factor, there is no apparent performance change compared to the performance for the default router, around 670 Mbps.

We look into the outputs from all machines and identify the cause for this phenomenon. The IPerf on each sender PC generates traffic consisting of 64-byte UDP packets at only around 220 Mbps. At the same time, the CPU core which is in charge of generating traffic already works at maximum speed. The amount of traffic cannot increase with the current software and hardware architecture. We plan to change the IPerf to lessen its CPU time consumption so that the modified IPerf can generate more traffic with the current CPU. Since the effect for the Packet Pacing is restricted by the current system architecture, we do not take this function into consideration for our optimization work.

F. Optimization And Analysis

After examining the three major optimization options, we combine the Interrupt Coalescing and the Packet Batching for optimizing the performance for the Linux-based router on the CRON Testbed.

For the Interrupt Coalescing, we adopt the Adaptive Interrupt Coalescing function. The default interrupt interval is set to 1000 \(\mu s\) so that the NIC driver automatically chooses the appropriate value between 0 \(\mu s\) and 1000 \(\mu s\) to be the interrupt interval. The range includes the commonly used values, 50 \(\mu s\), 75 \(\mu s\), and 100 \(\mu s\), for 10 Gbps networks. It also includes the value that surly decrease the CPU utilization, 1000 \(\mu s\). We rely on the driver to determine a balanced value.

For the Packet Batching, we set the number of batched packets to 96, 128, and 160 in order to probe the most effective value. We evaluate the performance for each batching number by running the same test for evaluating the performance for the default settings.

The result is shown in Figure 6. Batching 128 packets with Adaptive Interrupt Coalescing has the best average performance. We suggest that the CRON administrators configure their default Linux-based routers using this combination of optimization options.

IV. RELATED WORK

Recently, there are lots of work presented in area of High Speed Software Router. PacketShader [13], a high-performance software router framework for general packet processing with Graphics Processing
Unit (GPU) acceleration. PacketShader exploits the massively-parallel processing power of GPU to address the CPU bottleneck in current software routers. Combined with their high-performance packet I/O engine, PacketShader outperforms existing software routers by more than a factor of four, forwarding 64B IPv4 packets at 39 Gbps on a single commodity PC. For high performance network packet processing on commodity hardware. They minimize per-packet processing overhead in network stack and perform packet processing in the user space without serious performance penalty. On top of their optimized packet I/O engine, Packet-Shader brings GPUs to offload computation and memory-intensive workloads, exploiting the massively-parallel processing capability of GPU for high-performance packet processing.

Also, there is one other work focused on the performance optimization of TCP/IP over 10 Gigabit Ethernet [14]. They observed that the followings caused performance depression on Long Fat-pipe Networks (LFNs): short term bursty data transfer, mismatch between TCP and hardware support, and excess CPU load. They have established systematic methodologies to optimize TCP on LFNs. Including, utilizing hardware-based pacing to avoid unnecessary packet losses due to collisions at bottlenecks, modifying TCP to adapt packet coalescing mechanism, and modifying programs to reduce memory copies. They have achieved a constant through-put of 9.08Gbps on a 500ms RTT network for 5h.

V. FUTURE WORK

Our future work for High Speed Software Router consists of mainly three parts. The first part is to develop a high speed low overhead network measurement tool. In our experiment, we use Iperf, Nuttcp, and NetPerf as our network measurement tools separately, the result shows Iperf can get back better performance than other two. However, in case of small size packet traffic in high speed network, all these application level network measurement tools tend to do all kinds of odd things in its critical path with timers and threads since all these network measurement tool try to measure the network performance based on sockets, which will create large overhead so that the NICs are not able to handle. Thus, using a in-kernel packet generator but not sockets to measure the network performance of small size packet traffic over high speed network is critical to get accurate performance results.

The second is to continue our work on tuning and optimization for the hardware and software on our High Speed Software Router. For hardware future work, since we are using AMD CPUs with 8-cores, CPU binding will be very important, and will have a large impact on the results. We need to consider the CPU core binding with the driver’s interrupt and the our network measurement tools so that the CPU usage could be evenly distributed into the CPU cores. Also we need to consider carefully the host server hardware architecture, we use the Sun Fire X4240 server, which provides up to 4 available PCI Express slots with at least 8 lanes each. This means the backbone bandwidth on the host server will be limited by 20 Gbps bandwidth per slot theoretically. Besides, we need to continue our work on optimize the operating system, including optimize Linux operating system network stack, huge packet buffer, batch processing, interrupt coalescing and so on.

The last future work is to test the other software router Click [15]. Click is a modular software architecture for building flexible and configurable routers based upon general-purpose processors. A Click router is assembled from packet processing modules called elements. Individual elements implement simple router functions such as packet classification, queuing, scheduling and interfacing with network devices. As we use Linux default reference router in our experiment, we need to check that if other software routers, such as Click router, may provide us better performance in compare to the Linux default reference router. However, according to our analysis of the current work, the critical bottleneck of our High Speed Software Router is at the sender and receiver side, we did not change the software router type in our router box.

VI. CONCLUSION

In this work, we fist present the necessary for us to develop a High Speed Software Router since nowadays 10-GigE technology are becoming a well established Ethernet standard. We describe the background of our hardware and software technologies used on our High Speed Software Router, including high bandwidth low latency communication for CPU and I/O bus, 10-GigE network interface and network stack in Linux kernel. In order to find the possible bottleneck in our High Speed Software Router, we introduce the Multi-Core AMD Opteron Processor, the On-board I/O bus, Myricom 10-GigE Network Adapter, and the Network Stack in Linux Kernel.

We then present the process for improving the performance for the Linux-based router on the CRON Testbed,
including measurement of the performance for the current router as a basis for comparison and the optimization on the NIC driver to measure the inuences for individual optimization options. After introducing our test environment conuration and the measurement method, we finished optimization options of packet batching, interrupt coalescing, and packet pacing. As we can see by varying the Transport Layer Maximum Segment Size (MSS), the performance of our High Speed Software Route varies a lot due to the processing capability of the router. After examining the three major optimization options, we found that the Interrupt Coalescing and the Packet Batching will have more impact on the performance for our High Speed Software Route.

Finally, we summarize the related work according to the previous work in terms of the implementation and the analysis of Software Route, including PacketShader [13] Software router, and the optimization of TCP/IP on 10Gbps Ethernet [14]. Furthermore, we finalize our future work as three parts, to develop a high speed low overhead network measurement tool, to continue research on tuning and optimization on High Speed Software Router, and to test the other software router Click in compare of our High Speed Software Route.

REFERENCES


