TCP/IP Offload Engine – Implementation using VxWorks and its Performance Analysis

V. Senthilkumar,
TATA Consultancy Services, Bangalore, India
Email: senthilkumar.vijayakumar@tcs.com

Abstract–**TCP/IP Offload Engine (TOE) is a system architecture aimed at offloading network processing from the host running an Internet server. By reducing the amount of TCP/IP processing handled by microprocessor and server I/O subsystem, TOE eases the server networking bottleneck and enables applications to take full advantage of the networking capabilities.**

In this paper, TOE is implemented using Intel IXP 1200 Network Processor running VxWorks, a Real-Time Operating System (RTOS). Memory-Mapped Networking is implemented in the next stage. The performance of TOE is then evaluated using metrics such as Throughput, CPU Utilization and latency on IXP 1200, Single core and Dual core processors. Simulation results show that TOE implementation can achieve an overall performance improvement in the range of 10 to 15% as compared to traditional TCP/IP processing.

I. Introduction

TCP/IP Offload Engine is a technology gaining popularity in high-speed networks where CPU has to dedicate more resources for network processing. It eases the server networking bottleneck and enables applications to take full advantage of the networking capabilities. TOE can be executed on a dedicated processor, node, or intelligent network interface using low-overhead, non-intrusive communication between itself and the host running the server application.

A. Purpose of TOE

Ethernet has become the most popular networking protocol for Local Area Networks (LANs). As networking has grown in popularity, Ethernet network link speeds have increased faster than computer processor speed. The growth of Ethernet from 10 Mb/s to 10 Gb/s has surpassed the growth of microprocessor performance in mainstream servers and computers. A fundamental obstacle to improving network performance is that servers were designed for computing rather than input and output (I/O). The Internet revolution has drastically changed server requirements, and I/O is becoming a major bottleneck in delivering high-speed computing [1]. The main reason for the bottleneck is the TCP/IP stack being processed at a rate lesser than the network speed.

The processor, which is designed primarily for computing and not for I/O, cannot keep up with the data flowing through the network. TOE solves this problem by offloading computationally intensive I/O tasks to the microprocessor.

B. Traditional TCP/IP Processing

The TCP/IP protocol suite consists of a set of layered protocols implemented in software and integrated into the operating system. The application interface to the TCP protocols has typically been through the socket interface [2]. In traditional TCP/IP implementations, every network transaction results in a series of host interrupts for various processes, such as send-packet segmentation and receive-packet processing.

On the contrary, TOE can delegate all processing related to sending and receiving packets to the network adapter leaving the host server's CPU more available for applications. As TOE involves the host processor only once for every application network I/O, it significantly reduces the number of requests and acknowledgments that the host stack must process. This finds applications in thin clients where the processing power of the client is swamped by TCP/IP processing. Freeing up the client's limited processing power will permit it to process other tasks with improved efficiency. Fig. 1 shows the comparison between standard TCP/IP and TOE-enabled TCP/IP stacks.

II. Motivation

The motivation behind implementing TOE is to decrease the load on the target CPU. A rough estimate of the CPU processing necessary to handle a given Ethernet link speed is one hertz of CPU processing is required for every single bit per second of network data processed.

Earlier implementation of TCP/IP processing over Ethernet was accomplished by software running on the server's central processor. As network connections scale beyond Gigabit

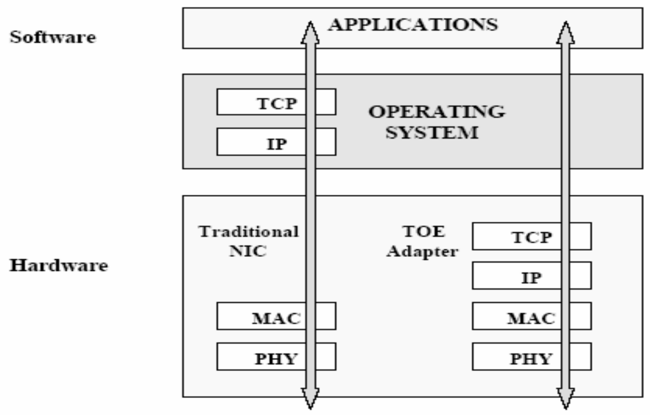

Fig. 1 Comparison between standard TCP/IP and TOE-enabled TCP/IP stacks

Ethernet speeds, burden on CPU increased due to large amount of TCP/IP protocol processing. Reassembling out-of-order packets, resource-intensive memory copies and interrupts have augmented the load on host CPU. TOE is emerging as a solution to limit the processing required by CPUs for networking links.

III. DESIGN AND IMPLEMENTATION

A. *Network Processor in TCP/IP Offloading*

Network Processors (NPs) are specifically designed processors for combined fast and flexible packet handling. Typically pairing an embedded processor complex with application-specific instructions and coprocessor support, NPs can achieve higher-layer packet processing at line speeds of several Gb/s [3]. Besides the instruction set, the entire design focuses on high-performance packet processing, including memory, hardware accelerators, bus, and I/O architecture [4]. NPs are well suited for most packet-processing tasks such as context switching, load balancing, traffic conditioning and network security. This paper focuses on the specific role of NPs and various processors in TCP/IP offloading.

B. *TCP/IP Stack Implementation*

Socket Programming is implemented using VxWorks tasks at the sending end communicate with the receiver through Ethernet or backplane. Socket communications can occur between VxWorks tasks and host system processes in any combination. Only for high speed communication TOE related system calls add to the improvement of system performance. Socket communications among processes are exactly the same, regardless of the location of the processes in the network or the operating system where they run. Fig.2. Shows the System Calls and their Scheduling sequence for implementation of TCP/IP Stack.

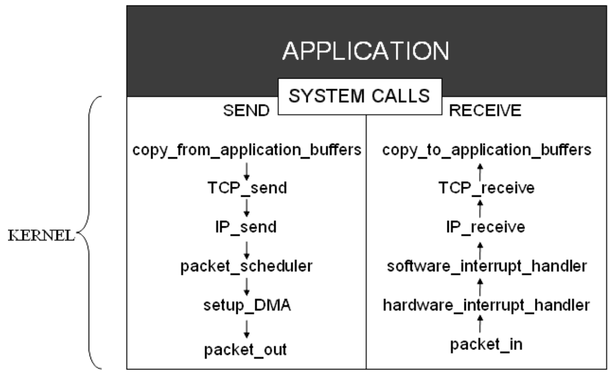

Fig.2. System Calls and their Scheduling sequence for implementation of TCP/IP Stack

The implementation is based on a client-server communication model. Fig. 3 shows the Network Protocol Stacking (TCP/IP Stack) & TOE Engine. TOE implementation in an Embedded RISC Processor such as Intel IXP 1200 Network Processor involves changes in network and transport layer.

C. *Stream Socket Implementation*

The server communicates with clients using stream-oriented (TCP) sockets. A socket is a communications end-point that is bound to a UDP or TCP port within the node. Stream sockets use TCP to bind to a particular port number which establishes protocol-based link [5]. Under VxWorks, application can use the sockets interface to access features (Eg. Multicasting) of the Internet Protocol suite.

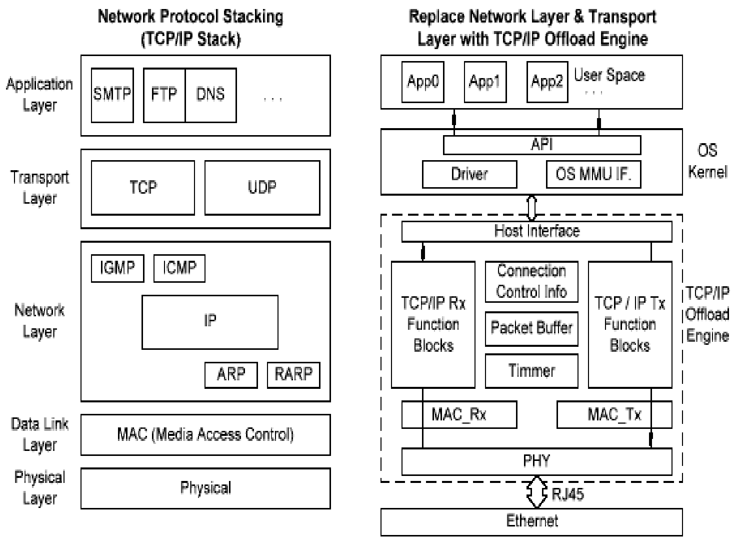

Fig. 3. Network Protocol Stacking (TCP/IP Stack) & TOE Engine.

Another process, running on any host in the network, can create a stream socket and request that it be connected to the first socket by specifying its host Internet address and port number. VxWorks allocates and initializes memory for the network stack only once, at network initialization time .The network stack uses routines to get the memory needed for data transfer. After the TCP sockets are inter-connected, a virtual circuit and Remote Procedure Call (RPC) is set up between them, allowing reliable socket-to-socket communications [6]. TCP/IP stack algorithm developed in VxWorks offloads the network and transport layer in hardware and other supported protocols, provided proper system calls are supplied by the hardware to offload those protocols from the host. The RTOS also performs non real-time processing tasks like connection management or exception handling and provides efficient communication with host applications by means of bypassing the operating system.

D. *Remote Procedure Call Component*

The Remote Procedure Call (RPC) component allows for distributed computing. RPC server offers services to external systems as remotely callable procedures. A remote RPC client can invoke these procedures over the network using the RPC protocol. To use a service provided by an RPC server, a client application calls routines (known as stubs) residing on the RPC client. RPC client in turn invokes remote procedure calls residing in the RPC server on behalf of the calling application. The primary goal of RPC is to make remote procedure calls transparent to applications invoking the local call stubs. To the client application, calling a stub appears no different from calling a local procedure. The RPC client and server can run on top of different operating systems, as well as different types of hardware.

To hide server remoteness and platform differences from client application, data flows between the two systems running RPC call must be translated to a common format. External Data Representation (XDR) is a method that represents data in an OS and machine-independent manner. The RPC client translates data passed in as procedure parameters into XDR format before making the remote procedure call. The RPC server translates the XDR data into machine-specific data format upon receipt of the procedure call request. The decoded data is then passed to the actual procedure to be invoked on the server machine. The output data of this procedure is formatted into XDR when returning it to the RPC client. The RPC concept is illustrated in Fig.4.

There are remote procedure calls (RPCs) in the distributed environment of embedded systems. RPC provides the inter task communication between two system work in the peer-to-peer communication mode, and not in the client server mode. Client and server can make either local or remote calls.

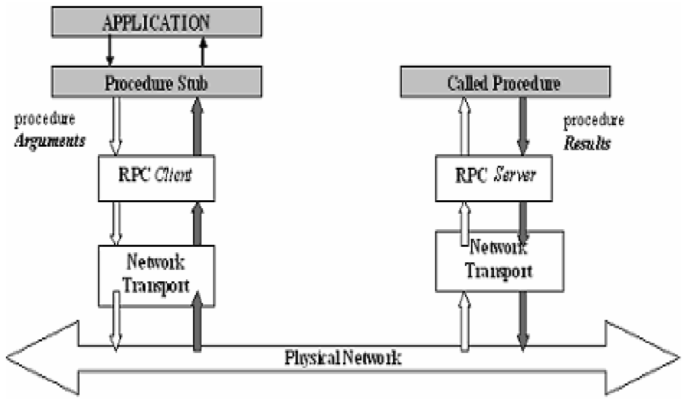

Fig. 4. Remote Procedure Calls between Client and Server

E. Stream Memory-Mapped Sockets

Memory-Mapped Networking API is a memory-mapped socket interface between the application and TCP server with support for zero-copy asynchronous communication .VxWorks includes a set of socket calls based on a data abstraction called a zero copy buffers, which helps to share data buffers (or portions of data buffers) between separate software modules. This type of socket interface allows applications to read and write without copying data between application buffers and network buffers. The TCP subset of this new interface is called zero-copy TCP. Zero–copy buffers can be called and processed through particular system calls executed in the main loop of TCP server program.

Zero – copy buffer segments can minimize data copying: A data buffer can be incorporated (by reference, so that only pointers and lengths move around) into a new zbuf segment. Conversely, we can move pointers to the data into zbuf segments, and examine the data directly.

The above implementation will decrease the number of transactions across the system I/O bus: Memory bus copies help to reduce the wait time for the bus and decreases latency. With TOE, network adapters will be able to respond faster, therefore enabling a faster end-to-end communication.

IV. Results and discussions

Metrics such as Throughput, CPU Utilization and latency are used to evaluate the performance of TOE. The metrics are measured on Intel IXP 1200 Network Processor with Intel Pentium 4 host, standalone systems with Intel Pentium 4 (Without HT), Intel Pentium D and AMD Athlon desktop processor.

A. Throughput

Throughput has always been the key indicator for network performance. Fig. 5 shows the throughput tests with various frame sizes (bytes). The average throughput of the Intel IXP1200 with Intel Pentium 4 host was 7.20Gbps.This was 135% higher than standalone AMD Athlon processor (3.00Gbps), and 109% higher than standalone Intel Pentium 4 processor (3.38Gbps). The average throughput of the standalone Intel Pentium D processor throughput was 4.53Gbps, 34% higher than AMD Athlon processor, and 25% higher than Intel Pentium 4 processor.

B. CPU utilization

CPU utilization is also an essential performance measurement for CPU. Traditional adapters typically consume 1 Hertz of processor for each bit of TCP/IP data moved. There are two major components that drive this baseline: the amount of data copied within the protocol stack as data is moved from network buffers to user buffers and back, and the number of transactions to move the data. In order to normalize the CPU utilization, the results are plotted to a CPU efficiency measure defined as the throughput achieved divided by the CPU utilization, and presented in units of Megabits per percent CPU (Mbps/%CPU). As shown in Fig. 6, the TOE with Intel IXP 1200 had the highest transmit throughput and best CPU efficiency index across all I/O sizes from 512 bytes to 64KB .For IXP1200 with Intel Pentium 4 host, average CPU efficiency was 3.5 times higher than AMD Athlon and 3.9 times higher than Intel Pentium 4 processor. Among the offload without Network processor, Intel Pentium D provided best performance and highest CPU efficiency.

C. Latency

Latency is another performance metric impacted by the use of TCP offload. Decrease in number of transactions between system I/O bus and memory bus helps reduce the waiting time for the bus, there by reducing latency. By performing TCP/IP processing in the adapter, the number of transactions are reduced. This TOE makes to respond faster, therefore enabling a faster end-to-end communication.

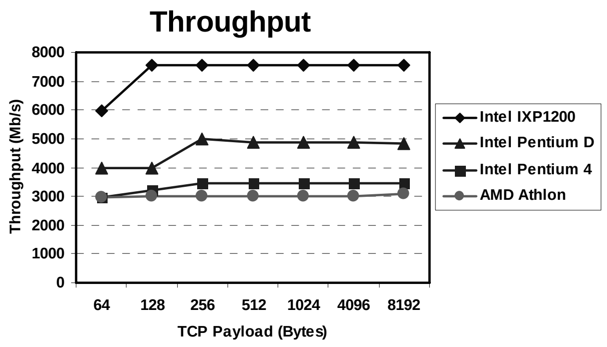

Fig.5. Throughput Vs TCP Payload

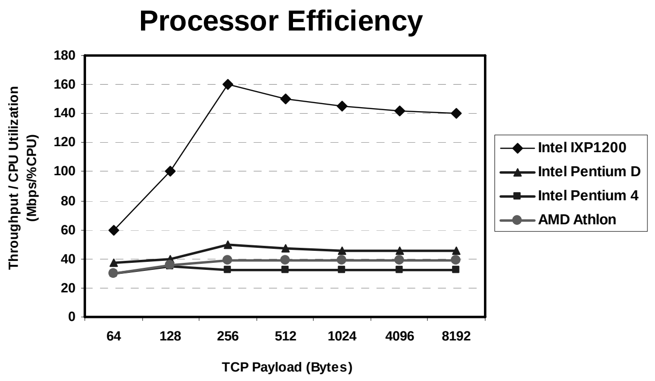

Fig.6. Throughput/ CPU utilization Vs TCP Payload

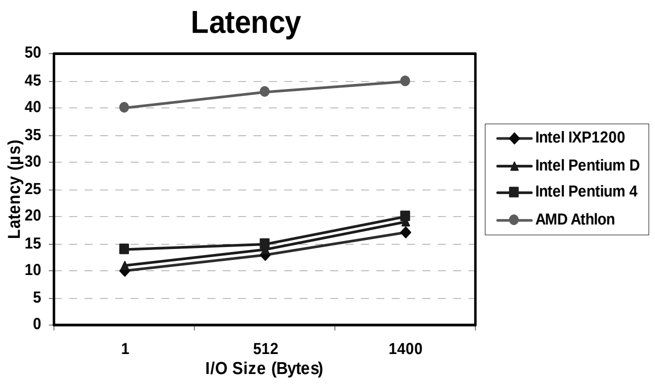

Fig.7. Latency Vs I/O Size

As shown in Fig.7, the IXP1200 had the shortest latency of 10.6µs with 1 byte I/O size, and is the best performing adapter across the studied range. Second best was the Intel Pentium D, with a 1 byte latency of 11.1µs. Intel Pentium 4 had a 1 byte latency of 13.4µs and AMD Athlon had 41.2µs. Table 1. Compares the performance of the different processors under study.

TABLE I
PERFORMANCE COMPARISON FOR TCP/IP OFFLOADING

Processor Type	**Throughput 256 bytes to 64KB (Mb/s)**	**CPU Utilization at 256 bytes to 64KB (Mbps/%CPU)**	**Latency at 1 byte I/O size (µs)**
Intel IXP1200 NP (533 MHz)	7.54	160	10.6
Intel Pentium D (2.8 GHz)	5.00	50	13.4
Intel Pentium 4 (2.8 GHz – Without HT)	3.45	32	11.1
AMD Athlon Desktop Processor (1GHz)	3.01	39	41.2

V. CONCLUSION

Implementation of TOE guarantees a decrease in load on the target CPU which in turn delivers high-speed computing. Offloading executes all phases of the TCP stack in hardware. With full offload, a TOE relieves the host not only from processing data, but also from connection management tasks. Depending on the end-user application, data path offload or full offload may be equally effective in lowering host CPU utilization and increasing the data throughput. By offloading TCP/IP protocol processing, host processor can be relieved from computing intensive protocol stacks and is free to focus its CPU cycles on the applications. TOE reduces the amount of TCP/IP processing by an average 10-15 %.

Future work would include performance analysis on other multi-core and server class processors. As the technology grows and matures, several applications for TCP/IP, TOE and Ethernet will emerge.

ACKNOWLEDGMENT

We would like to extend our sincere thanks to the reviewers for their helpful comments which improved the clarity and readability of this work.

REFERENCES

[1] M. Rangarajan, A. Bohra, K. Banerjee, E. Carrera, R. Bianchini, L. Iftode and W. Zwaenepoel. “TCP Servers: Offloading TCP Processing in Internet Servers. Design, Implementation and Performance”. Rutgers University Technical Report, DCS-TR-481, March 2002.

[2] M.Benz, “An Architecture and Prototype Implementation for TCP/IP Hardware Support”, Proc. of the TERENA Networking Conference, May 2001.

[3] Douglas E. Comer, “Network Systems Design using Network Processors: Intel IXP Version”, Prentice Hall, 2004

[4] INTEL IXP1200 Processor Family: *Hardware Reference Manual*, Intel Corporation, Portland OR., Part number 278303-008, 2001

[5] Douglas E. Comer & David L. Stevens, "Internetworking with TCP/IP, Vol. III: Client-Server Programming and Applications--BSD Socket Version, 2nd Edition Prentice Hall, 1998.

[6] P. Balaji, W. Feng, Q. Gao, R. Noronha, W. Yu, and D. K. Panda “Head-to-TOE Evaluation of High-Performance Sockets over Protocol Offload Engines” In the Proceedings of the IEEE, Sep 2005.