Abstract

Offloading the Transport Control Protocol/Internet Protocol (TCP/IP) is a task that relieves the main processor from having to service interrupts to the kernel for protocol handling. The first stage of this research was a frame carrier offloading simulation as an Alpha Test. The current stage is the development of a TCP/IP offloading framework using a bottom up approach. An Intelligent Network Interface Card (INIC) is also analyzed as a hardware candidate for a TCP/IP offloading implementation.

1. Theoretical Analysis

The purpose of this research project is offloading TCP/IP for a multi-homed host connected to a high bandwidth network. High traffic networks nodes with a single processor interrupt the processor each time a packet is received. Gigabit Local Area Networks consume main processor cycles by loading it with frame carriers [Davis01]. Performance is wasted on packet handling. By offloading, the CPU load is reduced leaving more CPU time for the application and socket layers.

2. Methodology

Classical offloading research projects use a top-down approach which starts at the transport layer and move to layers bellow [Zhou99]. This project will focus in a bottom-up approach studying issues at the data link layer and moving toward higher layers.

This research project is composed of 5 important components:

- A frame carrier study.
- An exhaustive study of TCP/IP and related protocols to develop a framework as a base for TCP/IP offloading.
- An exhaustive study of Ethernet, Address Resolution Protocol (ARP), Internet Protocol (IP), Internet Control Message Protocol (ICMP), Transport Control Protocol (TCP), User Datagram Protocol (UDP), and sockets inside the Linux kernel.
- A simulation and a real test using an include and test approach for future framework implementation.
- A parallelism study between object components (hardware).

In the first stage of this project we studied how Ethernet could be used for handling routing transactions inside of an INIC (Intelligent Network Interface Card). An offloading TCP/IP framework forms the current stage of this research project.

3. TCP/IP Offloaded Framework

An important stage of this research project is the development of an offloading framework for TCP/IP and related protocols. For an ideal implementation ICMP, frame carrier, and a socket interface standard must also be included. This framework will be used in this research project. The framework will help offloading all possible TCP/IP related tasks.

Offloading TCP/IP candidates include all underlying and related protocols. In Figure 1 we have a clear view of the protocols involved [Beck98]. As it was mentioned earlier, frame carrier has to be offloaded to the maximum. We will focus on TCP, IP, and ICMP assuming frame carrier offloading is maximized. ARP and UDP are also considered.
3.1 Frame carrier offloading importance.

Datagram handling occurs in software. According to [Comer00], 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 fragment traverse them.

High bandwidth networks with high MTU (Maximum Transmission Unit) capacity would receive fragmented datagrams with low MTU’s if there exists a fragmentation point. Let us examine the following: sender A sends a datagram to B. Assume both A and B has high bandwidth and a frame carrier with a payload of 9000 octets. A payload of this size would be handled easily when the main processor receives it. If there were points between the route from A to B with frame carriers of 1500 octet payloads, fragmentation would occur. Routers pad frame carriers with 0 if a frame carrier with a higher payload is encountered. Thus every 9000 octet payload is divided into 6, 1500 octet payloads. This results in 1 to 6 interrupts for handling datagrams inside the frame carrier. The following expression determines the amount of frames received by B for every 1 sent by A, assuming fragmentation, where X is the fragmentation point’s lower payload:

\[ \text{IntGen} = \text{ceil} \left( \frac{A}{X} \right) \]

The ceiling function is used because there would be packets that need padding when a fragmentation point is encountered.

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 payload relationship.
- De-fragmentation offloading at receiving end.

Also, if A has a higher bandwidth than B, during transmission, A would be source quenched. The high speed is wasted since B will tell A to send less information so that B will be able to handle it.

The frame carrier should be offloaded to the maximum. If the frame carrier is handled by software such as SLIP, PLIP and some unorthodox Ethernet approach, the performance increase of offloading TCP/IP is reduced. Frame carriers exist in a layer below those of TCP/IP.

3.2 IP offloading

The entire offloading world is composed of the essential parts of data communication: offloaded sending tasks and offloaded receiving tasks. The IP offloaded mini-world will be discussed in the following section.

Offloaded IP sending task (Figure 2) starts when it receives the higher layer protocol’s payload to be encapsulated in IP. Offloaded modules should be run for determining the datagram length, fragmentation and identification, TTL (Time To Live) and Flags, IP version classificatory, and header checksum. Further study of these modules will determine if some of these tasks could also be performed in parallel. Note that there is no discarded datagrams upon sending.

![Figure 1. Protocols involved during TCP/IP offloading.](image1)

![Figure 2. Offloaded IP Sending tasks (Ipv4)](image2)
Offloaded IP receiving tasks rely on an offloaded frame carrier. A module for version identification should be run first. Depending on which IP version is identified an Ipv4 or Ipv6 offloaded module should be run next (Figure 3). With in the IP (Figure 4) module, IP datagram header checksum should be verified. Later on, the IP packet identifier module and de-fragmentation module should handle datagram fragments. If an error exists in the datagram the packet is discarded. IP does not handle errors as it based on a best effort approach.

3.3 A good implementation of TCP/IP should include ICMP.

TCP/IP discards unreliable datagrams. In internetwork infrastructure, ICMP is responsible to deal with error reporting that TCP and IP are not designed to handle. A good implementation of TCP/IP must include ICMP. There are routers and IP dependencies that exist in an ICMP/IP world [Comer00]. ICMP will substitute the circle in Figures 3 and 4.

3.4 Ipv6 and Ipv4

New generation Ipv6 should be handled [Deering98]. TCP will remain unchanged but IP could vary in nodes running in Ipv6 or Ipv4 (Figure 5). So we could have an X/Ipv4 or X/Ipv6 datagram, where X could be any higher layer protocol. Ipv6 is an extension of Ipv4 that improves address space. In America address space is not an important issue today. Super populations in India, China, and other countries are dealing with address space problems [Sass01]. Ipv6 Isles in the World Wide Web are encapsulating their payload in Ipv4 datagrams. Ipv6 – Ipv4 transition should handle both datagrams. A good IP implementation should handle both protocols.

Ipv4 and Ipv6 handling maximizes internetwork interprocess communication. A routed datagram from an Ipv6 network to an Ipv4 or vice versa should be handled without interrupting the main processor.

3.5 Socket interface needed

In Figure 1 we could see that layer exists between the application and TCP/IP layer. Different OS implementations vary in how sockets are handled. In order to provide an interface between hardware and software a socket layer should be created. This layer is included in the framework and will be manipulated by a driver.

4. Hardware

The offloading hardware manipulator candidate is an Intelligent Network Interface Card (INIC) [Boon01][Sass01]. An INIC is capable of handling frame carriers and higher layer protocols upon programming its hardware. TCP/IP and related
protocols could be implemented. Offloaded framework coding would be carried out in an include and testing approach in the INIC firmware. In the final stages of this research project we should determine, using the include and test approach, the optimum layer for offloading.

Until now, software models are used during the first stages of this research project. The INIC is simulated using software objects [Floyd01]. During the first stage The INIC object was capable of handling Ethernet routing between multiple INICs.

4.1 Multiple hardware awareness

Parallelism must exist between INICs. If there are one or more INICs available they should be able to interconnect. There is no reason to interrupt main processor if a message should be routed from one INIC to another. If routing should be done between two network interfaces, they should do it either at the frame carrier layer or at the IP layer without interrupting the CPU.

Parallelism should exist in a MIMD (Multiple Instruction Multiple Data) architecture. There should also exist a shared memory area where an INICMap should reside. This INICMap indicates how many INICs could be accessed through the server, where they are located, and where their local memory resides, and where the I/O priority queues exists for each one.

5. Future work

IP routing studies are conducted recently for a better TCP/IP offloaded framework. Further studies about ICMP handling TCP, UDP, ICMP, and ARP will be conducted. Also an INIC candidate is also analyzed for feasibility. Simulation Data of first stage alpha test is also analyzed.

References


