The Front End Processor Resurrection in a World Dominated by Gigabit Technology

Juan M. Solá-Sloan
Advisor: Isidoro Couvertier Ph.D.

Computer Information Science and Engineering
University of Puerto Rico, Mayagüez Campus
Mayagüez, Puerto Rico 00681-5000
juan_sola@yahoo.com
icouver@ece.uprm.edu

Abstract

A legacy FEP (Front End Processor) is a special computer with dedicated hardware design solely to handle communication among one or more nodes and the host. Its main job is to offload main CPU (Central Processing Unit) of processing communication specific commands. Problems of interrupt pressure, low resources availability, and the load of protocol processing are introduced by gigabit speed network technology and described in detail on this article. A new approach for using legacy FEP design goals makes their kind rise from the grave. A TOE-FEP (TCP Offload Engine Front-end Processor) is discussed as a possible solution for solving the problems impose by augmenting Ethernet speeds. Also, the advantages and disadvantages of the propose approach are discussed. Finally, this new concept, arise questions that remind open for further research.

Introduction

Gigabit speed local area networks consume main processor cycles by loading it with frame carriers to handle [Broadcom01]. This results in too many datagrams waiting for processing in a loaded CPU that is handling all the layers beyond the data-link layer. Remember that IP stacks are handle by software coded inside the operating system. Overall datagram handling degrades performance. By offloading, the CPU load is reduced leaving more processing time for applications and socket management. Legacy FEPs were machines dedicated to handle communication between a node and a host. FEP design concepts could be applied today on gigabit technology for building a TOE-FEP.

Current Gigabit technology is faster than the central processing units used on high-end transaction servers today. A rough estimate of the CPU required to handle a given Ethernet link speed is, for every one bit per second of network data processed, one hertz of CPU processing is required. Therefore, for a 10Gbps network device a 10Ghz CPU is needed for half duplex communication a 20Ghz CPU is needed for full duplex [Yeh02]. In both of the above cases the CPU needs to work at full-utilization. On current technology, this problem leads to an increase in interrupts generated by the net device.

Gigabit Speeds and Interrupt Pressure

Interrupt pressure is defined in [Gilfeather01] and presented as a mayor problem for gigabit speed networks. The inter-arrival time t of one packet, when the network is used at maximum, is calculated by the following formula:

$$8pr = t$$

In this formula p is the payload size of the frame carrier and r is the speed of the net device. Therefore for an orthodox 10Mbps connection the value for t is:

$$8 \times 1500 \times \frac{1seg}{10Mbits} = 1200m\sec$$

Projecting this amount to gigabit speeds the inter-arrival time for 1Gbps = 12µsec and for 10Gbps = 1.2 µsec (see table 1). Therefore an interrupt is generated every 1.2 microseconds. This overwhelms mainstream market CPUs.

r= network speed	t= inter-arrival time microseconds
10Mbps	1200
100Mbps	120
1Gbps	12
10Gbps	1.2

Table 1 : Projecting inter-arrival time

Payload Size Issue

If every 1.2 microseconds the net device issues an interrupt, then what will happen if we increase the payload size? This issue makes attractive the idea of expanding the orthodox payload size for incrementing the inter-arrival time. Even with jumbo frames of 9000 octets the equation increments the 1.2µsec to 7.2µsec. The attractiveness of the jumbo frame is contrasted with the reality of datagram fragmentation.

If the interrupt amount sky rockets when the net device is fast, then more processing time is required for every payload received by the data link layer. High bandwidth networks with high payload capacity, would receive fragmented datagrams wasting the payload size of the frame carrier, if there exists a fragmentation point between the end-to-end communications. As examine in [Couvertier02] fragmentation could occur anywhere in the path from node A to node B. We know that reassembling datagrams at the ultimate destination can lead to inefficiency; even if some of the physical networks encountered after the point of fragmentation have large MTU capability, only small fragments traverse them [Comer01]. These datagrams are handled by software, and we are assuming that the final destination is a single processor machine. In a typical single processor gateway, there is no way to handle the computational overhead with parallelism [Kent87]. Also, if datagrams are fragmented along the way, they could incur in network congestion. As said in [Couvertier02] there are two main options for handling this situation: Routers between A to B must have an equivalent frame carrier structure that produce a 1 to 1 packet relationship, or datagram re-assembly offloaded at the receiving end. So increasing the payload size will only generate more fragments from any datagram transmitted.

Fragmentation and PMTU

In 1987, Kent and Mogul proposed in [Kent Mogul 87] PMTU (Path Maximum Transmission Unit) discovery. PMTU finds the minimum MTU along the path from node *A* to node *B*. Datagrams are then delimited to the value found by the PMTU discovery mechanism. Contemporary operating systems include PMTU mechanisms as part of their implementation [Winguides00]. This method avoids the issues incurred by fragmentation and re-assembly of datagrams.

In [Solá03] a real connection was monitored and the packets received were analyzed. No fragmented datagrams were received. The don't fragment flag was turn on in the majority of the datagrams received. Therefore, we cannot assume that small packets are datagram fragments. Many of them are native small packets delimited by the PMTU discovery mechanism at the sender.

PMTU mechanisms attack the problem of dealing with fragment loss but does not solved the problem of generating an interrupt at the ultimate destination. Fragments and native small packets will generate interrupts at the ultimate destination that will not use the jumbo gigabit Ethernet payload at maximum. Nevertheless, interrupts will be generated according to the value found by the PMTU mechanism at the sender.

The TCP/IP Offload Engine

The 10GEA (Ten Gigabit Ethernet Alliance) propose in [Yeh02] that have been adopted by the industry as the TOE (TCP Offload Engine). This is an alternative way to treat the problems imposed in TCP/IP processing by increasing the speed of the frame carrier. A TOE is an entity outside the main CPU that is capable of reliable TCP/IP protocol processing. Advantages of using a TOE entity are:

- CPU Interrupt reduction
- CPU time is increased for applications
- Overall performance increase
- Efficient handling of IP stacks

The TOE entity can be included in an FEP to offload main CPU of datagram handling.

Advantages of a Front-end Processor with a TCP/IP Offload Engine

The problems faced at high-speed networks and loaded CPU is low resource availability, interrupt processing and IP stack processing overhead. An FEP with embedded TOE technology will impart a final blow to these issues. In [Innovative02] a FEP is defined as a dedicated computer linked to one or

more host computers or multi-user minicomputers; performs data communications functions and serves to off-load attached computers of network processing. Therefore we can see that a FEP as a TOE solution. We will call the FEP with TOE the TOE-FEP. This TOE-FEP also, could act as a gateway that handles communication to-and-from a gigabit network and the Internet.

As proposed in [Couvertier02] the load of IP stack processing could be extracted from main CPU. In fact, current technology offloads the task of generating an interrupt every time a packet is sense on the medium. The main tasks of this proposed front-end processor is solving the problems exposed previously in the first paragraph. Error checking and error handling are tasks classically handle by old school FEPs. The TOE-FEP should manage this task efficiently.

In [Solá03] a study of the packets received by a net device is analyzed. Some hybrid networks could generate broadcast packets targeting DHCP (Dynamic Host Configuration Protocol) and NetBios. Easily, a TOE-FEP could offload main CPU on these tasks. However, in [Solá03] the computer acting as the gateway was filtering this kind of network traffic. A TOE-FEP should handle the Internet communication solely. Depending on the Intranet design, fragmentation of a datagram, DHCP and NetBios may not be an issue. Intranet communication does not need the TOE-FEP in any way. However, large Intranets may use more than one TOE-FEP when heavy traffic or fragmentation of packets could occur.

The payload size proposed by the 10GEA for gigabit Ethernet is 9000 octets. On the other hand, in [Gilfeather01] the authors propose a new size of 64,000 octets. Their simulation uses this size for achieving lower interruptions to the CPU. The TCP server approach used by [Gilfeather01] re-assembles fragmented datagrams into jumbo frames. The same approach must be used when building a TOE-FEP but at the transport layer rather than in the Internet layer. In a previous work [Solá03], datagrams sniffed from a real network show that no datagrams were fragmented at all. All of the packets received were delimited datagrams. Some PMTU mechanism delimited these packets. Therefore, the TOE-FEP must handle the interrupt coalescing process at the transport layer. It must issue interrupts when some buffer size is filled, the transmission is over or when an urgent bit is set in some datagram. To avoid fragmentation inside the Intranet, the buffer size must be equal to the payload size of the frame carrier available from the TOE-FEP to the host computer.

One of the assumptions is that the transmission between the server and the TOE-FEP is using also TCP/IP. Nevertheless, other non-mainstream protocols could be used to connect the TOE-FEP and the host computer. By using TCP, the TOE-FEP would gather fragmented or small native datagrams from the sender and will re-build them inside a new TCP packet that maximizes the MTU of the gigabit speed network.

Disadvantages of using a TOE-FEP

An orthodox FEP is a common PC dedicated solely to communication processing. In [Solá02] the authors propose a Parallel TCP/IP Offload entity that could be design following the suggested framework. A typical off-the shelf computer could not process TCP/IP in parallel. In the 70s, an FEP was built specially to handle communication. The FEP was a dedicated computer with devoted hardware to offload the main computer from message handling. We encourage our readers to analyze the framework propose in [Solá02] if a real implementation is going to be achieved.

In a hybrid network, where various transport layer protocols are used, the TOE-FEP is forced to forward packets that it is not able to handle and is destine. Policies should be established preventing or allowing other protocols to bypass the TOE-FEP if needed. Security issues could impose the policies discussed.

Applications requiring urgent processing will need immediate action from the TOE-FEP. This could be done either by flushing the TOE-FEP buffers or bypassing it. On these cases, interrupt pressure and interrupt coalescing are issues that are beyond the scope of the TOE-FEP.

Future Work

An issue that arises is that, if a TOE-FEP will interrupt the main CPU every time a buffer is filled, what is the proper buffer size that does not issue a time out in the server? The TOE-FEP could also act as a firewall? What type of datagrams the TOE-FEP should filter? Which are the security issues involved in good TOE-FEP design? Does the framework in [Solá02] is a feasible implementation? These are questions that open the door for further research.

References

[Broadcom01] *Broadcom Gigabit Adapter: Performance Testing Comparison*. 2001 [online] http://www.etestinglabs.com

[Comer01] Comer Douglas. 2001. "Computer Networks and Internets." *New Jersey: Prentice Hall*.

[Couvertier02] Couvertier Isidoro and Juan M Solá. 2002. "TCP/IP Offloading Framework for a TCP/IP Offloading Implementation". [online] *Computing Research Conference 2002. University of Puerto Rico, Mayaguez Campus.* http://mayaweb.upr.clu.edu/crc/crc2002/papers/Sola_Juan.pdf

[Gilfeather01] Gilfeather Patricia, Underwood Todd. "Fragmentation and High Performance IP". *IEEE Transactions on Communications* 0-7695-0990-8/01 2001.

[Innovative02] *Innovative Electronics & Computing*. "Glossary of Terms" [online] http://www.connectworld.net/cgi-bin/iec/02GLSF.html#FRO

[Kent87] Kent Cristopher A and Jeffrey C. Mogul. 1987. "Fragmentation Considered Harmful". [online] Palo Alto, California: Digital / Compaq, Western Research Laboratory. Available from the World Wide Web: http://research.compaq.com/wrl/techreports/abstracts/87.3.html

[Solá02] Solá Sloan, Juan M. and Isidoro Couvertier. 2002. "A Parallel TCP/IP Offloading Framework for a TCP/IP Offloading Implementation." *Grenoble, France: IP SOC Workshop 2002 Proceedings.*

[Solá03] Solá Juan M and Isidoro Couvertier. 2002. "UDP, TCP and IP Fragmentation Analysis and Its Importance in TOE Devices". *Computer Research Conference 2003* [online] http://mayaweb.upr.clu.edu/crc/crc2003/papers/JuanSola.pdf

[Yeh02] Yeh, Eric, Herman Chao, Venu Mannem, Joe Gervais and Booth Bradley. 2002. [online] "Introduction to TCP/IP Offload Engine (TOE)". *10 Gigabit Ethernet Alliance*. http://www.10gea.org

[Winguides00] "Cable Modem and DSL Speed Tweak to Increase Performance" [online] "http://www.winguides.com/registry/display.php/716