Large Receive Offload implementation in Neterion 10GbE Ethernet driver

Leonid Grossman
Neterion, Inc.
leonid@neterion.com

Abstract

The benefits of TSO (Transmit Side Offload) implementation in Ethernet ASICs and device drivers are well known. TSO is a *de facto* standard in version 2.6 Linux kernel and provides a significant reduction in CPU utilization, especially with 1500 MTU frames. When a system is CPU-bound, these cycles translate into a dramatic increase in throughput. Unlike TOE (TCP Offload Engine) implementations, stateless offloads do not break the Linux stack and do not introduce either security or support issues. The benefits of stateless offloads are especially apparent at 10 Gigabit rates. TSO hardware support on a 10GbE sender uses a fraction of a single CPU to achieve full line rate, still leaving plenty of cycles for applications. On the receiver side, however, the Linux stack presently does not support an equivalent stateless offload. Receiver CPU utilization, as a consequience, becomes the bottleneck that prevents 10GbE adapters from reaching line rate with 1500 MTU. Neterion Xframe adapter, implementing a LRO (Large Receive Offload) approach, was designed to address this bottleneck and reduce TCP processing overhead on the receiver. Both design and performance results will be presented.

1 Introduction

With the introduction of 10 Gigabit Ethernet, server I/O re-entered the "fast network, slow host" scenario that occurred with both the transitions to 100Base-T and 1G Ethernet.

Specifically, 10GbE has exposed three major system bottlenecks that limit the efficiently of high-performance I/O Adapters:

- PCI-X bus bandwidth
- CPU utilization
- Memory bandwidth

Despite Moore's law and other advances in server technology, completely overcoming these bottlenecks will take time. In the interim, network developers and designers need to find reliable ways to work around these limitations.

One approach to improve system I/O performance has come through the introduction of Jumbo frames. Increasing the maximum frame size to 9600 byte reduces the number of packets a system has to process and transfer across the bus.

While Jumbo frames have became universally supported in all operating systems, they have

not been universally deployed outside of the datacenter.

As a consequence, for the foreseeable future, networks will still need some kind of offloading relief in order to process existing 1500 MTU traffic.

As occurred in previous "fast network, slow host" scenarios, the need to improve performance has triggered renewed industry interest in developing NIC (Network Interface Card) hardware assists, including stateless and stateful TCP assists, as well as the all-critical operating system support required for widespread deployment of these NIC assists.

To date, the acceptance of stateless and stateful TCP assist has varied.

Stateful TCP Offload Engines (TOE) implementations never achieved any significant market traction or OS support. Primary reasons for lack of adoption include cost, implementation complexity, lack of native OS support, security/TCO concerns, and Moores law. On the other hand, stateless assists, including checksum offload and TSO (Transmit Side Offload) have achieved universal support in all major operating systems and became a de-facto standard for high-end server NICs. TSO is especially effective for 10GbE applications since it provides a dramatic reduction in CPU utilization and supports 10Gbps line rate for normal frames on current server systems.

Unfortunately, TSO offloads the transmit-side only, and there is no similar stateless offload OS support today on the receive side. To a large degree, this negates the overall effect of implementing LSO, especially in 10GbE applications like single TCP session and back-to-back setups.

This is not surprising, since receive-side offloads are less straightforward to implement due to potential out-of-order receive and other reasons. However, there are several NIC hardware assists that have existed for some time and could be quite effective, once Linux support is in place.

For example, some of the current receive-side assists that are shipped in Neterion 10GbE NICs and can be used for receive-side stateless offload include:

- MAC, IP, and TCP IPv4 and IPv6 header separation; used for header pre-fetching and LRO (Large Receive Offload). Also improves PCI bus utilization by providing better data alignment.
- RTH (Receive Traffic Hashing), based on Jenkins Hash, and SPDM (Socket Pair Direct Match); used for LRO and RTD (Receive Traffic Distribution).
- Multiple transmit and receive queues with advanced steering criteria; used for RTD, as well as for NIC virtualization, NIC sharing, and operations on multi-core CPU architectures.
- MSI and MSI-X interrupts; used in RTD, as well as for reducing interrupt overhead
- Dynamic utilization-based and timer-based interrupt moderation schemes; used to reduce CPU utilization.

2 PCI-X bus bandwidth bottleneck

Theoretically, a PCI-X 1.0 slot is limited in throughput to 8+Gbps, with a practical TCP limit (unidirectional or bidirectional) around 7.6Gbps. PCI-X 2.0 and PCI-Express slots support unidirectional 10Gbps traffic at line rate Neterion has measured 9.96Gbps (unidirectional) with PCI-X 2.0 Xframe-II adapters.

In order to saturate the PCI bus, a high-end 10GbE NIC needs to implement an efficient DMA engine, as well as support Jumbo frames, TSO, and data alignment.

3 Memory bandwidth bottleneck

Typically, memory bandwidth is not a limitation in Opteron and Itanium systems, at least not for TCP traffic. Xeon systems, however, encounter memory bandwidth limitations before either PCI bus or CPU saturation occurs. This can be demonstrated on Xeon systems with 533Mhz FSB vs. 800Mhz FSB. In any case, memory bandwidth will increase as a bottleneck concern since advances in silicon memory architectures proceed at a much slower pace than CPU advances. Neither stateful nor stateless TCP offload addresses this problem. Upcoming RDMA over Ethernet RNIC adapters will ease memory bandwidth issues, and if RNIC technology is successful in the market, this will be one application where TOE can be deployed (most likely, without exposing the TOE as a separate interface)

4 CPU utilization bottleneck

On the transmit side, LSO and interrupt moderation provide the desired result Neterion has achieved utilization in the range of 10-15% of a single Opteron CPU in order to saturate a PCI-X 1.0 bus with TCP traffic. On the receive side, however, CPU utilization emerged as the biggest bottleneck to achieving 10GbE line rate with 1500 bytes frames. With current NICs and operating systems, using multiple processors doesnt help much because in order to support cache locality and optimize CPU utilization, a TCP session needs to be kept on the same CPU. Without achieving the cache locality, additional CPU cycles are being used in a very inefficient fashion. Moores law is often cited as a main argument against deploying TCP assists and offloads. However, the industry wants to deploy full-rate 10GbE and cannot wait for CPUs that don't required offloading. Also, from an application prospective CPU utilization expended on stack processing must drop to single digits, and on current systems, the only way to achieve such a low utilization rate for 10GbE processing is to bring in some sort of hardware assist. The resolution to the CPU bottleneck is to add Linux support for header separation and prefetching, as well as for Receive Traffic Distribution and Receive Side Offload.

5 Header separation and prefetching

Neterion's Xframe-II supports several flavors of true hardware separation of Ethernet, IP and TCP (both IPv4 and IPv6) headers. This has been proven to be effective in achieving optimal data alignment, but since cache misses on headers represent one of the most significant sources of TCP processing overhead, the real benefit is expected to come from the ability to support OS header pre-fetching and LRO.

6 Receive Traffic Distribution

The key to efficient distribution of TCP processing across multiple CPUs is maintaining an even load between processors while at the same time keeping each TCP session on the same CPU. In order to accomplish this, the host must be able to identify each TCP flow and dynamically associate the flow to its particular hardware receive queue, particular MSI, DPC,

and CPU. In this way, load-balancing multiple TCP sessions across CPUs while preserving cache locality is possible. Neterion's Xframe-II 10GbE ASIC achieves this through receive descriptors that carry SPDM or RTH information on a per packet basis, giving the host enough visibility into packets to identify and associate flows.

7 Large Receive Offload

In short, LRO assists the host in processing incoming network packets by aggregating them on-the-fly into fewer but larger packets. This is done with some hardware assist from the NIC. It's important that an LRO implementation avoid a very expensive state-aware TOE implementation that would break compatibility with current operating systems and therefore have only limited application.

To illustrate the effectiveness of LRO, consider a network passing 1500 MTU packets at a data rate of 10 Gigabit per second. In this best possible case network traffic consists of universally full-sized packets the host-resident network stack will have to process more than 800,000 packets per second. If it takes on average 2000 instructions to process each packet and one CPU clock cycle to execute each instruction, processing in the best case will take consume more than 80% of a 2Ghz CPU, leaving little for doing anything other than receiving data. This simplified calculation demonstrates the critical characteristic of networks that the performance of transport protocols is dependent upon the granularity of data packets. The fewer packets presented to the protocol stacks, the less CPU utilization required leaving more cycle for the host to run applications.

The idea of Large Receive Offload, as the name implies, is to give the host the same amount of data but in bigger "chunks." Reducing the number of packets the stacks have to process lowers the load on the CPU. LRO implementation relies on the bursty nature of TCP traffic, as well as the low packet loss rates that are typical for 10GbE datacenter applications.

To implement Large Receive Offload, Neterion's Xframe-II 10GbE ASIC separates TCP headers from the payload and calculates SPDM or RTH information on a per packet basis. In this way, it is possible to identify a burst of consecutive packets that belong to the same TCP session and can be aggregated into a single oversized packet. Additionally, the LRO engine must perform a number of checks on the packet to ensure that it can be added to an LRO frame.

The initial implementation of Neterion's LRO is a combination of hardware (NIC) and software (Linux driver). The NIC provides the following:

- multiple hardware-supported receive queues
- link-layer, IP, and UDP/TCP checksum offloading
- header and data split, with link, IP, TCP, and UDP headers placed in the host-provided buffers separately from their corresponding packet datas
- SPDM or RTH "flow identifier."

The Linux driver controls the NIC and coordinates operation with the host-resident protocol stack. It is the driver that links payloads together and builds a single header for the LRO packet. If the flow is "interrupted," such as a sequence gap, the driver signals the host-resident network stack and sends all the accumulated receive data.

The simple algorithm below capitilizes on the fact that the receive handling code at any point in time potentially "sees" multiple new frames. This is because of the interrupt coalescing, which may or may not be used in combination with polling (NAPI).

Depending on the interrupt moderation scheme configured "into" the adapter, at high throughput we are seeing batches of 10s or 100s received frames within a context of a single interrupt.

The same is true for NAPI, except that the received "batch" tends to be even bigger, and the processing is done in the `net_device->poll()` softirq context.

Within this received batch the LRO logic looks for multiple back-to-back frames that belong to the same stream.

The 12-step algorithm below is essentially a set of simple hardware-friendly checks (see check A, check B, etc. below) and a simple hardware-friendly header manipulation.

Note that by virtue of being a pseudo-code certain low-level details were simplified out.

8 Large Receive Offload algorithm

1) for each (Rx descriptor, Rx frame) pair from the received "batch":

2) get LRO object that corresponds to the descriptor->ring.

3) check A:
- should the frame be dropped? (check FCS and a number of other conditions, including ECC) if the frame is bad then drop it, increment the stats, and continue to 1).

4) is the LRO object (located at step 2) empty? if it contains previously accumulated data, goto step 6); otherwise proceed with a series of checks on the first to-be-LROed frame (next).

5) check B:
- is it IP frame?
- IP fragmented?
- passes a check for IP options?
- either TCP or UDP?
- both L3 and L4 offloaded checksums are good?
- for TCP: passes a check for flags?
- for TCP: passes a check for TCP options? if any check fails - goto step 11). otherwise goto to step 10).

6) use hardware-assisted Receive Traffic Hashing (RTH) to check whether the frame belongs to the same stream; if not (i.e. cannot be LROed), goto 11).

7) check C:
- IP fragmented?
- passes a check for IP options?
- offloaded checksums are good?
- for TCP: passes a check for flags?
- for TCP: passes a check for TCP options? if any of the checks fail, goto step 11).

8) check D:
- in-order TCP segment? if not, goto step 11).

9) append the new (the current) frame; update the header of the first frame in the already LROed sequence; update LRO state (for the given ring->LRO) accordingly.

10) check E:
- too much LRO data accumulated? (in terms of both total size and number of "fragments")
- is it the last frame in this received "batch"? if 'no' on both checks, continue to 1).

11) call netif_rx() or netif_receive_skb() (the latter for NAPI) for the LRO-ed frame, if exists; call netif_rx() or netif_receive_skb() for

the current frame, if not “appended” within this iteration (at step 9).

12) reset the LRO object and continue to 1).

9 Conclusion

Stateless hardware assists and TCP offloads have become a de-facto standard feature in both high-end Server NICs and operating systems. Support for additional stateless offloads on the receive-side, with native driver support in Linux, is required in order to provide 10Gbps Ethernet data rates in efficient manner.

10 References

- Xframe 10GbE Programming manual
- The latest Neterion Linux driver code (available in 2.6 kernel)

Proceedings of the Linux Symposium

Volume One

July 20nd–23th, 2005
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc.*
C. Craig Ross, *Linux Symposium*
Stephanie Donovan, *Linux Symposium*

Review Committee

Gerrit Huizenga, *IBM*
Matthew Wilcox, *HP*
Dirk Hohndel, *Intel*
Val Henson, *Sun Microsystems*
Jamal Hadi Salimi, *Znyx*
Matt Domsch, *Dell*
Andrew Hutton, *Steamballoon, Inc.*

Proceedings Formatting Team

John W. Lockhart, *Red Hat, Inc.*