High Performance IP Forwarding Using Host Interface Peering

Joe Touch, Anne Hutton, Simon Walton, Stephen Suryaputra

USC/ISI Computer Networks Division

Abstract

The ATOMIC-2 project at USC/ISI has installed a production high-bandwidth LAN based on Myrinet, a 640 Mbps LAN technology. This work includes the development of an efficient means of IP forwarding on a host-based gateway, because router interfaces for Myrinet are not available. This work presents measurements which show the limitations of a standard high-bandwidth network interface driver and the capabilities of an enhanced driver that uses peer DMA. Peering involves the direct transfer of packets from the incoming network interface card (NIC) to the outgoing NIC, rather than staging packets through host memory.

In these measurements three implementations of the NIC driver are compared: an high-performance conventional driver and two variations of peer DMA. The conventional driver stages packets by copying them into and out of host memory, requiring two transfers per packet, but often requiring fragmentation and reassembly into host memory mbufs. This was optimized to pre-allocate extended mbuf clusters in the host kernel, to avoid this overhead. Preliminary measurements indicated that bus bandwidth was a limitation, and that techniques to avoid transferring the data twice over the same bus would be useful.

The peer DMA variants are designed to transmit packets once over the bus, from the incoming NIC to the outgoing NIC directly. Custom host-based routers support such peering by performing IP forwarding in the host interfaces directly, however we sought a way to support peering using conventional kernel-based forwarding. In the first peer variant the entire packet remains on the incoming NIC and the header is read word-wise by the CPU over the bus. In the second variant the IP header alone is copied into the host's main memory and the packet's payload remains resident on the incoming NIC. The header copy reduces the bus contention that word-wise accesses create in the first variant, by transferring the entire header in one operation.

Since our investigation of the optimization of host-based routers focuses on optimizing the data path we first present some background. We discuss the way conventional host-based routers process packets, and the two ways we optimized that processing, by reducing an extra data copy. We then present our results, showing how we increased data throughput by over 40%. Finally, we discuss the implications of this optimized routing, and how it affects network interface design and host operating system software.

For our testbed, we use 200 Mhz Pentium Pro PCs, running FreeBSD 2.2.5.[2] These hosts also have a 33 Mhz, 32-bit wide PCI bus, used for high-bandwidth peripheral access. In experiments performed elsewhere, the combination of these PCs and the Myrinet NICs have been shown to provide host-memory to host-memory data transfer rates in excess of 1 Gbps, using very large transfer units (64 KB and larger), pipelined directly into the NIC DMA. However, at the IP layer these hosts support UDP rates of 300 Mbps and TCP rates of 150 Mbps.

For our experiments, we relied on the packet multiplexing capability of the Myrinet switches to merge traffic from multiple sources, to find the limiting routing bandwidths. We found that as a host-based router, the hosts provide bandwidths near 335 Mbps, using existing production drivers and the FreeBSD routing software. This rate is not substantially higher than the single-host data source limit of 300 Mbps. We knew that existing host routing copied data twice across the peripheral bus, and that this would limit the maximum bandwidth to 500 Mbps; we felt that reducing the number of copies would alleviate this contention, and possibly increase the throughput of the system.

It is well known that memory performance in workstations has not paralled improvements in processor performance or network bandwidth. Much work has been done on techniques to minimize data copying by bypassing or reducing the level of operating system intervention and promoting direct application to network interface interaction- Several techniques have been introduced to minimize the number of times network data crosses the CPU/memory data path, notably hardware streaming [1] and kernel-level streaming [3]. In hardware streaming data is transferred from source to sink device without CPU involvement. This avoids the extra copy into kernel RAM, but is focused on

inter-peripheral communication, such as video-to-disk. In network routing, the host CPU still needs to process the packets between the input and output devices; whereas such intermediate processing is entirely avoided in hardware streaming. Kernel level streaming transfers data from source to sink over the system bus without an application process being included in the data path. Data does, however, pass through memory and the technique relies on the Sytem V STREAMS interface.

In a conventional host-based router, routing processing is performed inside the host, by the CPU. When a packet arrives, the NIC signals the host, which signals the DMA on the NIC to proceed with a copy of the packet from the NIC RAM into the host RAM. The host CPU then processes the packet, examining its link and IP header, and determining the appropriate outgoing interface, outgoing link header, and any modifications to the IP header. The host packet copy is then modified appropriately, and the outgoing NIC then DMAs the packet into its RAM, and emits it on the outgoing link. However, in the case where routing (forwarding) to an outgoing interface is the dominant case, the conventional data path is not optimal. The packet is copied twice, once into host memory, and once out of host memory.

A better algorithm would permit packets to be transferred directly from incoming NIC to outgoing NIC, without the extra copy. This would reduce the I/O channel load, free host resources, and reduce the transfer latency through the router. The I/O channel would have reduced signalling load, because the channel arbitration is accessed only once per packet, rather than twice.

We examined two different ways of supporting peer DMA routing. In both methods, packets remain in shared memory on the incoming NIC, and are eventually peer-DMA'd directly to the outgoing NIC. The host continues to perform the routing functions, and so needs access to the packet's link and IP headers. Both methods assume that the NIC contains a reasonable amount of packet buffer space, and that space is sufficient for queuing packets while they are routed and waiting to be output. Neither method involves changes to kernel code only to the network interface driver.

In the first variant, the entire packet is left in the NIC, and the host accesses these headers by reading its fields directly over the PCI bus. Each field is read separately, as a shared-memory access. In the second variant, called 'HEADER COPY', the link and IP headers are copied, in full, into the host memory, where they are manipulated by the host using local RAM accesses. The modified headers are copied out over the original packet's header, then the result is DMA'd in its entirety to the destination NIC. Both techniques required modifications only to the device driver, and the changes affect packet transfers only on the incoming NIC.

Both of these techniques require special processing for ARP and ICMP packets. These packets are usually associated with low-level control exchanges, often handled within the driver software. Many such packets generate an immediate response, which typically overwrites the incoming packet, and is rapidly sent back out. As a result the peer DMA methods would request a transfer from the incoming NIC back to itself; although this operation is not prohibited by the PCI bus specification, it is often not supported, and can lock the bus and freeze the host. This case is easily handled by detecting the special case, and effecting the peer DMA by pointer manipulation, rather than by an excess self-copy.

The standard driver, in general, increased only 10% using multiple sources, as compared to a single source. For HEADER-COPY, the design finally selected, the CPU was pegged for small packets and multiple sources, and per-packet overheads dominate for small packet sizes. This is more easily seen in the packets/second graph. The packet rate is nearly constant for packets under 1 KB, around 12,000 packets/second, indicating that either interrupt processing or routing algorithm processing causes resource contention at the CPU. For larger packets, costs proportional to packet size dominate, suggesting backplane resource contention. By avoiding the additional packet copy, throughputs and packet rates are increased substantially, up to 45%, to 480 Mbps.

Peer DMA achieves an IP forwarding throughput in excess of 480 Mbps using 8K packets, more than a 40% improvement over non-peer forwarding. The CPU utilization drops with large-packet peering, from 90% down to below 70%, but remains over 90% for small packets (below 4K), regardless of method. Packet forwarding rates increase for peering with packet sizes over 1K and above, linearly increasing as the packet size doubles, from a 10% improvement at 1K to a 30% improvement for 8K packets.

This research indicates that peer DMA support in the drivers can substantially improve the IP forwarding bandwidth, and reduce the CPU utilization, for large packet sizes, i.e., over 4K. The peering also requires substantial packet buffering on the NIC, to store incoming packets while the headers are processed in the kernel. This has implications on high-performance NICs with low packet sizes, i.e., gigabit ethernet, and NICs without significant packet-

based buffering, i.e., some current 100 Mbps ethernet and OC-3c ATM NICs.

However, when a peer DMA capable driver was used for packets destined for the router itself, i.e., for the end host, throughput was substantially reduced, compared to the standard driver. The optimized driver should be used only for host-based routers where the dominant traffic is through, rather than into, the host. This is not different from the effects of user protocol optimizations, so-called single-copy stacks, which optimize throughput into user space memory; for similar reasons, such systems are likely to be ill-suited to host based routers. For hosts with mixed traffic, a combination of these two systems, with early demultiplexing (packet labels) indicating which algorithm to select, is one possible design alternative.

We assume that NICs support DMA, and that each NIC supporting incoming traffic also contains shared memory sufficient for packet queuing. In addition, our experiments used only Myrinet interfaces, because available fast ethernet and ATM NICs did not support equivalent shared memory use. If the NIC lacks sufficient shared memory, the host memory must be used as a staging area, defeating the peer-DMA altogether. Extensive memory can also be required when NIC link rates vary widely, to be used for line rate matching, or if advanced queuing algorithms are supported, such as priority queues or early discard queuing.

We also assume that fragmentation is either not required or is trivial. This requires that MTUs do not vary largely between the NICs.The NICs are assumed to deal in units of packets, the same packets as processed by the host routing algorithm. This latter case may not be true when packets are aggregated for link transmission, as in IP over SONET or gigabit ethernet.

We assume that packet data is ignored by the host CPU. This is not the case in Active Nets[4], or when data is transcoded, reformatted, segmented and reassembled, authenticated or encrypted.

As discussed earlier, the peer DMA techniques reduce peripheral system resource contention. In our experiments, this resource is a shared bus with 1 Gbps capacity. When using a switched backplane, or when a shared bus backplane has higher throughput, these techniques are not needed to reduce peripheral resource contention. They do continue to reduce bandwidth use into and out of the host RAM, and to reduce use of that RAM.

Finally, there is the possibility of moving the IP routing algorithm into the NIC itself, completely obviating the need for host CPU access to the packet headers. This would require copying the routing tables and algorithms into the NIC, as well as sufficient general-purpose processing capability to handle the routing algorithm. This somewhat complicates the shared queuing model of existing routers, and may interfere with policy-based routing and queuing, as well as resource reservation. It remains a possibility, but not for Myricom NICs, whose processors are completely loaded running the link-level protocol and packet management algorithms.

Current systems are packet processing limited; a combination of streamlined routing algorithms and aggregate interrupt processing should further increase host-based capability. Moving some of the IP processing out to the NIC co-processor may enable this, where co-processing is available. It is also apparent that as processor speeds increase the advantages of peer DMA will aid throughput for small packet sizes.

[1] Druchel, P., Abbot, M., Pagels, M., and Peterson, L., “Network Subsystem Design: A Case for an Integrated Data Path,” IEEE Network, Vol 7, No 4, pp 36-43, July 1993.

[2] McKusick, M., Bostic, K., Karels, M., and Quarterman, J., *The Design and Implementation of the 4.4 BSD Operating System*, Addison Wesley, 1996.

[3] Murphy, B., Zeadally, S., and Adams, C., “An Analysis of Process and Memory Models to Support High-Speed Networking in a UNIX Environment,” USENIX, Proceedings of the Technical Conference, pp 239-251, Jan 22-26 1996.

[4] Tennenhouse, D.L., Smith, J.M., Sincoskie, W.D., Wetherall, D.J., Minden, G.J., “A survey of active network research,” *IEEE Communications Magazine*, pp. 80-86, Jan. 1997.