The Impact of a Zero-Scan Internet Checksumming Mechanism

Gregory G. Finn, Steve Hotz, Rod Van Meter
USC/Information Sciences Institute[1]

ABSTRACT

This paper describes a "zero-scan" mechanism that reduces Internet checksumming overhead from a per-byte scan (or copy) cost, to a small and constant per-message cost. Unlike previous techniques, this mechanism requires no message buffering within the source. This will allow Internet transport protocols to achieve transfer latencies comparable to specialized protocols implemented directly on high-speed LAN (link-layer) services. In addition, this mechanism is transparent to systems outside of the source LAN. Hence, this mechanism affords applications the portability of Internet protocols without sacrificing the high performance of specialized LAN transport protocols.

The proposed zero-scan checksumming scheme eliminates the last requirement for an additional data copy/scan, beyond the scan required to transmit or receive from the network channel. If this checksumming mechanism is combined with zero-copy operating system mechanisms that provide low-overhead transfer across application and kernel boundaries, a network interface architecture that provides separate message buffering is no longer required. A consequence is that the network interface may be reduced, essentially, to DMA engines plus link- and physical-layer logic. Taken one step further, the network interface could be integrated with the CPU to create an "internet microprocessor". These alternative interface designs are discussed, along with their requirements and effects upon operating system and computer system architectures.

1. Introduction and Motivation

Message latency is a significant obstacle for distributed computing. It limits the minimum time required to execute a remote procedure call (RPC), and so limits the practical range for distributed applications. Historically, message latency has been dominated within local and campus areas by transmission (injection) latency and latency due to message processing overhead. This is no longer true.

Transmission latency has been reduced by a factor of 10,000 over the last quarter century due to the marked rise in capacity. When the ARPANET began in 1969, it provided point-to-point links that ranged up to approximately 100 Kbit/s of data bandwidth per port. Commercial LANs now provide port bandwidths well above 1 Gbit/s. Examples are the Myrinet [1], which was based upon ATOMIC [2], and HiPPI [3]. Message processing latency has likewise decreased by a factor of 100, as processor speeds have risen from the 1 MIP mainframe that was a typical host in 1969 to the 100 MIPs microprocessors that are now available. Figure 1 illustrates (and compares) the rise in capacity and processing speeds.

With transmission and processing latencies removed as key concerns, other sources of overhead are relatively more responsible for message latency. This work focuses on latency resulting from per-byte message copies (and scans) within end systems. As application portability and interoperability are desirable, we address ourselves to internetworking protocol stacks; specifically the TCP/IPv4 suite.[2]

1. This research was sponsored by the Advanced Research Projects Agency under Contract No. DABT63-93-C-0062. Views and conclusions contained in this report are the authors' and should not be interpreted as representing the official opinion or policies, either expressed or implied, of ARPA, the U.S. Government, or any person or agency connected with them.

2. This paper uses TCP/IPv4 as its primary example and focus, and the term "Internet" is used for specific illustrations. However, the proposed mechanisms and discussion are more generally applicable to other internetworking suites (e.g. TCP/IPv6 or ISO/OSI), thus the term "internet" is also used throughout.

1.1 Zero-Pass/Cut-Through Systems

Ideally, a message should move directly between the application memory and network channels, incurring exactly one per-byte scan which is required to transmit/receive the packet onto/from the network. We refer to such a system as a “zero-pass” system, or as providing “cut-through” network transfer. In practice, several per-byte operations are commonly found in existing systems. Each per-byte scan (or copy) operation absorbs memory bandwidth and adds latency.

Considerable work has been done to analyze, and to provide mechanisms to eliminate, sources of per-byte overhead. In the past, one-copy and zero-copy techniques have focused on aspects of interface design that allow messages to move more directly between application memory and message buffers in the network interface [4][5][6][7], and on operating system and memory management issues that allow messages to cross protection domains without requiring a data copy [5][8][9][10][11]. However, an actual zero-pass system does not exist for transport-layer Internet messages.

A significant source of message latency remains: an embedded transport-layer checksum induces a per-byte message latency that is not addressed by previous techniques. For Internet protocols, to date, that checksum function either requires a separate per-byte computation, or it is combined with another non-essential per-byte operation. In either case, at least one *additional* per-byte operation is required for the checksum, and thus it is impossible to fully exploit zero-pass system mechanisms.

1.2 This Work

The overhead associated with end-to-end checksumming can be the principal contributor to message latency across a gigabit LAN. It can more than double the message latency seen by receivers on local-area networks (LANs), which has a serious, negative effect on distributed computing.

We present a mechanism that effectively eliminates checksum-induced latency, and thereby removes an obstacle that currently exaggerates the performance differences between internet and LAN protocols. Those performance differences encourage the use of LAN protocol stacks by distributed applications. That practice impedes application portability. The checksumming mechanism described here can improve the performance of existing distributed internet applications, and more importantly, would encourage a broader use of internet protocols for distributed computing, consequently improving application portability. Lastly, the mechanism is an enabling technology for high-performance network interface designs.

This paper discusses Internet checksum latency problems and current solutions in Section 2. Our zero-scan mechanism that reduces Internet checksumming overhead to a small and constant per-message cost is described in Section 3. Section 4 considers the implications of cut-through interface designs on computer and operating system architecture. An alternative network interface design made possible by zero-scan checksumming is introduced in Section 5.

	Instructions Per Bit
1 MIP@100 Kb/s	10
100 MIPs@1 Gb/s	0.1

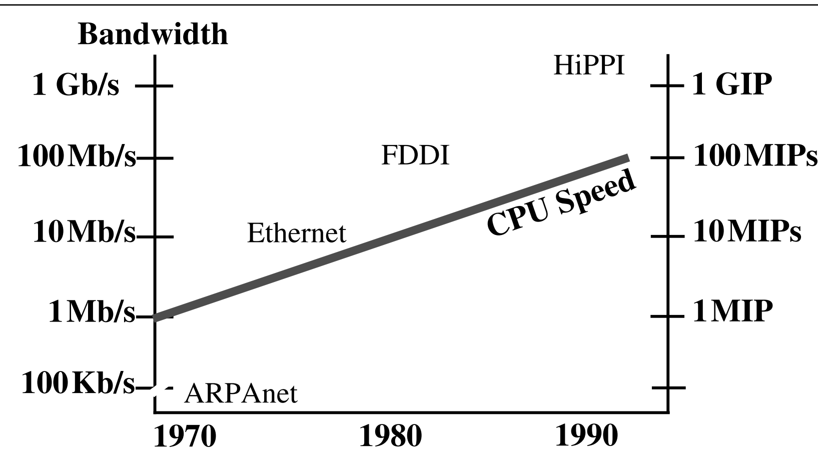

Figure 1. Bandwidth Rise

2. Checksum-Induced Latency in Internet Protocols

The Internet protocol stack relies on transport-layer checksumming, in both TCP and UDP, to ensure end-to-end data fidelity [12][13][14]. Since these checksum fields are embedded within the transport-layer message headers, calculation and insertion of the checksum prior to transmission *requires* an additional scan of the payload by the source. This is evidenced by current approaches to checksum calculation.

The naive approach to checksumming requires the host to perform a software scan of both incoming and outgoing transport-layer payloads to calculate and insert, or to validate, the checksum. This results in a per-byte latency cost, regardless of any other high-performance techniques that may be used. To alleviate this cost, a designer may choose to include internet checksumming support in the network interface and calculate the checksum as a message is moved between system memory and buffers in the network interface [15]. This avoids a software checksum scan by the host, but now the message *must* be stored inside the interface so the checksum can be embedded before the packet (i.e. the header) is transmitted. This “improved design” requires a copy into a message buffer that was not necessarily needed using the naive “software scan” approach.

These checksumming mechanisms require an additional scan (or copy) of the transport packet, and incur the associated transfer latency. While other techniques (see Section 1.1) eliminate other causes of latency in computing systems, the requirements imposed by the embedded checksum undercuts these efforts. For example, interface designs that allow messages to “cut-through” between application memory and the network channels [16][19] do not copy them into message buffers in the interface, hence a software scan must be performed.

A need exists for an embedded checksum mechanism that neither requires that the source externally buffer a message nor that it calculate the checksum in software. With such a mechanism, performance differences between internet protocols and specialized LAN protocols [17][18] could be significantly reduced.

2.1 Scale of Checksum-Induced Latency

The argument has been advanced that in the future the bandwidth of the CPU/memory data path is likely to remain within the same order of magnitude as network bandwidth [20]. If the source memory bus can deliver data at a rate equal to the bandwidth of a LAN channel, copying a message into buffers in the interface effectively doubles the minimum message latency across that LAN. This is approximately the situation that exists now when workstations are attached to gigabit networks.

The effect that checksum-induced buffering latency has on an existing cut-through interface can be gauged by determining how long it takes to calculate the checksum, embed it, and transmit a message, as opposed to directly transmitting that message. For a Myrinet SBus interface attached to a SPARCstation-20/71, the time required to copy a 1024-byte message into a buffer in the interface while calculating the checksum is at least 22 µs, whereas the time required to transmit that message into a Myrinet channel is 13 µs. See Figure 2.

Buffering latency for a 1024 byte packet sent across a 0.5 Km LAN. Propagation is 0.6*c*. Times are in µs.

Copy	Xmit	Prop	Total
22	13	3	38
--	13	3	16

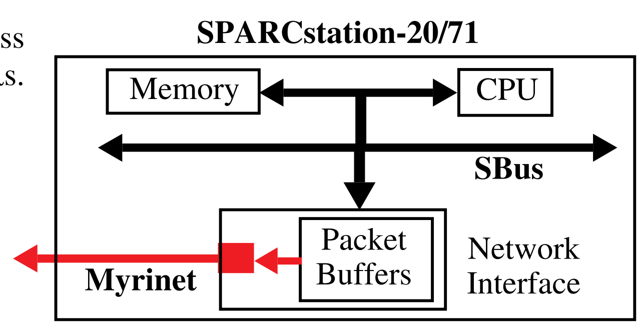

Figure 2. Checksum-Induced Transfer Latency

The effect that reduced transfer latency has upon total end-to-end message latency is progressively less important as end-to-end propagation time rises. The removal of checksum-induced latency reduces transfer latency on the order of 50% to the first gateway on the source LAN, however, the queueing delays experienced as a message passes through successive store-and-forward gateways significantly increase message propagation time. The effect of eliminating the checksum scan will be most pronounced within a LAN, producing noticeable but less important performance improvement for applications communicating across one or two gateways.

2.2 Prior Art

The latency due to calculation of an embedded internet checksum can be avoided by using a network- or transport-layer trailer [22]. A protocol that employs this scheme repositions the checksum field at the end of the checksummed payload so that the checksum calculation can occur as the packet is transmitted. The checksum can then be placed into the correct position without buffering the packet. However, this scheme is specific to each particular transport (or network) protocol. Moreover, it is not used by TCP or UDP, and is not included in new Internet protocol designs (e.g. TCP under IPv6 [23]).

A related approach manipulates transport-layer packets that use an embedded checksum into a packet form that allows a checksum field to be placed in a trailing position [21]. This scheme clears space at the end of a packet by performing a low-cost swap of the trailing payload bytes into the embedded checksum field. However, this mechanism is difficult to deploy incrementally, as it is not transparent outside of the source network. Hence, it requires "dual stacking" and end-to-end negotiation for its use.

Another approach that avoids the checksum scan simply does not use internet protocols in high-performance distributed applications. This restricts this class of computing to a single link-level network, and is thus undesirable. A distributed application should be able to transparently exploit resources on different portions of an organizational intranet, even though it us unlikely that such an application would be run over a 20-hop transcontinental internet connection.

A less restrictive approach uses link-layer transport protocols within the source network, but internet protocols to destinations outside the network. This type of "dual stacking" requires higher-level demultiplexing, and generally interferes with either application portability or protocol interoperability.

None of the previous work offers a transparent solution that can avoid the per-byte operation and can maintain the interoperability afforded by current internetworking protocols. Current approaches to checksum calculation require an additional data scan that can double communication latency within a local network.

3. Zero-Pass Checksumming

Zero-pass checksumming [24] is a mechanism that allows messages that contain embedded checksums to be sent and received without incurring a checksum-induced per-byte latency.[3] This approach is based on the idea that the checksum need not be embedded within the header at the source system. By relaxing this requirement, the per-byte checksum calculation can be combined with another per-byte scan; namely with the transmission or reception of the packet. By exploiting a per-byte scan that cannot be avoided for message communication, this approach affords the possibility of an absolute minimum number of distinct data scans.

There are sender-calculated and receiver-calculated versions of zero-pass checksumming. In both cases, the source transmits the message without prior embedding of the checksum.

- In the sender-calculated version, the checksum calculation takes place as the message is transmitted, whereupon it is appended as a link-level trailer.
- In the receiver-calculated version, the checksum calculation occurs as the message is received by the first-hop node along the path to the final destination. This version requires that the

3. A patent is pending on this mechanism.

source network can guarantee sufficient data fidelity to protect the packet between the source interface and 1st-hop receiver interface.[4]

In both versions, embedding of the checksum does not take place at the source. Embedding is postponed until the message reaches the next node with a receiving interface attached to the local network, whereupon the checksum value is available, either in a link-layer trailer appended to the message or in a register in the receiving interface. This mechanism relies upon the observation that the node that first receives an internet message is typically either a local destination or a gateway. In either case, the internet checksum can be embedded by this node.[5]

The remainder of this section presents an illustrative design of zero-pass checksumming, and discusses issues of transparency to external hosts and interoperability with current network interfaces.

3.1 Implementation Issues

This mechanism may be considered a part of the link-layer protocol or of an integrated checksumming mini-layer that resides above the link layer but below the internet layer. Since the checksum is embedded by the first interface that receives the message, the transport-layer internet message is correctly formatted when it is received by the higher layers.

An example of how one might embed an internet checksumming layer within a link-layer packet is illustrated in Figure 3. Additional fields allow the receiver to efficiently decide if a checksum should be embedded, and at what location. The *Offset* field points to the embedded checksum field. *Length* determines the number of payload bytes that contribute to the checksum. *Bias* contains a value that initializes the checksum accumulation register. Bias can contribute the checksum value of the UDP or TCP pseudo-header. The *I* field indicates whether or not the checksum value has been inserted into the checksum field.

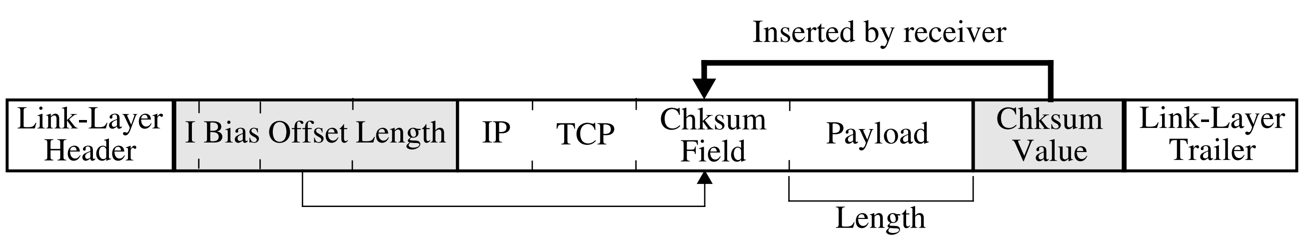

Figure 3. Checksumming Layer

The values of Bias, Offset and Length are known to the source when a message is transmitted. Within the source network, they allow interface hardware to calculate the correct checksum value when the message is transmitted or received. In Figure 3, the TCP checksum value would be calculated when the message is transmitted. Extracting that checksum value from the trailer and placing it into the checksum field when received would incur a small and constant cost per message.

The checksum calculation can be performed by specialized hardware located in the network channel interface. Advances in logic speed and organization now allow the Internet checksum to be calculated at rates well above a gigabit/second using either custom logic or inexpensive off-the-shelf programmable logic parts [25].

4. While the checksum is an end-to-end mechanism, current systems that calculate the checksum during the DMA transfer between host and interface memory must trust the data fidelity provided by the interface, its memory and DMA logic. Here that trust boundary is extended further, to include the transfer across the local source network.

5. Embedding of the checksum requires that the message be stored. This would not occur, for example, within some types of bridges or "cut-through" IP switches.

3.2 Transparency

Since it is impractical to require that all internet protocol stacks be changed, the use of zero-pass checksumming must be transparent to protocol software and applications outside the source network where it is used. This is accomplished by requiring that any host on the source network, that receives a message that was sent using zero-pass checksumming, embed the checksum value into the checksum field. For IP, this constitutes the final step in creating a conformant Internet transport message.

If the destination lies within the source network, the conformant Internet message is passed upwards toward the application. If the destination lies outside the source network, the IP gateway will forward the conformant message. As a consequence of transparency, hosts outside the source network obtain the benefit of reduced message latency without any change to either their network interface hardware or protocol software.[6]

The I flag is used to identify internet messages that are conformant. For internet messages arriving from outside the source network, the gateway would set the I flag in the checksumming wrapper to indicate that the message does not require checksum insertion.

3.3 Interoperability with Existing Network Interfaces

Zero-pass checksumming would be most cleanly integrated by including it in every host attached directly to a local network. However, it can also be incorporated into existing networks. One way of achieving this would be to encapsulate IP messages using two different link-layer protocols: one to carry standard IP and one to assist zero-pass checksumming. The use of the new protocol would cue interfaces and driver software. At this point, there are two possible approaches.

The first approach depends on the sender-calculated variation of zero-pass checksumming which allows hosts that implement zero-pass to interoperate more freely with those that do not. Because the sending interface is responsible for calculating (and appending) the checksum, the receiver does not depend on hardware assistance for performance. Hosts that have interfaces that do not implement zero-pass checksumming can have their network interface driver reprogrammed to recognize the new IP encapsulation. This will allow them to achieve the latency reduction benefit when receiving messages from those that do implement zero-pass checksumming.

Alternatively, if hosts that do not implement zero-pass checksumming are not programmed to recognize the new IP encapsulation, they presumably discard any such messages that are received. To allow interoperation in this case, hosts that implement zero-pass checksumming must recognize[7] (and use standard IP encapsulation when sending to) hosts that do not implement zero-pass checksumming. Because this approach makes a clear distinction between capable and non-capable hosts, either the sender-calculated or the receiver-calculated version of zero-pass checksumming could be deployed.

Either approach will allow interoperability with existing network interfaces, thus the deployment of zero-pass checksumming can be transparent to non-capable hosts within the local network as well as to all external hosts. Consequently, zero-pass checksumming is a mechanism that can be used to reduce communication latency without sacrificing the portability offered by standard internetworking protocols. This mechanism may also facilitate new network interface and system designs that were not otherwise practical. The remainder of this paper presents ideas for future work and discusses the basic system issues that are raised.

---

6. Better performance would result if ACK and/or reply packets could also be sent using zero-pass.

7. The information to distinguish capable and non-capable hosts could be acquired along with, or immediately after, address resolution (i.e. ARP). If this is not supported in the address resolution protocol explicitly, the driver software could probe using an ICMP-ECHO encapsulated with the new zero-pass link-layer packet.

4. Cut-Through Internet Interface Design Issues

A cut-through internet interface moves internetwork packets directly between system memory and the network channels, incurring essentially no per-byte overhead. Such an interface is now possible, as the checksum-related buffering (or scan) of messages is eliminated by the zero-pass checksumming mechanism.

Message overhead, specifically latency, can be minimized by combining a cut-through internet interface with other mechanisms (e.g. zero-copy operating systems and protocol stacks). Such mechanisms are required, as cut-through internet capability has little practical interest if the path between application memory and network channel requires other per-byte operations.[8] The details of these mechanisms are addressed in a large body of related work (see Section 1.1), and are beyond the scope of this paper.

The remainder of this section discusses other requirements a cut-through interface places on the network interface and system architecture. It focuses on the system components a data packet may traverse between the network channel and a buffer provided by the operating system. This includes components that may provide interface buffering (Section 4.1), movement into host memory (Section 4.2), and cache interaction (Section 4.3). Ideally, these components will cause minimal interference to a cut-through design. Although the following discussion focuses on message transmission, the issues are much the same for incoming message processing.

4.1 Interface Buffering

The performance of a cut-through interface requires that a message not be copied from main memory into a buffer in interface memory. There are two strategies for interface designs that avoid this overhead.

- A network interface can include on-board buffer memory where packets reside prior to transmission. Cut-through performance would require that the buffer memory be directly writable by the application, so that outgoing messages can be built therein and subsequently transmitted directly onto the network channel.
- A network interface without separate buffering can be built, where message data is retrieved from system memory as required by the channel when transmitting. Such an interface would employ a direct memory access (DMA) engine that feeds a transmission FIFO.

The on-board buffer memory used in the first approach is generally restricted in size, which limits the number of applications that can take advantage of it simultaneously. It may also require more specialized support from the operating system (e.g. specific I/O page remapping).

As the second approach does not face the limitations imposed by on-board buffers, the remainder of this paper assumes the second design. However, to achieve the desired latency reduction the designer must ensure that, on average, the FIFO does not empty during transmission and that its depth is small relative to message size. In general, the delivered bandwidth of the system memory bus must be substantially greater than network channel bandwidth.

To keep the FIFO from emptying, the network interface can be given highest priority access to system memory when transmitting. That is the approach taken by Mosaic [26], where the CPU has lower priority than either the network input or output channel DMA engines. This requirement can be relaxed if the network channel is asynchronous.[9]

8. If other per-byte scan operations are required, the checksum calculation could be performed concurrently and the zero-pass checksumming technique would be less critical.
9. Underrun onto a synchronous channel can be prevented if the physical layer provides a “no-data” symbol that can be inserted into a packet frame when the network interface has no data to present when the transmit clock ticks.

4.2 DMA -vs.- PIO

Message data can be moved between the network channels and system memory by either programmed input/output (PIO) or by DMA. PIO is more flexible, and in some systems PIO is effectively faster than DMA. However, the reason often mentioned for the effectiveness of PIO is that it allows checksum calculation to be combined with data copying at no additional cost in RISC architectures [9]; this is no longer an advantage with a zero-pass checksum. Without this advantage, there is little reason to commit processor cycles to transfer internet messages.

The use of DMA to move data to/from network channels will allow the CPU to perform other tasks. However, DMA engines access system memory sequentially. As this is inherently less flexible than PIO, additional DMA features would be useful to facilitate network message processing: i.e. a scatter/gather capability and the ability to adjust for data alignment.

4.2.1 Need for Scatter/Gather DMA

To support cut-through transfer for a broad range of applications, the transmit DMA engine should have the capability to assemble a complete message from component pieces. This allows the outgoing message header and payload to be located in separate areas of memory and so supports various strategies for implementing layered protocols, e.g. stenciling. Headers (and/or portions of a payload) that remain largely static for the duration of a connection can be associated with the connection and modified for reuse, rather than generated from scratch for each message. Scattering of incoming messages into multiple system buffers is also a feature deemed important to reduce complexity [9].

4.2.2 Need for Realignment

Headers can be of variable length, thus it is unrealistic to impose a restriction that all pieces of a message be both aligned and be multiples of the basic wordsize of a particular processor. To send a valid message and payload, the output-channel DMA controller must deal with misalignment. This can be accomplished with a shifter and an alignment FIFO.

4.3 Cache Memory Interaction

One of the critical issues of system design is whether to interface I/O directly to the data cache or to system memory. Since the network is emerging as the mechanism that carries the bulk of I/O data in and out of workstations, the same question arises: Should the network interface be directly attached to the data cache or to system memory? To answer that question, we examine the results of studies on I/O channels that were directly attached to cache and to studies of cache efficiency when dealing with internetwork data streams.

4.3.1 Directly Attached I/O Channels

As competition for a limited cache memory rises, the effectiveness of the cache will be reduced. Previous studies of I/O channels that are directly connected to the cache [28] suggest that the cache miss rate is little affected as long as the ratio of the rate of cache references made by I/O channels to that made by the CPU remains below 0.05. Above that rate the miss ratio rises significantly.

For a RISC processor, the percentage of memory references that are loads and stores is approximately 25% [27]. A high-speed processor may produce data cache references at a rate of 50 million accesses/second. An active gigabit channel generates 64-bit references at a rate of 15.6 million/second. This produces access ratios of 0.31 or 0.62 when one or both channels are active respectively. These ratios are far above 0.05, and suggest that a decision to directly connect a high-speed network channel to a data cache should be approached cautiously.[10]

10. While this first-order analysis does not take into account the burstiness of network traffic, the results cited in Section 4.3.2 are for actual network traffic.

The reduction in data cache efficiency due to I/O interference could be eliminated by providing the network interface with its own specialized cache. This specialized data cache would be loaded only by the network interface, but could be read by the CPU. Care must be taken to avoid inconsistency. The size, organization, miss and write strategies of a this specialized cache need not be the same as for the CPU data cache. The effectiveness of such a technique is a subject for future study.

4.3.2 Data Cache Efficiency for Internetwork Applications

The cache hit ratios seen for incoming message data has been found to be poor under UNIX, Mach and HPUX operating systems [20][29]. The data cache in these studies was 256 KB, virtually indexed and physically tagged. To determine the effectiveness of a large data cache in a zero-copy system, the cache residency of message data was determined, after the kernel has loaded the cache by copying messages into buffers, at the point where the operating system passes control to the application process. Even when the effect of virtual memory translation misses was eliminated, an average hit ratio of only 28% was observed.

4.3.3 Summary

Although there are additional studies to be made, it is unlikely that passing network traffic through a large CPU data cache will significantly reduce memory bus traffic and enhance performance of network applications. Since attaching a high-speed network interface directly to the CPU data cache causes interference when the network channels are active, there seems little reason to do it.

The consequence of not passing network traffic through the CPU data cache is that consistency must be maintained otherwise. This can be achieved by bus snooping or (less efficiently) by software invalidation. It can also be achieved by requiring that messages be stored in pages of system memory that are non-cacheable.

5. Internet Microprocessor

A cut-through network interface *without on-board memory* can be exploited by internet protocols using zero-pass checksumming.With the elimination of buffer memory, which usually comprises a significant portion of interface hardware, it may be practical to incorporate a network interface directly into a microprocessor package. This may afford (1) reduced system size and cost, and (2) better performance through a tighter coupling of and faster arbitration between CPU and network processing.

This section examines the feasibility and the basic issues of incorporating an internet interface into a microprocessor. Section 5.1 presents a possible design for an internet microprocessor that considers the cut-through interface issues discussed in Section 4. A primary issue for such a device would be to allow connection with heterogeneous network technologies; this is discussed in Section 5.2.

5.1 Feasibility Concerns

The logic area devoted to a cut-through network interface, including DMA support, is very small in comparison to the area devoted to CPU and on-chip memory. This was demonstrated by the Mosaic chip [26]. The additional logic needed to generalize a LAN-specific interface to support Internet protocols includes the logic to calculate the Internet checksum and append the value to outgoing messages. This additional amount of logic should not be an obstacle. Likewise, with the advent of denser packaging technologies, such as ball grid arrays, the additional signals required for the network channels should not impose an obstacle. For example, a 2.4 Gb/s Myrinet system-area network (SAN) port requires twenty signal lines.

An example architecture[11] for an Internet microprocessor is depicted in Figure 4. There is no provision for incoming or outgoing on-chip message buffering. Message buffers reside in system memory, and are DMA'd between system memory and the network channels. Small FIFOs would usually reside within the network physical-layer logic.

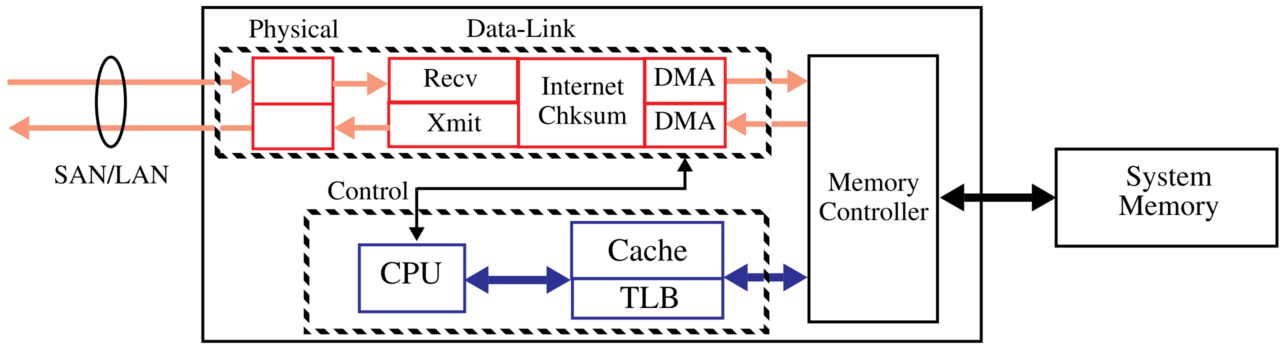

Figure 4. Internet Microprocessor

Studies cited earlier (see Section 4.3) suggest that there is little advantage to be gained from interaction between the network data streams and the data cache; this is reflected in Figure 4. In this case, the suggested mechanism to maintain cache coherence would be for all network I/O to pass in/out of pages that are pinned and marked non-cacheable. This requires support for non-cached pages in the microprocessor and cache subsystems. The PowerPC [30] is an example of a microprocessor family that provides this support.

The sum of the expected demands made by the CPU/cache and network channels dictate the minimum transfer bandwidth the system memory and memory bus should support. The memory controller must arbitrate between microprocessor or cache requests and channel requests for system memory. Since message data must cut-through between the network channels and system memory, network channels cannot be denied access to memory without risking overruns or underruns.[12] If the demand for memory cycles made by the network channels is too great, the CPU/cache may be starved. To avoid the excessive demand presented by sustained simultaneous input and output, output inhibit [16] can be used to prevent transmission of new messages during message arrival (see also Section 4.1).

5.2 Interfacing to Multiple Networks

It is unlikely that any particular networking technology will become the only one in use. A consequence of this is that a microprocessor that incorporates a network interface must be able to attach to different physical networks. This presents an obstacle. Producing a microprocessor chip in versions that correspond to each network technology in use is likely to be unattractive.

One strategy to deal with this obstacle would be to produce a small number of variations for those high-performance networks most likely to be encountered, such as the proposed 'Gigabit Ethernet'. Other network interfaces would then be attached as peripheral devices conventionally, on the memory or I/O buses.

Another strategy would be to create a standard interface between the microprocessor chip and a prototypical network interface. The desired specific network interface would then be incorporated into an MCM package with the microprocessor. This is suggested by the organization in Figure 4.

11. Alternatively, an extremely simple network interface design would substitute PIO in place of the DMA engines and checksumming hardware of Figure 4. This becomes viable as CPU and PIO speeds increase. This makes the CPU busy and therefore unable to perform other tasks when the network channels are active.

12. A cut-through interface may include a small FIFO but it should not normally buffer an entire message.

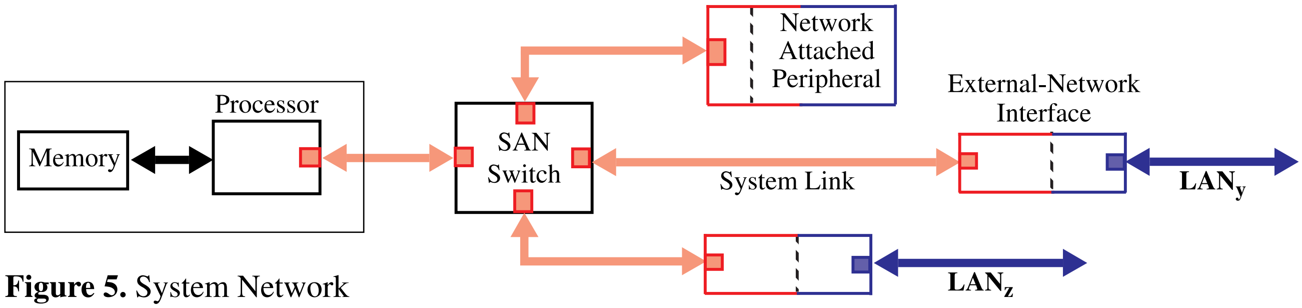

Figure 5. System Network

A third strategy would substitute a standard SAN in place of the standard system bus, as shown in Figure 5. The microprocessor, external-network interfaces and peripheral devices would be nodes on that SAN. In this approach, one must be careful not to reintroduce unnecessary buffering delay in the peripherals, and especially not in the external network interfaces. A SAN used in such a capacity should have characteristics that allow a straightforward, unbuffered conversion between SAN and LAN packet formats.

To support cut-through from a source node onto an external network, the external network interface must know that a message awaits transfer and the source must know when the message can be cut through onto the external network channel. If the system network is asynchronous, this Request/Grant feedback signalling can be provided in-band as part of the channel flow control that passes through a switch.

6. Summary

Calculation and insertion of an embedded internet checksum by the source requires a per-byte scan of a transport message before transmission. This extra pass across the message adds latency between the presentation of a message to the transport layer by the application and the time that message begins transmission on the network channel. Across high-performance networks this results in an effective doubling of minimum message latency.

The proposed zero-pass checksumming mechanism eliminates the need for an extra scan across a message before transmission, by postponing the embedding of the internet checksum until a message reaches the next store-and-forward node. By removing this critical obstacle, this mechanism allows internet transport protocols to achieve transfer latencies comparable to specialized protocols implemented directly on high-speed LAN (link-layer) services. Because this mechanism is transparent to systems outside of the source LAN, it affords applications the portability of internet protocols without sacrificing the high performance of specialized LAN transport protocols.

This mechanism may also serve as an enabling technology for novel high-performance network interface designs. Because this mechanism requires no message buffering within the source, it allows computing systems to take full advantage of zero-copy operating system mechanisms and cut-through network interfaces. Essentially, the network interface may be reduced to DMA engines plus link- and physical-layer logic. If providing high-performance internet message processing becomes an important factor in the design of future microprocessors, the ability to effectively utilize a cut-through interface will allow the development of “internet microprocessors”.

7. Acknowledgments

The authors would like to thank Mike Carlton, Ted Faber, Jon Postel, and Craig Milo Rogers for their constructive comments on previous versions of this paper. In addition, Craig Milo Rogers provided key insights and collaborated in the development of the zero-pass checksumming scheme.

8. References

[1] Boden, N., Cohen, D., Felderman, R., Kulawik, A. E., Seitz, C. L., Seizovic, J. N., Wen-King Su
Myrinet: A Gigabit-per-Second Local Area Network
IEEE Micro, Vol. 15, No. 1, February 1995.

[2] Cohen, D., Finn, G. G., Felderman, R., DeSchon, A.
The Use of Message-Based Multicomputer Components to Construct Gigabit Networks
ACM Computer Communication Review, Vol. 23, No.3., July 1993.

[3] HIgh Performance Parallel Interface
Mechanical, Electrical, and Signalling Protocol Specification
ANSI X3T9/88-127, X3T9.3/88-023, REV 8.1, June 1991.

[4] Dalton, C., Watson, G., Banks, D., Calamvokis, C., Edwards, A., Lumley, J.
Afterburner
IEEE Network, July 1993.

[5] Smith, J. M.,Traw, C. B. S.
Giving Applications Access to Gb/s Networking
IEEE Network, July 1993.

[6] Buzzard, G. jacobson, D., Marovich, S., Wilkes, J.
Hamlyn: a high-performance network interface with sender-based memory management
Hot Interconnects '95.

[7] Steenkiste, P. A.
Gigabit Nectar: Architecture and Performance
Carnegie Mellon University, CMU-CS-95-192, September 1995.

[8] Druschel, P., Peterson, L. L.
Fbufs: A High-Bandwidth Cross-Domain Transfer Facility
Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles, Dec. 1993.

[9] Druschel, P.
Operating System Support for High-Speed Networking
TR 94-24, Dept. of Computer Science, University of Arizona, August 1994.

[10] Tzou, S., Anderson, D. P.
The Performance of Message-passing using Restricted Virtual Memory Remapping
Software - Practice and Experience, Vol. 21(3), pp. 251-267, March 1991.

[11] Cheriton, D.R.
The V Distributed System
Communications of the ACM, 31(3):314-333, March 1988.

[12] Plummer, W
IEN 45: TCP Checksum Function Design
Bolt Beranek and Newman, Inc., June 1978

[13] Postel, J.
RFC 768: User Datagram Protocol
USC/Information Sciences Institute, August 1980.

[14] Postel, J. (editor)
RFC 793: Transmission Control Protocol
USC/Information Sciences Institute, September 1981.

[15] Steenkiste, P. A.
A Systematic Approach to Host Interface Design for High-Speed Networks
IEEE Computer, March 1994.

[16] Davie, B. S.
The Architecture and Implementation of a High-Speed Host Interface
IEEE Journal on Selected Areas in Communications, Vol. 11, No. 2, February 1993.

[17] Pakin, S., Lauria, M., Chien, A.
High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet
Proceedings of the 1995 ACM/IEEE Supercomputing Conference, San Diego, CA, 1995.

[18] Black, D., Smith, R., Sears, S., Dean,Randall
FLIPC: A Low Latency Messaging System for Distributed Real Time Environments
Proceedings of the 1996 USENIX Technical Conference, San Diego, CA, January 1996.

[19] Traw, C. B. S., Smith, J. M.
Hardware/Software Organization of a High-Performance ATM Host Interface
IEEE Journal on Selected Areas in Communications, Vol. 11, No. 2, February 1993.

[20] Druschel, P., Abbot, M. B., Pagels, M. A., Peterson, L. L.
Network Subsystem Design
IEEE Network, July 1993.

[21] Bridges, M., Subramaniam, S., Edwards, D.
Internet Draft, December 1994
TCP Embedded Trailer Checksum
Hewlett-Packard Corp.

[22] Cheriton, D. R.
RFC 1045 VMTP: A Versatile Message Transaction Protocol
February 1988.

[23] Deering, S., Hinden, R.
RFC 1883: Internet Protocol, Version 6 (IPv6) Specification
April 1996.

[24] Finn, G. G., Hotz, S., Rogers, C. M.
A Zero-Pass End-to-End Checksum Mechanism for IPv6
USC/Information Sciences Institute, March 1995.

[25] Touch, J., Parham, B.
RFC 1936: Implementing the Internet Checksum in Hardware
USC/Information Sciences Institute, December 1995.

[26] Seitz, C., Wen-King Su
A Family of Routing and Communication Chips Based on the Mosaic
Proceedings of the Washington Symposium On Integrated Systems, Seattle, WA., 1993.

[27] Patterson, A. D., Hennessy, J. L.
Computer Architecture A Quantitative Approach, Second Edition, Ch. 5
Morgan Kaufman, 1996.

[28] Smith, A. J.
Cache Memories
Computing Surveys, Vol. 14, No. 3, September 1982.

[29] Pagels, M. A., Druschel, P., Peterson, L. L.
Cache and TLB Effectiveness in the Processing of Network Data
TR 93 04, Dept. of Computer Science, University of Arizona, March 1994.

[30] *PowerPC Microprocessor Family: The Programming Environments*
Motorola Inc., 1994.