Efficient High-Speed Data Paths for IP Forwarding using Host Based Routers

Simon Walton, Anne Hutton, JoeTouch[†]

USC / Information Sciences Institute

Abstract

Host-based forwarding uses general purpose computers with two or more network interfaces to act as a router. These routers offer certain advantages over the use of dedicated hardware, allowing open, public source code access to the forwarding, queuing, and routing algorithms, and the use of more flexible, commodity host interfaces and host CPUs. The main drawback of host-based forwarding is its inefficiency in supporting high-bandwidth interfaces; although host I/O busses support gigabit throughputs, existing host routers achieve only 25% of this capacity. This paper examines a version of host-based forwarding that substantially increases the capacity of host-based routers, by transferring packets directly between host interfaces, rather than staging them through the host memory. On a 200 Mhz Pentium-Pro FreeBSD PC, the resulting system supports bandwidths in excess of 480 Mbps, a 45% increase compared to conventional techniques.

KEYWORDS: host routers, IP forwarding, data paths, peer DMA, router performance

1.0 Introduction

Host workstations are increasingly being used as routers. Using commodity platforms and network interfaces, these host-based routers can be cheaper, and allow more flexible configuration and programmability than production routers. This work optimizes the data path in a host-based router, to increase routing bandwidth, while reducing backplane bandwidth and CPU load.

Production routers are special-purpose systems, which use custom backplanes and line interface cards, with closed, proprietary software. These systems are often limiting, because of

†. This work is supported by the Defense Advanced Research Projects Agency through Ft. Huachuca contract #DABT63-93-C-0062 entitled “Netstation Architecture and Advanced Atomic Network”. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Department of the Army, the Defense Advanced Research Projects Agency, or the U.S. Government.

their closed software and proprietary backplane bus. Systems requiring software modification, such as experimental routing testbeds (e.g., DARTnet and CAIRN) and dynamically-reprogrammable routers (Active Networks [7]), typically require host-based routers.

Host interfaces for new gigabit technologies are typically available much earlier than their router line-card equivalents. In some cases, such as the Myrinet, no router line cards are available. In addition, the router line-cards are vendor, and sometimes product specific. Host interfaces focus on commodity standards, such as PCI and VME busses. As a result, host-based routers provide the earliest, and sometimes only way to interconnect new technologies among themselves, and to other existing technologies.

Our investigation of the optimization of host-based routers focuses on optimizing the data path. First, we 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 25% while reducing CPU load in the host. Finally, we discuss the implications of this optimized processing, and how it affects network interface design and host operating system software.

2.0 Background

The University of Southern California Information Sciences Institute’s (USC/ISI) Computer Networks Division current runs the largest production Myrinet LAN, a 1.28 Gbps mesh-connected network co-invented by USC/ISI and the California Institute of Technology (CalTech), and spun-off to a company called Myricom. The Myrinet provides full-duplex gigabit bandwidth using cables composed of dual simplex byte-wide copper links. At ISI, this LAN is connected to a host-based router, interconnecting 53 office workstations to OC-3c (155 Mbps) ATM and fast ethernet (100 Mbps).

The Myrinet links are connected to LAN switches, each providing 8 ports of full-bandwidth cross-connect. The switches use hardware signalling to provide backpressure to the source, and route packets inside the LAN using source-routed, cut-through packet switching. Myrinet link packets carry up to 8,432 bytes of payload.

At the link layer, the LAN emulates an ethernet, using ethernet-like link addresses. The LAN supports broadcast by serially copying the outgoing packets at the source interface, sending a copy to every destination interface on the LAN in sequence. Multicast is provide by broadcast, using driver-based select inside the host. The Myrinet network interface cards (NICs) contain 256 KB (kilobyte) of shared memory (512 KB to 1 MB in newer versions), used to contain the interface control program, shared and local variables, and packet buffering. The NIC also contains a 32-bit (20 million instructions/sec.) processor, dedicated to link and LAN control, but not sufficient to participate in other processing. DMA is provided by the NIC across the host PCI bus, in either direction.

For our testbed, we use 200 Mhz Pentium Pro PCs, running FreeBSD 2.2.5 [3]. 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 (gigabits per second), using very large transfer units (64 KB and larger), pipelined directly into the NIC DMA [6]. 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 forwarding software. This rate is not substantially higher than the single-host data source limit of 300 Mbps. We knew that existing host forwarding 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.

3.0 Prior and related work

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.

First some terminology about routing. Routers relay packets between different networks, using a combination of forwarding and queuing. Forwarding is the process of determining the destination interface and queue of an incoming packet, and includes some header manipulation. Queuing is the process of storing, retrieving (and potentially dropping) packets according to queue, network, and header context. Routing has multiple meanings; it can mean the combined process of forwarding and queuing, or the inter-router process of computing routes used by the forwarding algorithm. As a result, we refer separately to the process of forwarding and queuing here.

Several techniques have been introduced to minimize the number of times network data crosses the CPU/memory data path, notably hardware streaming [2] and kernel-level streaming [4]. 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 packet forwarding and queuing, 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.

There has also been substantial work in avoiding OS intervention by providing user access to network data [9]. Direct application access to the network interface is used to achieve both low-latency and high bandwidth using commodity workstation clusters. Protocol processing is moved to the application by virtualizing the interface using a combination of NIC capabilities (on board buffers, programmable co-processors, DMA) and OS mechanisms. In the case of fast ethernet NICs, which do not have an on-board co-processor, kernel intervention is necessary. The main benefit of the NIC co-processor is to allow examination of the packet header and direct DMA of the data into the user-space buffer reducing the number of data copies. U-Net is currently working on implementing a subset of the Java Virtual Machine for the Myrinet NI - this would allow customized packet processing on the NI and would encourage further use of host based routers in an Active Nets context.

An industry consortium is also extending the concept of virtualizing NIs to provide user-level access to the network interface [8]. The interest is primarily in low latency message passing, which may streamline the interrupt processing of current NICs, and allow outboard co-processors to be controlled and programmed using a standard interface.

Finally, there are a number of techniques for optimizing custom router design, summarized in [5]. Peer-DMA transfers between router line cards has been utilized there, including variations on where whether the header forwarding processing occurred on-card or outboard, on a separate processor. One recent system at BBN shows how this can be accomplished using commodity CPUs in a custom backplane architecture [5].

4.0 Conventional host-based forwarding

This section describes the flow of IP packets in a conventional host-based router. A host router is composed of a host with at least two host interfaces (Figure 1). The host RAM is managed by its CPU, predominantly as a pool of shared memory used by the operating system kernel. The host also has an I/O controller which manages DMA access by the host interfaces over an I/O channel. This channel is conventionally a shared bus, e.g., a PCI bus, but can also be a switched matrix, as in recent workstations from Sun Microsystems and Silicon Graphics.

The network interface cards, or NICs, each have a DMA controller and some internal memory, as well as an interface to their particular link. The DMA controllers manage data transferring from the NIC RAM directly in to the host RAM, without the supervision of the host CPU. For simplicity, only one incoming and one outgoing NIC are shown in these figures.

In a conventional host-based router, forwarding 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. These steps are shown in Figure 1

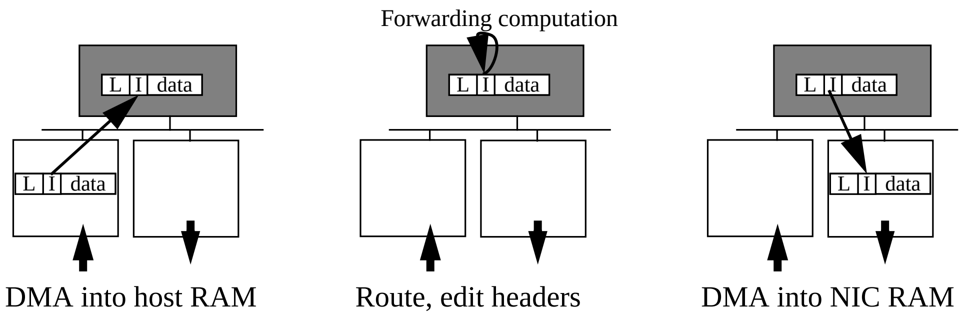

FIGURE 1. Full packet kernel copy, with DMA to/from host memory only

In the forwarding step, both link and IP headers are modified by the forwarding algorithm. The IP header is typically modified only at its hopcount field, but other fields, such as source route, encapsulation, and timestamp options can also be altered. The link header can indicate link level destination (and possibly source) addresses, and can also have additional demultiplexing fields, such as channel identifiers in the Myrinet, or virtual circuit identifiers (VCIs) in ATM. The link header is usually replaced entirely by the router, appropriate for the destination of the outgoing interface link. Data inside these packets, interior to the IP packet, is typically not modified, although it can be examined, encrypted or authenticated.

Buffer management in the conventional host case focuses on main memory storage; buffers in the NICs are used as temporary queues to buffer the link rate to the DMA transfer rates. NIC buffers are used only as long as required to transfer packets into main memory on the incoming path, or out to the link on the outgoing path. All other queuing, and queue management is performed in main memory [3].

These steps are somewhat different if the host is the destination of the incoming packet. In that case, the packet is copied from kernel memory into user memory, based on further demultiplexing of the packet header by the kernel. However, in the case where 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 bus arbitration is accessed only once per packet, rather than twice. If the channel is implemented using a shared bus, the bandwidth over this bus would be cut in half (this is not an issue if the channel is switched). The host memory would not be required, so kernel RAM resources would be increased. The host would also have increased I/O channel access bandwidth for other devices, such as video and disks, because the packet bandwidth over its port is reduced from two packet copies to zero.

5.0 Peer-DMA forwarding

Here we describe an alternative to the packet path in conventional host-based routers. We were not sure whether host PCI bus contention was a significant problem, but wanted to examine the benefits of avoiding the excess packet data copying of the conventional system. The goal is to leave most of the packet in the incoming NIC, until its destination is determined. When the packet is ready to be transferred, it can be DMA'd directly from the source NIC to the destination NIC, without an intermediate copy into the host memory.

This streamlined DMA processing has several benefits. By avoiding the superfluous copy into host memory, shared peripheral resources are economized. In a shared bus, such as PCI, bandwidth use is halved. Even in switched peripheral systems, bandwidth in and out of host memory is significantly reduced. Finally, packets are moved with reduced transfer latency. This lowered transfer latency is typically not noticed in host-based routers, where queuing latencies dominate.

We examined two different ways of supporting peer DMA forwarding. 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 forwarding 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 forwarding and queued for output. Neither method involves changes to kernel code only to the network interface driver.

In the first variant, called 'IN-PLACE', the entire packet is left in the NIC, and the host accesses these headers by reading its fields directly over the PCI bus (Figure 2). 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 Figure 3. Both techniques required modifications only to the device driver, and the changes affect packet transfers only on the incoming NIC.

The two variants differ in complexity and efficiency. IN-PLACE forwarding requires only minor modification to the driver routines, where packet buffer pointers address shared NIC memory rather than kernel RAM in the host. However, each field accessed by the host CPU requires peripheral bus arbitration, and also runs at a slower clock rate than CPU accesses to main memory. In addition, these accesses cannot be cached, because they refer to shared memory. In this case, the bus arbitration overhead is increased.

HEADER-COPY transfers the entire header into main memory when the packet arrives, streamlining peripheral bus access, but using copying fields the host may not access. The header must be copied back out to the incoming NIC before the peer DMA can begin. These steps were not difficult to add to the driver, but not as trivial as the IN-PLACE method. In this case the bus arbitration overhead is slightly higher than the conventional case, because of the additional access for the header copy-back operation.

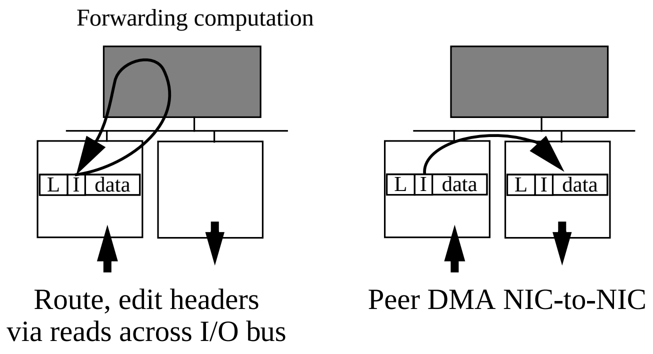

FIGURE 2. IN-PLACE forwarding, followed by peer DMA

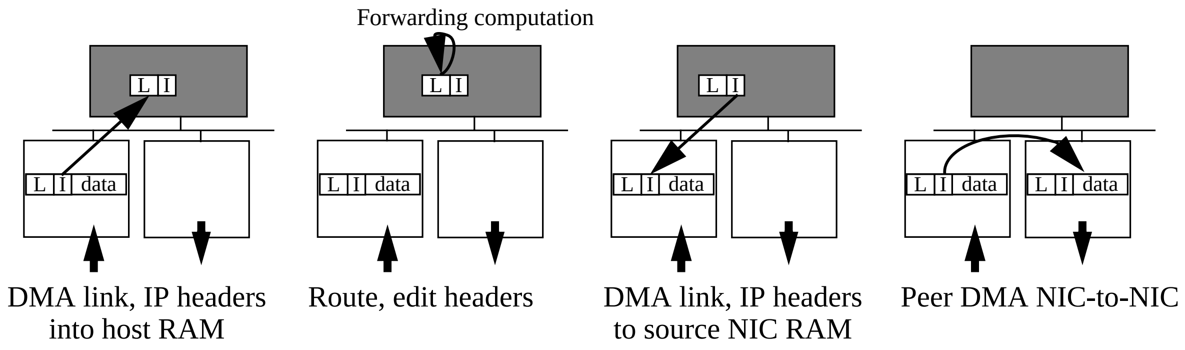
Forwarding computation
L I
L I data
DMA link, IP headers into host RAM
Route, edit headers
DMA link, IP headers to source NIC RAM
Peer DMA NIC-to-NIC

FIGURE 3. HEADER COPY forwarding, followed by peer DMA

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.

6.0 Results

For our experiments, we used two measurement software tools - *netperf* 2.11, and *traffic*. *Netperf* generates both UDP and TCP packets, and *traffic* generates only UDP; for our measurements, we used only UDP packets, because we focus on throughput of the router, and both types of packets are forwarded identically. Neither tool contains flow control, and the send side is typically faster than the receive side for UDP. For CPU load measurement, we used a passive monitoring tool developed at ISI, called *cpuload*.

Our experiments used both single pairs and multiple pairs of hosts sending and receiving, and used only Myrinet NICs to simplify the testbed. A single PC host can send only 300 Mbps UDP without checksums, 275 Mbps with checksums.We found that a PC-based router can easily support these bandwidths, such that hosts directly connected, or routed through an intermediate PC, achieved the same bandwidth. As a result, we used the Myrinet switch's packet multiplexing capability to aggregate streams from multiple sources, and demultiplex them to separate destinations, as shown in Figure 4. These switches are internally non-blocking, and the link rate supports 1.28 Gbps; for up to 4 hosts, the links and switches would provide no contention for our experiments. We used up to three sources and three sinks for our tests; additional sources and sinks were tested, and did not affect the results.

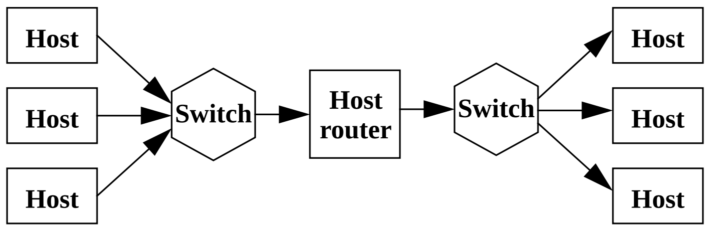

FIGURE 4. Experiment testbed configuration

We measured the three competing solutions to host-based forwarding: conventional, using the standard drivers, IN-PLACE, in which header fields are read over the PCI bus individually, and HEADER-COPY, in which the link and IP headers are copied into host memory prior to forwarding (shown Figure 1, Figure 2, and Figure 3, respectively). We measured throughput in bandwidth (bits/second) and packet rate (packets/second), and monitored CPU load, over a range of packet sizes from 128 bytes through 8 KB. These results are shown in Figure 5. Latency was not measured because routers are dominated by queuing latency, which is not affected by our solution.

First, we measured a single pair of hosts, both with and without an intermediate router. We found that both UDP bandwidth and packet rates were unaffected by the inclusion of the host-based router. The router CPU was less loaded using the standard driver (37%) than either the IN-PLACE (65%) or HEADER-COPY (56%) drivers. Because our system is not otherwise limited, the peer DMA drivers only serve to complicate packet processing, requiring additional effort to access the header over the PCI bus, or to copy the header back out to the incoming NIC before the final DMA. The read-through over the PCI bus in the IN-PLACE driver is higher cost than the HEADER-COPY; it is likely this cost is a combination of bus contention tying up the CPU, and the fact that read-through data cannot be cached.

For two pairs of sources and sinks, the standard driver tops out at 335 Mbps, the IN-PLACE at 419 Mbps, and the HEADER-COPY at 441 Mbps. The higher utilization is possible due to a higher offered load, a total of 600 Mbps of offered UDP packets. The HEADER-COPY is more efficient than the IN-PLACE driver, because copying the entire header into the host and back out is more efficient than accessing the fields independently over the PCI bus. In these cases, CPU load was nearly 100% for all packet sizes, except for the HEADER-COPY driver, where the CPU load dropped to 65% for 8 KB packets, measured by our cpuload tool.

For three pairs of sources and sinks, we modified our experimental technique. For the one and two pair cases, netperf and traffic generated identical results. For three or more sources, netperf could not be used; one source would block and restart later, staggering the streams, defeating the goal of generating three simultaneous sources. Nonetheless, for three or more sources, the standard driver did not increase in throughput - it peaked at 335 Mbps, as before. The IN-PLACE driver peaked at 472 Mbps, and the HEADER-COPY driver at 482 Mbps.

The standard driver, in general, increased bandwidth only 10% using multiple sources, as compared to a single source. For HEADER-COPY, the final design, the measured CPU load was 100% for small packets and multiple sources, and per-packet overheads dominate for small packet sizes. This is also evidenced in the packets/second graph, Figure 6. The packet rate is nearly constant for packets under 1 KB, around 12,000 packets/second, indicating that either interrupt processing or forwarding algorithm processing causes resource contention at the CPU. For larger packets, costs proportional to packet size dominate, suggesting backplane bandwidth contention. Backplane bus arbitration does not appear to be an issue, because the CPU is 100% loaded at low levels, and because the CPU is involved in each step also requiring bus arbitration. By avoiding the additional packet copy, throughputs and packet forwarding rate is increased substantially, up to 45%, to a bandwidth of 480 Mbps.

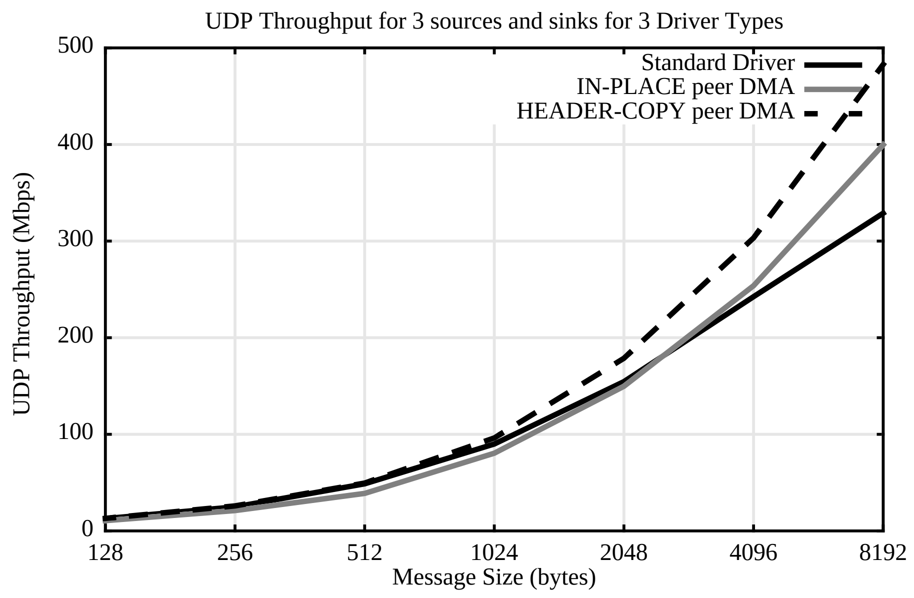

FIGURE 5. UDP bandwidth comparison of three driver types

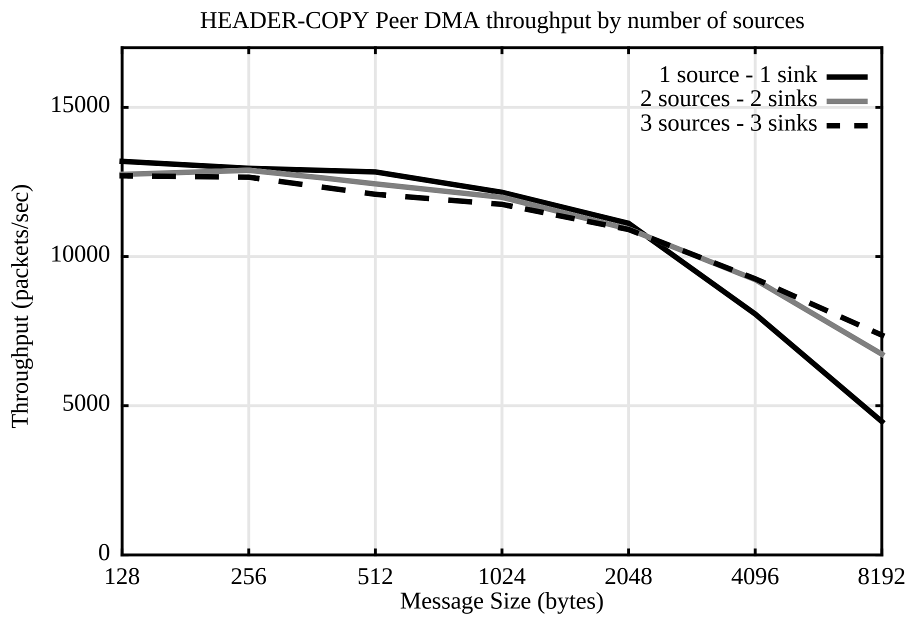

FIGURE 6. HEADER-COPY packet rate throughput

These results focused on UDP packet processing. We expected TCP processing to be identical, but found otherwise. The host-host throughput for a single pair through an intermediate host router is 100 Mbps; lower than a point to point connection which achieves 150 Mbps and does not use an intermediate router. The lower throughput occurs for both conventional and peer-DMA implementations of the host router. There is no current theory on why the rate through a router is lower; it is possible that the hardware flow-control feedback of the Myrinet is affecting the throughput, but our lab currently lacks a comparable link technology capable of supporting TCP over 200 Mbps in which to compare these effects. Using ATM links, the point to point TCP throughput is 100 Mbps, the same as using Myrinet with the host router. The peer-DMA is not currently suspected, because throughput is lowered even for conventional forwarding.

7.0 Implications

The results of these experiments are that HEADER-COPY peer DMA is an effective technique for increasing host-based router throughput, and for reducing the CPU load for forwarding large packets. However, when the same 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 the link bandwidth is sufficient to necessitate seeking a higher performance forwarding algorithm. The results of the UDP tests indicate that link bandwidths in excess of 400 Mbps, closer to 500 Mbps would be required for peer DMA to be beneficial. Below these rates, conventional forwarding is sufficient. This result is host dependent; as CPU and backplane bandwidths increase, conventional forwarding will be faster, requiring ever higher link bandwidths to necessitate peering. However, given the advent of gigabit ethernet, Myrinet, and OC-12c ATM (622 Mbps), link speeds warranting alternate methods are likely to be common.

We assume that NICs support DMA, and that each NIC supporting incoming traffic also contains shared memory sufficient for packet queuing. 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. Note that peer DMA supports any queuing algorithm, because both forwarding and queue management occur in the host CPU.

We also assume that fragmentation is either not required or is trivial. This requires that the maximum transmission units (MTUs) do not vary largely between the NICs, that all NICs can be set to use the smallest, or that packets are forwarded to external nets, where the default Internet MTU is used. The NICs are assumed to deal in units of packets, the same packets as processed by the host forwarding 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, or when data is transcoded, reformatted, segmented and reassembled, authenticated or encrypted. In these cases, the entire packet must be manipulated by the CPU anyway, so the initial data copy of the standard driver aggregates the packet transfer in one step, just as the HEADER-COPY aggregates the individual field accesses of the IN-PLACE driver.

As discussed earlier, the peer DMA techniques reduce peripheral system bandwidth 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. Bus arbitration overhead does not appear to be a significant issue.

Finally, there is the possibility of moving the IP forwarding 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 forwarding algorithm. It also requires moving queue management algorithms into the NIC, and complicates the shared queuing model of existing routers, which may interfere with policy-based forwarding 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.

8.0 Conclusions

Host-based routers provide a flexible, open architecture for router implementation. By using commodity components they support a variety of LAN and WAN links that purpose-built routers do not as rapidly support. Techniques to increase the capacity of host-based routers will remain important, as a result.

We found that peer DMA can increase host-based router throughput by up to 45%, supporting bandwidths of 480 Mbps. Peering is more efficient using a header-copy driver, where the entire link and IP headers are copied in a single DMA operation into the host RAM. For large packet sizes the resulting system also relieves CPU load substantially.

Peer DMA host-based forwarding requires NICs with substantial shared memory resources, because packet queues are stored on the interfaces themselves, rather than in host RAM. The queueing algorithm remains in the host CPU, supporting advanced queue management.

Current systems are packet processing limited; a combination of streamlined forwarding 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.

9.0 References

[1] Boden, N., Cohen, D., *et al*, “Myrinet – A Gigabit-per-Second Local-Area Network,” IEEE-Micro,Vol.15, No.1, February 1995, pp.29-36. <http://www.myri.com/research/publications/Hot.ps>.

[2] 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.

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

[4] 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.

[5] Partridge, C., et al., “A 50-Gb/s IP Router,” IEEE/ACM Transactions on Networking, Vol 6, No 3, June 1998.

[6] Prylli, L. and Tourancheau, B., “Bip: a new protocol designed for high performance networking on myrinet,” Proc. of the PC-NOW Workshop, IPPS/SPDP98, Orlando, USA, 1998.

[7] 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.

[8] Dunning, D., Regnier, G., “Virtual Interface Architecture”, IEEE Proceedings of the Hot Interconnects Symposium V, 1997.

[9] Welsh, M., Basu, A, and von Eicken, T., “ATM and Fast Ethernet Network Interfaces for User-level Communication,” IEEE Proceedings of the Third International Sympsosium on High Performance Computer Architecture, pp. 332-342 Feb 1-5, 1997.