



Intel[®] Technology Journal

Communications Processing

Addressing TCP/IP Processing Challenges using the IA and IXP Processors

Addressing TCP/IP Processing Challenges Using the IA and IXP Processors

Dave Minturn, Corporate Technology Group, Intel Corporation
Greg Regnier, Corporate Technology Group, Intel Corporation
Jon Krueger, Technology and Manufacturing Group, Intel Corporation
Ravishankar Iyer, Corporate Technology Group, Intel Corporation
Srihari Makineni, Corporate Technology Group, Intel Corporation

Index words: TCP/IP, network processing, 10 Gbps, acceleration, IXP, IA, IPF, ETA

ABSTRACT

The majority of datacenter applications such as web services, e-commerce, storage, and firewall use Transmission Control Protocol/Internet Protocol (TCP/IP) as the data communication protocol of choice. As such, the performance of these applications is largely dependent upon the efficient processing of TCP/IP packets. In addition, with the arrival of the 10 Gigabit Ethernet, the TCP/IP packet processing needs to become faster and more efficient to let the applications benefit from higher bandwidths. Motivated by this, the material presented in this paper focuses on (a) network bandwidth and the associated TCP/IP processing requirements of typical datacenter applications, (b) challenges of scaling TCP/IP packet processing to 10 Gigabit speeds, and (c) exploring to what extent these challenges can be addressed by the Intel[®] IA32 and IXP network processors. We also discuss potential techniques that further help TCP/IP processing.

INTRODUCTION

TCP/IP over Ethernet is the most dominant packet processing protocol in datacenters and on the Internet. It is used by both server and network infrastructure applications (e.g., firewall, proxy server, etc.). We show that the TCP/IP component of these applications constitutes a significant amount of the overall processing and hence it is important to understand its characteristics and make sure that it can scale well up to 10 Gbps and beyond. Analysis done on TCP/IP in the past [1] has

shown that only a small fraction of the computing cycles are required by the actual TCP/IP processing and that the majority of cycles are spent in dealing with the Operating System (OS), managing the buffers, and passing the data back and forth between the stack and the end-user applications. Many improvements and optimizations have been done to speed up the TCP/IP packet processing over the years. CPU-intensive functions such as checksum calculation and segmentation of large chunks of data into right-sized packets have been offloaded to the Network Interface Card (NIC). Interrupt coalescing by the NIC devices to minimize the number of interrupts sent to the CPU is also reducing the burden on processors. In addition to these NIC features, some OS advancements such as asynchronous IO and pre-posted buffers [12] are also helping to speed up TCP/IP packet processing. While these optimizations, combined with the increases in CPU processing power, are sufficient for processing packets at 100 Mbps and 1 Gbps Ethernet speeds, these are clearly not sufficient for 10 Gbps speeds. Upcoming applications like storage over IP, web services, and ubiquitous broadband connectivity to homes enabling video-on-demand applications require multi-Gbps speeds that require TCP/IP packet processing to scale to these speeds with the least amount of computing power. [Figure 1](#) shows the transmit and receive-side throughput and CPU utilization data for the Microsoft Windows^{*} 2003 Enterprise Edition^{*} TCP/IP stack running on an Intel[®] Xeon[™] processor (2.6 GHz) with

[®] Intel is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

^{*} Other brands and names are the property of their respective owners.

[™] Xeon is a trademark of Intel Corporation or its subsidiaries in the United States and other countries.

Hyper-Threading Technology. These measurements were taken using NTtcp, a variant of the tcp micro benchmark [11] optimized for the Windows OS.

This paper is organized as follows. We perform an in-depth measurement and evaluation of packet processing on Intel’s state-of-the-art server and network processors. The primary focus of this study is to (a) quantify the computational requirements of the TCP/IP packet processing component within server and network infrastructure workloads, (b) identify and analyze the challenges of scaling the TCP/IP packet processing to 10 Gbps in both the server and network infrastructure environments, and (c) show how Intel server and network processors are addressing some of these challenges by discussing the relevant Intel research.

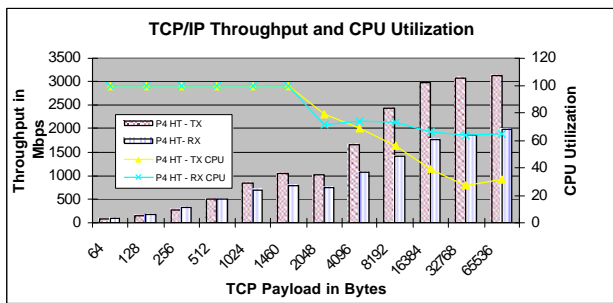


Figure 1: TCP/IP performance on Intel® Xeon™ processor

TCP/IP PROCESSING IN THE DATACENTER

In order to understand the impact of the TCP/IP processing on datacenter applications, we first need to characterize the networking requirements of these applications. In this section, we study applications ranging from firewall applications and load balancers (that primarily run on networking infrastructure equipment) to front-end web servers, application servers, and back-end database servers. We discuss the TCP/IP requirements of these workloads.

Server Workloads

The networking requirements for server workloads can be characterized by experimenting with commercial benchmarks such as SPECweb99 [3], TPC-W [4], and TPC-C [5]. Figure 2 illustrates the types of incoming and outgoing network traffic for these benchmarks.

To represent web server workloads, we chose SPECweb99 since it is a benchmark that mimics the load (70% static and 30% dynamic requests) in a web-server environment. A primary requirement for SPECweb99 is that the network rate needs to be within 320 Kbps and

400 Kbps per connection. Based on measurement and tuning experiments¹ in our labs, we find that the rate on a dual Intel Xeon processor-based system is around 330 Kbps. Based on this estimate, Figure 3 shows the network bandwidth requirements for the web servers as a function of the achieved performance (number of simultaneous connections; x-axis is normalized to 4000 connections).

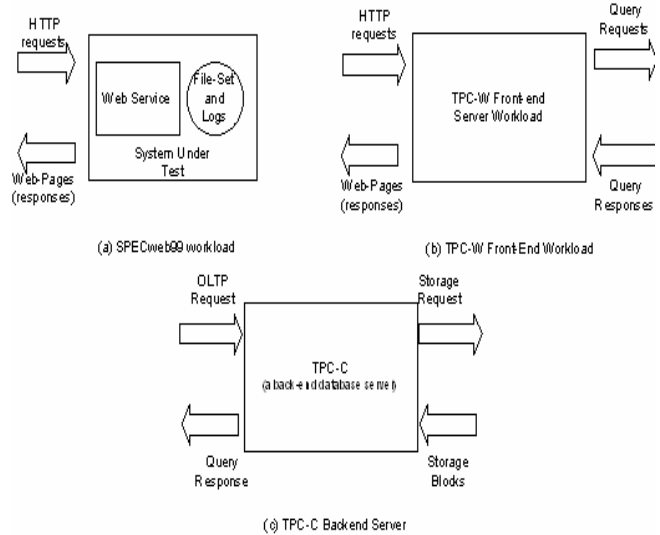


Figure 2: Networking content in server workloads

The relationship of network rate to performance is basically linear. It should also be noted that SPECweb99’s network traffic is dominant on the transmit-side, with an average application buffer size (average file size) of roughly 14 Kbytes.

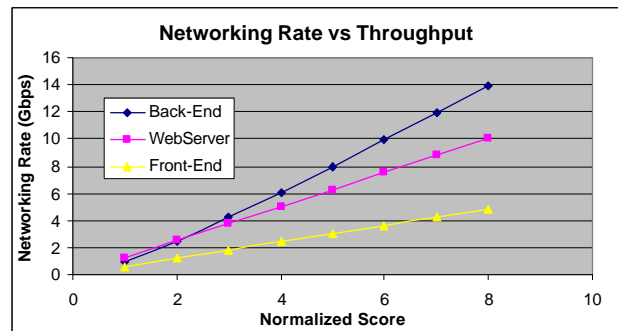


Figure 3: Networking rate for server workloads

To study front-end servers, we analyzed TPC-W, an e-commerce benchmark that models an online book store. TPC-W specifies 14 different types of web interactions that require different amounts of processing on the system(s) under test (SUT). Each interaction requires multiple requests and responses (for the HTML content, the images, and other data) between the clients, the web server, the image server, and the back-end database

server. TPC-W experiments¹ in our lab (for a scale factor of 10000 items) show that the SUT receives roughly 2.5 Kbytes of network data from the clients, while it transmits around 46 Kbytes of data to the clients. Based on these experiments, we also find that this outgoing network data is typically broken up into roughly 24 Kbytes per transaction transmitted by the web server and 22 Kbytes per transaction transmitted by the image server. The breakup of incoming network data from the clients is around 1.6 Kbytes to the web server and 0.9 Kbytes to the image server. Based on this data, in Figure 3, we show how the web server network bandwidth changes with an increase in throughput (measured in web interactions per second (WIPS); x-axis is normalized to 3000 WIPS).

Next, we study the networking content in a back-end database server by analyzing TPC-C, an online-transaction processing benchmark that is based on fulfilling database transactions in an order-entry environment. The purpose here is to study the potential networking requirement for storage-intensive servers that access storage blocks over TCP/IP. Based on measurements in our labs, we have estimated the number of IO accesses per transaction as a function of the overall throughput. Figure 3 shows the potential networking requirements of back-end servers as a function of overall performance (specified in number of transactions per minute (tpmc); x-axis is normalized to 40000 tpmc). It should be noted that the storage IOs per transaction depend not only on the performance level but also on the memory capacity available on the platform. It should also be noted that the application buffer size requested from the storage device is basically a physical page (8 Kbytes in our configurations).

Having studied the networking rates and payload sizes for these workloads, we also measured the number of instructions that are required to transmit or receive these payloads over TCP/IP (on a server running Windows Server 2003). Reference [6] describes the details of these experiments. Figure 4 shows the number of instructions (path length) required to transmit or receive various payload (buffer) sizes using TCP/IP. Based on the path length per payload as shown in Figure 4, the number of transmit and receives operations per transaction and the number of instructions per transaction for these three benchmark applications, we estimate the TCP/IP (data path) component. This is illustrated in Figure 5. We find that web server and front-end servers may spend

around 25% to 30% of their path length for TCP/IP processing. The packet processing component for back-end servers can be as high as 35% to 40%.

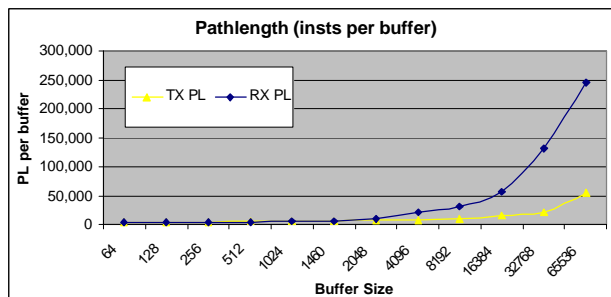


Figure 4: Path length for TCP/IP processing

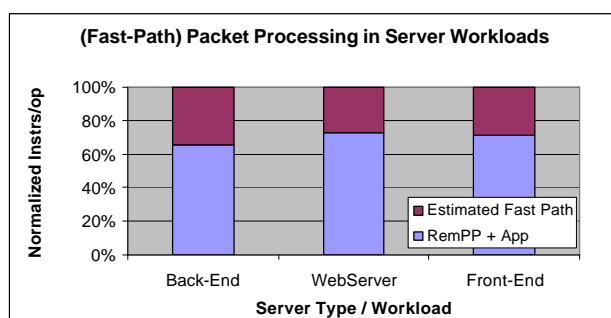


Figure 5: TCP/IP processing in server workloads

Network Infrastructure Workloads

Workloads in the network infrastructure are typically quite different from the workloads processed by end devices such as servers and workstations. Infrastructure equipment has the primary task of moving packets towards their final destinations. Examples of infrastructure equipment include routers, firewalls, and gateways.

As packets get closer to their final destinations in the Internet, the infrastructure equipment can get much smarter in its decision-making process to forward a packet.

Load balancing, URL redirection, L4 switching, web caching, and L7 switching are services that can be found in modern infrastructure equipment, and these services require a much deeper understanding of the TCP header and TCP payload in order for a forwarding decision to be made.

Layer 4 load balancing attempts to equally distribute the packet processing load amongst a group of end node servers. Packet forwarding decisions are based on the Layer 3 IP addresses and the Layer 4 TCP port numbers of arriving packets. The TCP payload does not need to be examined. Similarly, a Layer 4 switch will make its

¹ It should be noted that the commercial benchmark experiments that we discuss do not necessarily meet the strict reporting and auditing rules nor do they fully conform to the specification. Their use here is only to understand behavioral characteristics and should be treated as such.

circuit establishment, resource allocation, and port forwarding decisions based entirely on the IP addresses and TCP port numbers of arriving packets.

URL redirection is similar to Layer 4 load balancing in that the intent of the URL redirection is to equally distribute the web page server load amongst a set of end nodes. But, because each packet must be inspected for the destination URL, the TCP payload must be at least partially reassembled and searched for each packet received.

Layer 7 switches require a deep inspection of the TCP payload of each arriving packet in order to make a forwarding decision. The Layer 7 switch will establish circuits, allocate resources, and forward packets based on the IP addresses, TCP port numbers, and some other field or tag that has been identified in the TCP payload of each arriving packet. An example would be a Layer 7 switch for business e-commerce transactions. The e-commerce transaction switch would search each TCP payload for a specific XML tag or set of XML tags before making the final forwarding decision.

Unlike an end node that processes the entire TCP payload of each arriving packet, infrastructure equipment can process all, some, or none of the TCP payload depending on the types of services being provided.

Infrastructure equipment that processes TCP payloads will eventually forward the packets received. The transmitted packets are similar in size if not identical to the received packet sizes. The end result is that TCP processing in infrastructure equipment is symmetric with respect to receive TCP processing vs. transmit TCP processing. The symmetric TCP processing of infrastructure equipment can be quite different from the TCP processing found in end nodes. In a typical web server application for example, the client requests being received can be quite small, while the web pages being transmitted can be quite large.

TCP/IP PROCESSING IN SERVERS

In this section we discuss the major challenges faced by the servers in scaling the TCP/IP packet processing to multi-Gigabit rates without consuming excessive amounts of processing power. We then show how the Hyper-Threading Technology built into the Intel Pentium® 4 processors help address these challenges to

® Pentium is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

some extent. We also highlight some other potential techniques that help in this regard.

Challenges

TCP/IP processing on servers presents multiple challenges that are specific to their role and environment. A server's main role is to source and sink requests and data to and from clients and to do work on their behalf. This requires that the server terminates one or more connections from potentially many clients, simultaneously. The processing required to efficiently terminate client connections is the main TCP/IP processing challenge for network servers. A basic rule-of-thumb has long been used to estimate the network performance of servers. This rule-of-thumb states that it takes one hertz of CPU frequency to process one bit of TCP/IP network traffic in one second. For example, a 1 GHz processor should be able to source or sink 1 Gbit of TCP/IP network traffic. This "hertz per bit" rule is a generalization and only applies to the best case (bulk transfers) and in many cases the ratio becomes very large as seen in [Figure 6](#) [2]. Even in the best case, it is questionable whether it is acceptable to expend 20 GHz of CPU processing power to service TCP/IP on a full-duplex 10 Gigabit link.

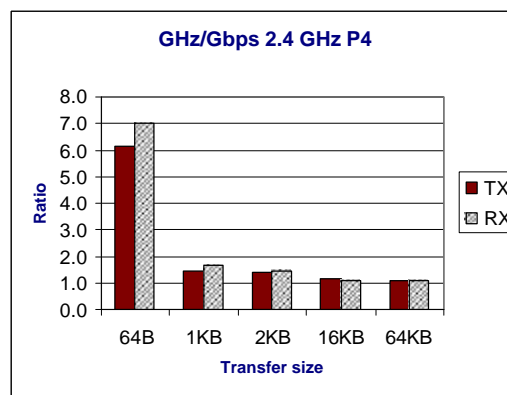


Figure 6: Measured GHz per Gbit

The challenge for the server environment is to enable the scaling of TCP/IP network bandwidth at the lowest possible CPU utilization. Low CPU utilization is important in order to leave sufficient processing cycles for useful work. There are multiple factors that make this a formidable challenge.

One major challenge for servers is the *buffering and copying of the data* carried by TCP/IP. These data copies are largely due to legacy applications and interfaces. The predominant application interface to TCP/IP networks is the socket interface. Traditional Berkeley sockets are synchronous in nature, and tend to favor the spending of

CPU cycles for generality. Even though some operating systems and applications have developed more performance oriented, non-blocking, application interfaces, many existing applications still rely on basic sockets because they are common across platforms and operating systems. The result is that some optimizations, in particular avoiding data copies, cannot be implemented while still supporting a large body of user application code. [Figure 7](#) is a high-level view of the end-to-end data path of a TCP/IP data transfer showing how the data at each end is copied by the CPU in order to effect the transfer.

The next major server challenge deals with *operating system integration and overhead*. It has long been shown that operating system overheads are a major factor in the performance of TCP/IP on a server [1]. The TCP/IP software stack on a server needs to share the CPU, memory, and IO resources with the OS. In order to do this, it needs to live by the rules of the OS in terms of scheduling, resource management, and interfaces. OS mechanisms such as system calls, preemptive scheduling, layered drivers, and interrupt processing, as well as data copying, are all sources of overhead that limit the efficiency and performance of TCP/IP on servers.

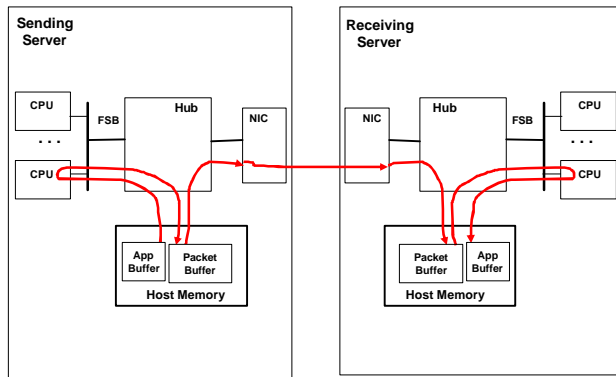


Figure 7: End-to-end data path

[Figure 8](#) shows an example of the processing breakdown for TCP/IP transmit. In this test case, the simple TTCP test program was transmitting 1 Kbyte packets on a dual processor server running the Linux[®] operating system. It clearly shows that the TCP/IP processing is not the dominant portion of the overall processing.

Another challenge for TCP/IP processing at the server is *memory latency and poor cache locality*. Server memory and cache architecture is optimized for the efficient execution of a broad set of applications. The TCP/IP software stack generally exhibits poor cache locality due largely to the fact that it deals with large amounts of control information and data that are entering or exiting the system through the network interface. For example,

an incoming packet is placed into memory by the network interface controller, then writes its status to a descriptor in memory, and generates an interrupt.

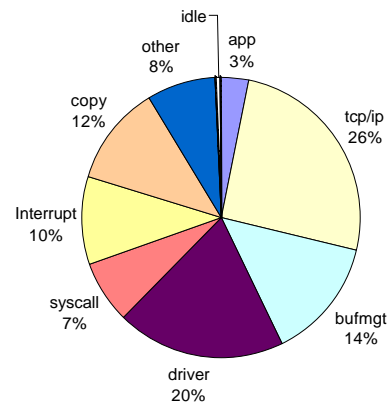


Figure 8: Server processing profile

The TCP/IP software must examine the control information including the descriptor and packet headers for each packet, and since the packet was just placed in memory, this will always cause a cache miss and thus a CPU stall. The data carried by TCP networks also exhibits poor temporal cache locality because new data are continuously being sent or received [14] and it has been shown that IO-intensive workloads cause a large number of memory stalls due to cache misses [15]. Given the rapid advances of network link technologies, combined with the growing disparity between processor speeds and memory latency (see [Figure 9](#)), this problem becomes a larger challenge over time.

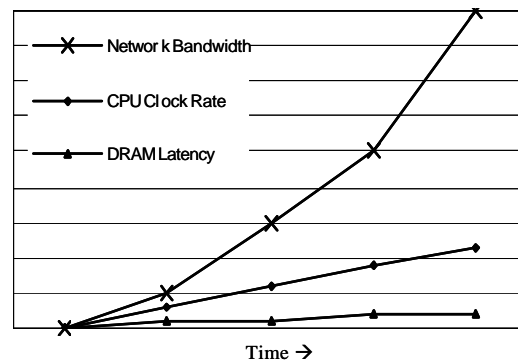


Figure 9: Performance improvement trends

Addressing the Challenges

In order for servers to scale to 10 Gigabits of TCP/IP performance and beyond, the problems of data copies, operating system overheads, and memory latency must be addressed. As in many complex systems, these issues and their solutions are inter-related.

One method to alleviate the need for data copies is to extend the sockets interface to allow non-blocking and/or

asynchronous interfaces. The fundamental idea is to be able to post transmit and receive buffers without blocking the application, and to synchronize the application and the networking stack when the operation is complete. This allows the underlying networking stack to prepare the application buffers and move data into or out of them while the application makes forward progress. There are examples where progress has been made in this area. Microsoft has extended the basic socket interface with their Winsock API that includes capabilities such as overlapped IO and completion ports. There are additional ongoing developments to extend the socket semantics such as the Open Group's Extended Sockets interface, and the Linux AIO (Asynchronous IO) extensions. Changing and extending long-standing APIs is a difficult and long-term problem because of the large number of existing legacy applications.

Another method for avoiding data copies is through protocol extensions. An example of this is the ongoing industry-standard effort to define Remote Direct Memory Access (RDMA) [13]. The RDMA proposal adds protocol layers that enable the data to carry information about their ultimate memory location, thus eliminating the need to copy them at the end station. The RDMA model has been implemented in previous networks [8, 9], and this is now being adapted to IP transports such as Stream Control Transmission Protocol (SCTP) and TCP.

Finally, rather than trying to avoid the data copy problem altogether, it may be possible to optimize it such that it is not a limiting factor to application performance. The ability to move data more efficiently via the use of advanced instruction set features, or to be able to overlap the data movement operation with other useful computations are possible means to optimize the data copy operation.

Another major server challenge to be addressed is *operating system overhead and integration*. As seen in Figure 8, the actual TCP/IP processing component is relatively small, while the combination of interrupt processing, system calls, data copies, buffer management, and other OS-related processing make up the bulk of the overhead.

To address the OS overhead issue, there has been an ongoing effort to offload large portions of the TCP/IP networking stack from the OS onto a device. These devices, generally called TCP Offload Engines or TOEs, generally offload the bulk of the TCP and IP processing combined with the Layer 2 media controller. TOE devices have at least two challenges to overcome. First, they are dependent upon interface support from the operating systems that they are trying to offload. Another challenge for TOEs is that they tend to make

tradeoffs between cost, performance, and flexibility. TOE devices that are fixed function, or hard coded, promise the best performance, but lack the flexibility afforded by software implementations. Processor-based, programmable TOE devices are more flexible, but have performance limitations, depending on the capabilities of their processing core(s), and they tend to be more expensive because of the processor and associated memory costs.

The Embedded Transport Acceleration (ETA) project [7] at Intel Research and Development has explored the partitioning of multi-processor systems in order to isolate the network-related processing from the operating system. The ETA prototype has shown that an Intel processor that has been partitioned away from the host operating system can perform TCP/IP processing more than twice as efficiently as a normal symmetric multi-processing system. Figure 10 shows the relative amount of CPU work that is expended for several pieces of the TCP/IP processing pie such as TCP/IP itself, buffer management, driver processing, and other OS-related processing.

Yet other server challenges to address are the issues around memory latency and cache locality. There are at least two methods that can be used in order overcome this challenge: concurrent processing and extensions to existing cache coherency protocols.

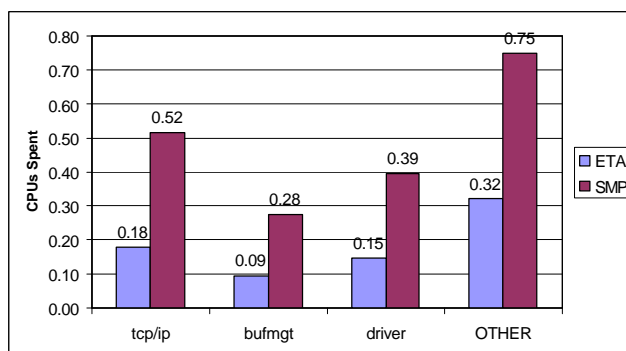


Figure 10: ETA processing efficiency

Server networking workloads tend to allow a great deal of concurrent processing. Most servers must deal with a large number of TCP/IP streams, and a packet of a given stream has minimal dependencies on the packets of another stream, and thus can be processed simultaneously. Concurrent processing of TCP/IP streams can be accomplished using multiple threads. Thread concurrency allows a given processing unit to hide the latency of memory accesses by overlapping the processing of a given packet while the processing of another packet is stalled waiting for memory. Figure 11 is a conceptual diagram of overlapping useful “work”

with memory stalls when threads are processing packets from multiple TCP/IP streams.

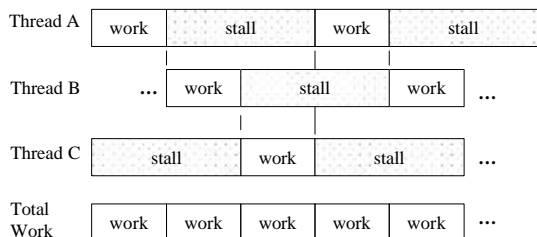


Figure 11: Thread concurrency

The Intel Pentium 4 processor supports Hyper-Threading (HT) Technology, a form of simultaneous multi-threading (SMT) that allows a single processor to appear as two (or more) logical processors. SMT allows multiple instruction streams to be issued simultaneously in order to more efficiently fill the CPU's instruction pipeline in the event of a stall, including stalls caused by long memory access latencies. We measured the effect of the HT multi-threading on simple throughput tests and saw roughly 30% to 50% improvements in throughput for both transmit and receive, as seen in [Figure 12](#).

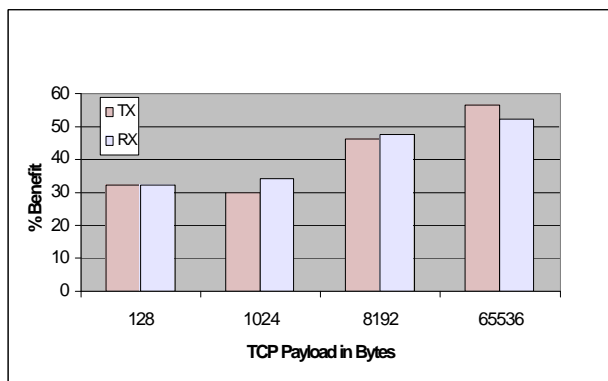


Figure 12: Benefits of SMT multi-threading

Memory stalls during packet processing can also be avoided by extending the semantics of certain memory transactions. If incoming data could be placed directly into the cache of the CPU that needs to do the TCP/IP processing, memory accesses could be avoided that would otherwise cause stalls in the CPU execution pipeline. Examples could include the control information, such as device descriptors and TCP and IP packet headers, as well as the data that are the payload of the packets. We have simulated the effects of directly depositing packets into the cache of the CPU to examine these effects. [Figure 13](#) shows simulation results of one possible scenario where the total time to process a given TCP/IP packet could be reduced substantially if the entire packet

was deposited directly into the CPU cache by the network interface. In this case, the memory references to the Network Interface Card (NIC) descriptor and the packet headers (the packet read latencies) would be replaced by much faster cache references, as well as the memory read references used to copy the payload data.

TCP/IP PROCESSING IN NETWORK INFRASTRUCTURE

In this section we focus on the TCP/IP processing in the network infrastructure. We present major challenges that limit the scalability of the TCP/IP packet processing to multi-Gigabit rates and show how these challenges can be overcome to a significant extent with systems based on Intel IXP2850 network processors.

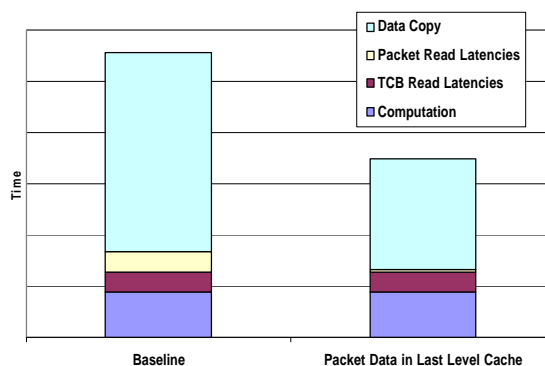


Figure 13: Effect of direct cache references

Challenges

TCP/IP processing performed by infrastructure devices presents an interesting set of challenges. Some of the challenges are a result of the complexity of TCP processing itself; for example, TCP checksumming. But most of the challenges presented in processing TCP relate to the environment in which TCP resides. Examples of these types of challenges include hiding memory latencies, lookup engines that can scale to millions of entries deterministically, and secure platforms with encrypted TCP payloads.

We highlight these challenges by showing an example of a converged firewall platform built using the Intel IXP2850 Service-Specific network processor. The converged firewall performs the normal packet forwarding and filtering that a standard firewall performs with the added features of IPSec for platform security and intrusion detection. The intrusion detection feature requires deep packet inspections into the TCP payloads. The IPSec feature requires us to perform decryption or encryption on every packet.

Our converged firewall can have millions of TCP connections open simultaneously. The resulting memory bandwidth that can result from processing packets for each of these connections can be daunting. This memory bandwidth is a combination of the packet data being moved into and out of memory, and the TCP context for each connection being read, modified, and written back to memory. Also, since our converged firewall is performing intrusion detection services on every packet received, this further increases the bandwidth utilization that the converged firewall will consume.

The chart in [Figure 14](#) depicts typical memory bandwidth requirements while running a simple firewall forwarding application with TCP in a 10 Gigabit Ethernet environment. Note that the chart shows memory bandwidths resulting from simultaneously receiving and transmitting packets at various sizes at 10 Gigabit rates.

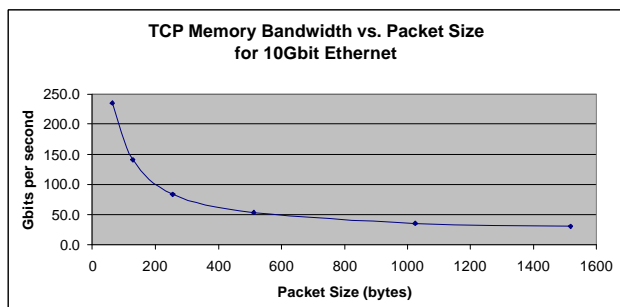


Figure 14: Memory bandwidth requirements

Related to the bandwidth of the memory being moved is the associated latency with waiting for a particular memory transaction to complete. For example, if a firewall thread in the IXP launches a memory read to fetch TCP state associated with a new packet which has arrived, the thread must wait for the associated TCP data to return from the memory controller before it can continue to process the packet. While it waits for the TCP data, the thread can perform no processing on the newly arrived packet. The thread must either process data associated with a different packet, or it must swap out and go to sleep.

Typical memory latencies for various IXP DRAM memory accesses are shown in [Figure 15](#).

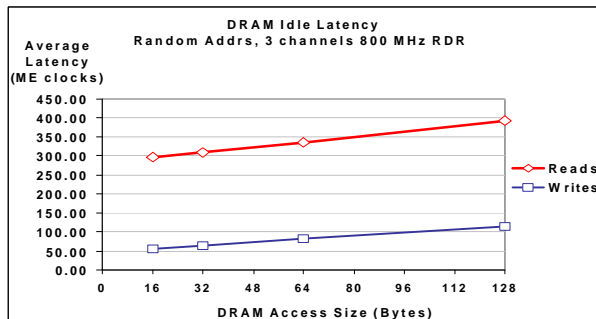


Figure 15: DRAM access latencies

By comparison, the chart shown in [Figure 16](#) illustrates the packet arrival times of various small frame sizes as network speeds increase. Note that at 10 Gigabits per second, the packet arrival times for small frames are considerably smaller than the smallest DRAM read latency shown in Figure 15.

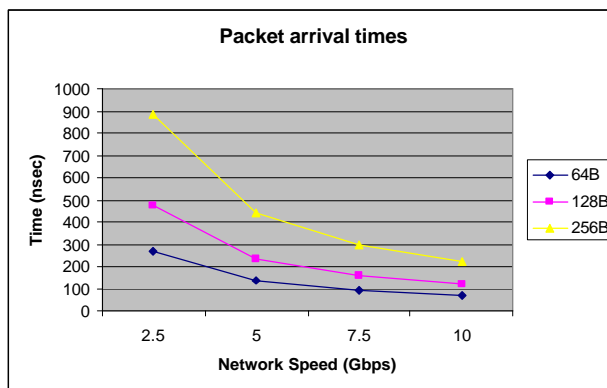


Figure 16: Packet arrival times

Context lookups while processing TCP connections can become problematic as the number of concurrent TCP connections grows. Context lookups are required to search for TCP state associated with a received TCP/IP packet. For a converged firewall that must support one million concurrent TCP connections, the challenge is to construct a lookup engine that will remain deterministic in returning a result with all one million connections established.

One of the difficulties in implementing a TCP state search is that lookup tables can require several dependent DRAM reads in order to resolve a single search. The chart in [Figure 17](#) details the average and worst-case latencies to perform a TCP context search using a simple hash table scheme. Note that the worst-case latencies approach two orders of magnitude difference as compared to the small packet arrival times illustrated in Figure 16.

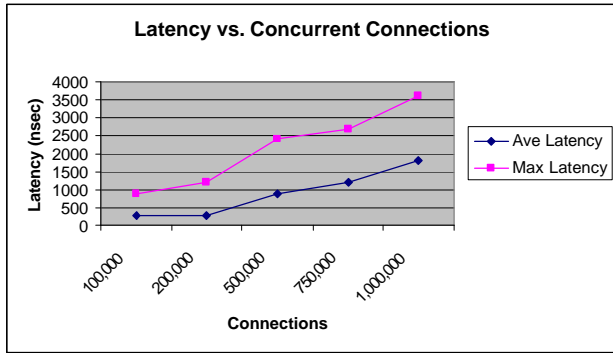


Figure 17: Context lookup latencies

TCP checksum and copying TCP payload data to an application are both problematic for our converged firewall. Both the checksum and the payload copy operate on every byte of the packet, and both will require a number of memory accesses in order to complete. The number of memory accesses is dependent on the size of the packet; the larger the packet the more memory accesses necessary. The memory accesses to complete the TCP checksum and the payload copy will increase both the memory bandwidth utilization and the total latency of our converged firewall.

The IPSec feature of our converged firewall will require us to decrypt a large portion of the arriving packets, and to encrypt a large portion of packets being forwarded. This can be problematic as most crypto routines are extremely computationally expensive, and quite complicated to implement using general-purpose CPUs. Crypto acceleration hardware is available from several third-party vendors. However, the bus interfaces to these chips may not provide the bandwidth or latency to allow us to achieve our 10 Gbps crypto rates, or the buses may be proprietary and require special hardware to interface with.

Addressing the Challenges

A firewall application built using an IXP2850 addresses the challenges of TCP/IP processing in several ways.

The IXP provides multiple memory interfaces that each use an independent set of internal buses to deliver data. The DRAM interface provides three channels of RDIMM access for a total DRAM bandwidth of 38 Gbps for reading or writing. The SRAM interface provides four channels of quad data rate (QDR) SRAM access for a total bandwidth of 48 Gbps for reading and writing. The Scratch interface provides a single channel of standard SRAM access for a total bandwidth of 22 Gbps for reading or writing. The end result is a memory subsystem capable of delivering over 100 Gbps of total memory bandwidth.

By allocating our TCP data structures across the various memory interfaces, we achieve a very good distribution of bandwidth utilization. This bandwidth efficiency yields very high returns as our packet processing rates increase.

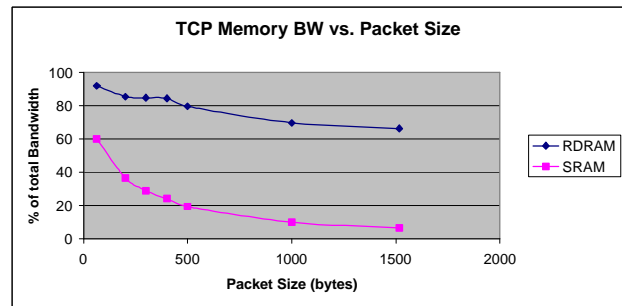


Figure 18: Memory bandwidth headroom

To help address the memory latency problem the IXP provides multiple processing cores called microengines, with each microengine having up to eight threads. The multiple threads allow us to hide the latency associated with the IO transactions of packet processing. Threads that have completed their memory transactions are running and processing while at the same time threads that are waiting for memory transactions to complete are sleeping.

Figure 19 illustrates an example of multiple threads hiding latency and computes associated with packet processing:

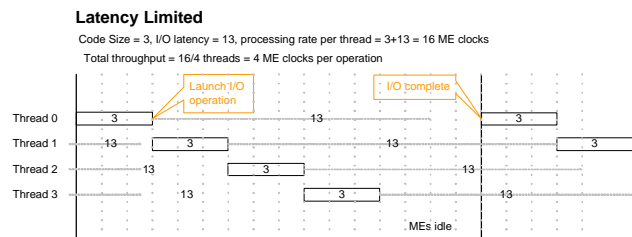


Figure 19: Multithreading

The first case to consider is Thread 0 running alone. In this case the total operation requires 16 microengine cycles, and there are 13 idle cycles while waiting for the IO transaction to complete. In the scenario with all four threads running over the same 16 microengine cycle period, we have processed four times more data, and reduced the total idle time to 4 microengine cycles.

The IXP implementation of the firewall application provides several hardware mechanisms useful in helping solve the TCP context lookup problem. Context lookups by the firewall are made more deterministic by implementing connection tables using multiple fixed size arrays, and using well-behaved hash algorithms to

distribute the established TCP connections evenly amongst the various fixed size hash buckets.

The IXP provides a hardware hash unit based on an irreducible polynomial. The hardware hash unit is used to return well-distributed hash results that in turn are used to generate offsets into the various hash tables. Also, the irreducible nature of the hash polynomial allows us to store a small portion of the hash result with each hash bucket to greatly simplify lookup hit and miss decisions.

The hash tables are distributed across both SRAM and DRAM to maximize bandwidth efficiency and to take advantage of parallelism in the hash table accesses across multiple threads. Finally, each IXP microengine contains a hardware Cyclic Redundancy Check (CRC) unit capable of reducing very large key sizes to 32-bit keys to streamline the use of the hardware hash engine for both IPv4- and IPv6-based addressing.

The chart in [Figure 20](#) shows the resulting average and worst-case lookup latencies using the advanced search functionality of the IXP firewall. Note the improvement over using the search engine based on a simple hash table design shown in [Figure 17](#).

The IXP provides hardware to assist in accelerating TCP checksum calculations for arriving packets. This greatly simplifies all TCP checksum calculations as the checksum can now be computed incrementally.

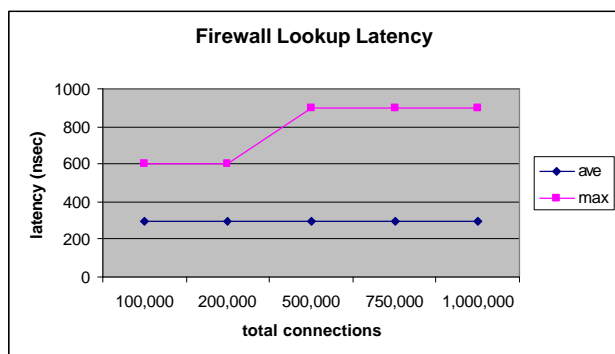


Figure 20: Firewall lookup latencies

The dependency on the payload length is eliminated, and checksums for both incoming and outgoing TCP segments can now be calculated by a function dependent solely on the length of the packet headers, which is a much more deterministic calculation. This greatly reduces both the memory bandwidth and the latency associated with computing the TCP checksum.

[Figure 21](#) compares the read bandwidth required to compute TCP checksums for various frame sizes at 10 Gbps using (1) software only and (2) the hardware

acceleration provided by the IXP-based converged firewall.

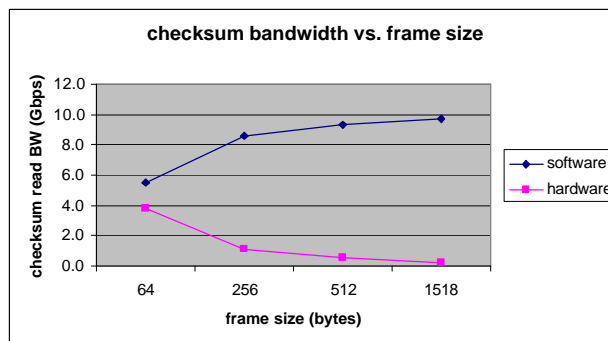


Figure 21: TCP checksum bandwidth

To help in eliminating data copies, the IXP provides hardware mechanisms to register or pre-allocate memory buffers in the system. This allows the packet-processing functions of the firewall to use the same set of allocated memory buffers as the security functions without incurring additional overhead. The firewall can pass all data to and from TCP by reference without the need for performing data copies.

To address the challenges of performing high-speed crypto, the IXP contains an integrated cryptographic unit capable of performing DES, 3DES, and AES crypto calculations. The crypto unit can also perform SHA-1-based message authentication. The integrated crypto allows the converged firewall to achieve 10 Gbps encrypt and decrypt rates without having to access an external bus or have special hardware to interface with the crypto functions.

CONCLUSIONS

Accelerating TCP/IP is essential to allow network-centric applications to benefit from the increase in the Ethernet bandwidth. There are several challenges that currently prevent TCP/IP processing from scaling to these speeds. On servers, the major challenges faced include memory latency, O/S overhead, and data copies. The impact of memory latency can be overcome to a large extent with multithreading techniques (such as Hyper-Threading Technology). The overhead associated with data copies can be minimized by using advanced streaming instructions or specialized hardware for copying data asynchronously. In addition, intelligent prefetching techniques such as placing the incoming data directly into the cache (DCA) also help. For network infrastructure equipment, the major hurdles include memory latency, memory bandwidth issues, and connection look-ups when supporting thousands of simultaneous connections. These are mostly overcome by

the highly parallel Intel IXP2850 network processors with a high-bandwidth memory subsystem. Special hardware assists like checksum and hashing engines on these network processors enable TCP/IP processing at 10 Gbps rates.

ACKNOWLEDGMENTS

We express our thanks to Michael Espig, Ramesh Illikkal, Annie Foong, and Ram Huggahali for their contributions to the server networking studies. We also thank Raed Kanjo, James Mead, and Will Auld for providing us insights to the measurement and analysis of commercial workloads. We thank Dale Juenemann, Wajdi Feghali, and Matt Adiletta for their contributions, guidance, and inspiration on TCP/IP performance studies on the IXP.

REFERENCES

- [1] D. Clark, et. al., "An Analysis of TCP Processing Overhead," *IEEE Communications*, June 1989.
- [2] A. Foong, T. Huff, H. Hum, J. Patwardhan, and G. Regnier, "TCP Performance Analysis Re-visited," *IEEE International Symposium on Performance Analysis of Software and Systems*, March 2003.
- [3] "SPECweb99 Design Document," available online at <http://www.specbench.org/osg/web99/docs/whitepaper.html>
- [4] "TPC-W Design Document," available online on the TPC website at www.tpc.org/tpcw/
- [5] "TPC-C Design Document," available online on the TPC website at www.tpc.org/tpcc/
- [6] S. Makineni and R. Iyer, "Performance Characterization of TCP/IP Processing in Commercial Server Workloads," to appear in the 6th *IEEE Workshop on Workload Characterization*, Austin TX, Oct. 2003.
- [7] G. Regnier, D. Minturn, G. McAlpine, V. Saletore, and A. Foong, "ETA: Experience with an Intel Xeon Processor as a Packet Processing Engine," *Hot Interconnects 11*, August 2003.
- [8] D. Cameron and G. Regnier, *The Virtual Interface Architecture*, Intel Press, 2002.
- [9] InfiniBand Trade Association, <http://www.infinibandTA.org>.
- [10] *Optimizing Intel IXP2800 RDRAM Performance: Analysis and Recommendations*, Intel Corporation, Version 001, October 2003.

- [11] "The TTCP Benchmark," <http://ftp.arl.mil/~mike/ttcp.html>
- [12] A. Jone, J. Ohlund, 2002 *Network Programming for Microsoft Windows Second Edition*, Microsoft Press, Redmond, Washington.
- [13] RDMA Consortium, <http://www.rdmaconsortium.org/home>
- [14] B. Ahlgren, M. Bjorkman, and P. Gunningberg, "Towards Predictable ILP Performance – Controlling Communication Buffer Cache Effects," December 1995.
- [15] B. Bershad and J. Chen, "The Impact of Operating System Structure on Memory System Performance," *The 14th Symposium on Operating System Principles*, 1993.

AUTHORS' BIOGRAPHIES

Dave Minturn is a senior architect in the Corporate Technology Group at Intel. He has 20 years of experience in the computer industry and has been with Intel since 1989. He has worked on parallel super computers, IO subsystems, networking protocols, and file systems. He is currently investigating various threading models and other optimizations to efficiently scale TCP/IP packet processing. His e-mail is dave.b.minturn@intel.com.

Greg Regnier has been at Intel since 1988 and is currently in Intel's Corporate Technology Group. Greg spent the first part of his Intel career in the Supercomputing Systems Division where he worked on the development of the message passing subsystems of the Touchstone Delta and Paragon parallel supercomputers. Greg was part of the core architecture team that developed the Virtual Interface Architecture. He co-authored a book on the Virtual Interface Architecture with Don Cameron that was published in 2002. His recent work has concentrated on improving the scalability and performance of the datacenter network using standard networking media and protocols. His e-mail is greg.j.regnier@intel.com.

Jon Krueger is a senior architect and software engineer with the Polymer Memory Group at Intel. He has worked in the field of networking for the past 14 years, and has been specializing in applications running on network processors for the past four years. His most recent focus has been in the area of TCP acceleration. Jon holds B.S. degrees in both Electrical Engineering and Computer Science from Oregon State University. His e-mail is jon.krueger@intel.com.

Ravishankar Iyer has been with Intel since 1999 and is currently with the Communications Technology Laboratory. Since joining Intel, he has worked in the areas of Internet server architecture and performance, commercial workload characterization, peer-to-peer computing, and network acceleration. Most recently, his work has been focused on characterizing and enhancing packet-processing performance in servers. Previously, he also worked for the Computer Systems Laboratory at Hewlett Packard Laboratory and in the Server Engineering Group at Intel Corporation. He received his Ph.D. degree in Computer Science from Texas A&M University in 1999. His research interests are in the areas of Internet/networking protocols, computer architecture, distributed computing, and performance evaluation. He has published over 30 papers in these areas. His e-mail is ravishankar.iyer at intel.com.

Srihari Makineni is a senior software engineer in the Corporate Technology Group at Intel. He joined Intel in 1995 and has worked on video conferencing, multimedia streaming, web/e-commerce applications, and system and server management technologies. His areas of interest include networking and communication protocols and computer architecture. His current work is focused on developing techniques for accelerating packet-processing applications on IA-32 and IPF architectures. He holds a Masters degree in Electrical and Computer Engineering. His e-mail is srihari.makineni at intel.com.

Copyright © Intel Corporation 2002. This publication was downloaded from <http://developer.intel.com/>.

Legal notices at <http://developer.intel.com/sites/developer/tradmarx.htm>.

For further information visit:

developer.intel.com/technology/itj/index.htm