# Introduction to TCP Offload Engines

By implementing a TCP Offload Engine (TOE) in high-speed computing environments, administrators can help relieve network bottlenecks and improve application performance. This article introduces the concept of TOEs, explains how TOEs interact with the TCP/IP stack, and discusses potential performance advantages of implementing TOEs on specialized network adapters versus processing the standard TCP/IP protocol suite on the host CPU.

BY SANDHYA SENAPATHI AND RICH HERNANDEZ

As network interconnect speeds advance to Gigabit Ethernet[1] and 10 Gigabit Ethernet,[2] host processors can become a bottleneck in high-speed computing—often requiring more CPU cycles to process the TCP/IP protocol stack than the business-critical applications they are running. As network speed increases, so does the performance degradation incurred by the corresponding increase in TCP/IP overhead. The performance degradation problem can be particularly severe in Internet SCSI (iSCSI)–based applications, which use IP to transfer storage block I/O data over the network.

By carrying SCSI commands over IP networks, iSCSI facilitates both intranet data transfers and long-distance storage management. To improve data-transfer performance over IP networks, the TCP Offload Engine (TOE) model can relieve the host CPU from the overhead of processing TCP/IP. TOEs allow the operating system (OS) to move all TCP/IP traffic to specialized hardware on the network adapter while leaving TCP/IP control decisions to the host server. By relieving the host processor bottleneck, TOEs can help deliver the performance benefits administrators expect from iSCSI-based applications running across high-speed network links. By facilitating file I/O traffic, TOEs also can improve the performance of network attached storage (NAS). Moreover, TOEs are cost-effective because they can process the TCP/IP protocol stack on a high-speed network device that requires less processing power than a high-performance host CPU.

This article provides a high-level overview of the advantages and inherent drawbacks to TCP/IP, explaining existing mechanisms to overcome the limitations of this protocol suite. In addition, TOE-based implementations and potential performance benefits of using TOEs instead of standard TCP/IP are described.

## TCP/IP helps ensure reliable, in-order data delivery

Currently the de facto standard for internetwork data transmission, the TCP/IP protocol suite is used to transmit information over local area networks (LANs), wide area networks (WANs), and the Internet. TCP/IP

[1] This term does not connote an actual operating speed of 1 Gbps. For high-speed transmission, connection to a Gigabit Ethernet server and network infrastructure is required.

[2] This term does not connote an actual operating speed of 10 Gbps. For high-speed transmission, connection to a 10 Gigabit Ethernet (10GbE) server and network infrastructure is required.

processes can be conceptualized as layers in a hierarchical stack; each layer builds upon the layer below it, providing additional functionality. The layers most relevant to TOEs are the IP layer and the TCP layer (see Figure 1).

As network speed increases, so does the performance degradation incurred by the corresponding increase in TCP/IP overhead.

The IP layer serves two purposes: the first is to transmit packets between LANs and WANs through the *routing* process; the second is to maintain a homogeneous interface to different physical networks. IP is a connectionless protocol, meaning that each transmitted packet is treated as a separate entity. In reality, each network packet belongs to a certain data stream, and each data stream belongs to a particular host application. The TCP layer associates each network packet with the appropriate data stream and, in turn, the upper layers associate each data stream with its designated host application.

Most Internet protocols, including FTP and HTTP, use TCP to transfer data. TCP is a connection-oriented protocol, meaning that two host systems must establish a session with each other before any data can be transferred between them. Whereas IP does not provide for error recovery—that is, IP has the potential to lose packets, duplicate packets, delay packets, or deliver packets out of sequence—TCP ensures that the host system receives all packets in order, without duplication. Because most Internet applications require reliable data that is delivered in order and in manageable quantities, TCP is a crucial element of the network protocol stack for WANs and LANs.

**Reliability.** TCP uses the *checksum* error-detection scheme, which computes the number of set bits on packet headers as well as packet data to ensure that packets have not been corrupted during transmission. A TCP pseudoheader is included in the checksum computation to verify the IP source and destination addresses.

**In-order data delivery.** Because packets that belong to a single TCP connection can arrive at the destination system via different routes, TCP incorporates a per-byte numbering mechanism. This scheme enables the TCP protocol to put packets that arrive at their destination out of sequence back into the order in which they were sent, before it delivers the packets to the host application.

**Flow control.** TCP monitors the number of bytes that the source system can transmit without overwhelming the destination system with data. As the source system sends packets, the receiving system returns acknowledgments. TCP incorporates a *sliding window* mechanism to control congestion on the receiving end. That is, as the sender transmits packets to the receiver, the size of the window reduces; as the sender receives acknowledgments from the receiver, the size of the window increases.

**Multiplexing.** TCP accommodates the flow from multiple senders by allowing different data streams to intermingle during transmission and receiving. It identifies individual data streams with a number called the TCP port, which associates each stream with its designated host application on the receiving end.

## Traditional methods to reduce TCP/IP overhead offer limited gains

After an application sends data across a network, several data-movement and protocol-processing steps occur. These and other TCP activities consume critical host resources:

- The application writes the transmit data to the TCP/IP sockets interface for transmission in payload sizes ranging from 4 KB to 64 KB.
- The OS segments the data into maximum transmission unit (MTU)–size packets, and then adds TCP/IP header information to each packet.
- The OS copies the data onto the network interface card (NIC) send queue.
- The NIC performs the direct memory access (DMA) transfer of each data packet from the TCP buffer space to the NIC, and interrupts CPU activities to indicate completion of the transfer.

The two most popular methods to reduce the substantial CPU overhead that TCP/IP processing incurs are TCP/IP checksum offload and large send offload.

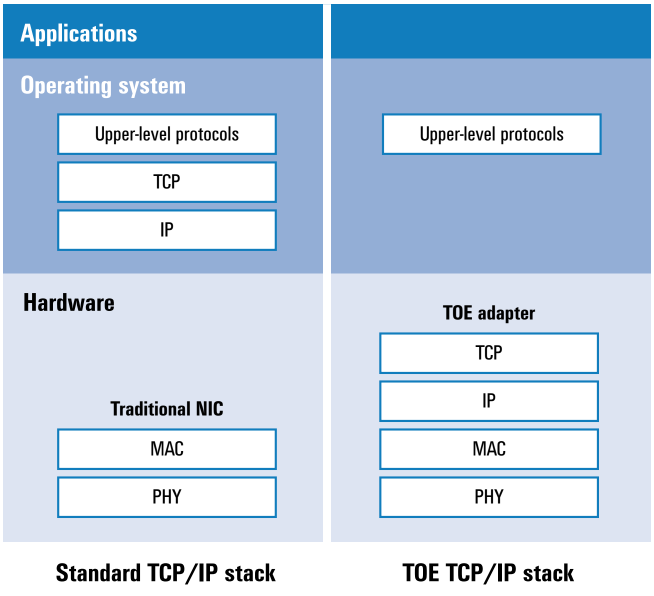

Figure 1. Comparing standard TCP/IP and TOE-enabled TCP/IP stacks

### TCP/IP checksum offload

The TCP/IP checksum offload technique moves the calculation of the TCP and IP checksum packets from the host CPU to the network adapter. For the TCP checksum, the transport layer on the host calculates the TCP pseudoheader checksum and places this value in the checksum field, thus enabling the network adapter to calculate the correct TCP checksum without touching the IP header. However, this approach yields only a modest reduction in CPU utilization.

### Large send offload

Large send offload (LSO), also known as TCP segmentation offload (TSO), frees the OS from the task of segmenting the application's transmit data into MTU-size chunks. Using LSO, TCP can transmit a chunk of data larger than the MTU size to the network adapter. The adapter driver then divides the data into MTU-size chunks and uses the prototype TCP and IP headers of the send buffer to create TCP/IP headers for each packet in preparation for transmission.

LSO is an extremely useful technology to scale performance across multiple Gigabit Ethernet links, although it does so under certain conditions. The LSO technique is most efficient when transferring large messages. Also, because LSO is a stateless offload, it yields performance benefits only for traffic being sent; it offers no improvements for traffic being received. Although LSO can reduce CPU utilization by approximately half, this benefit can be realized only if the receiver's TCP window size is set to 64 KB. LSO has little effect on interrupt processing because it is a transmit-only offload.

Methods such as TCP/IP checksum offload and LSO provide limited performance gains or are advantageous only under certain conditions. For example, LSO is less effective when transmitting several smaller-sized packages. Also, in environments where packets are frequently dropped and connections lost, connection setup and maintenance consume a significant proportion of the host's processing power. Methods like LSO would produce minimal performance improvements in such environments.

## TOEs reduce TCP overhead on the host processor

In traditional TCP/IP implementations, every network transaction results in a series of host interrupts for various processes related to transmitting and receiving, such as send-packet segmentation and receive-packet processing. Alternatively, TOEs can delegate all processing related to sending and receiving packets to the network adapter—leaving the host server's CPU more available for business applications. Because TOEs involve the host processor only once for every application network I/O, they significantly reduce the number of requests and acknowledgments that the host stack must process.

Using traditional TCP/IP, the host server must process received packets and associate received packets with TCP connections, which means every received packet goes through multiple data copies from system buffers to user memory locations. Because a TOE-enabled network adapter can perform all protocol processing, the adapter can use zero-copy algorithms to copy data directly from the NIC buffers into application memory locations, without intermediate copies to system buffers. In this way, TOEs greatly reduce the three main causes of TCP/IP overhead—CPU interrupt processing, memory copies, and protocol processing.

### CPU interrupt processing

An application that generates a write to a remote host over a network produces a series of interrupts to segment the data into packets and process the incoming acknowledgments. Handling each interrupt creates a significant amount of context switching—a type of multitasking that directs the focus of the host CPU from one process to another—in this case, from the current application process to the OS kernel and back again. Although interrupt-processing aggregation techniques can help reduce the overhead, they do not reduce the event processing required to send packets. Additionally, every data transfer generates a series of data copies from the application data buffers to the system buffers, and from the system buffers to the network adapters.

High-speed networks such as Gigabit Ethernet compel host CPUs to keep up with a larger number of packets. For 1500-byte packets, the host OS stack would need to process more than 83,000 packets per second, or a packet every 12 microseconds. Smaller packets put an even greater burden on the host CPU. TOE processing can enable a dramatic reduction in network transaction load. Using TOEs, the host CPU can process an entire application I/O transaction with one interrupt. Therefore, applications working with data sizes that are multiples of network packet sizes will benefit the most from TOEs. CPU interrupt processing can be reduced from thousands of interrupts to one or two per I/O transaction.

### Memory copies

Standard NICs require that data be copied from the application user space to the OS kernel. The NIC driver then can copy the data from the kernel to the on-board packet buffer. This requires multiple trips across the memory bus (see Figure 2): When packets are received from the network, the NIC copies the packets to the NIC buffers, which reside in host memory. Packets then are copied to the TCP buffer and, finally, to the application itself—a total of three memory copies.

A TOE-enabled NIC can reduce the number of buffer copies to two: The NIC copies

To improve data-transfer performance over IP networks, the TCP Offload Engine model can relieve the host CPU from the overhead of processing TCP/IP.

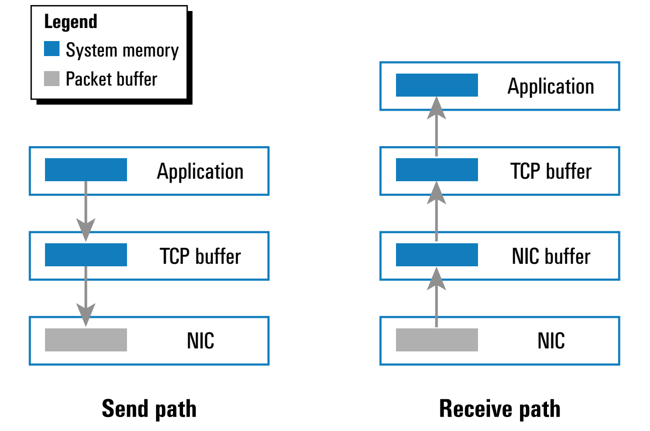

Figure 2. Transmitting data across the memory bus using a standard NIC

the packets to the TCP buffer and then to the application buffers. A TOE-enabled NIC using Remote Direct Memory Access (RDMA) can use zero-copy algorithms to place data directly into application buffers.

The capability of RDMA to place data directly eliminates intermediate memory buffering and copying, as well as the associated demands on the memory and processor resources of the host server—without requiring the addition of expensive buffer memory on the Ethernet adapter. RDMA also preserves memory-protection semantics. The RDMA over TCP/IP specification defines the interoperable protocols to support RDMA operations over standard TCP/IP networks.

### Protocol processing

The traditional host OS–resident stack must handle a large number of requests to process an application's 64 KB data block send request. Acknowledgments for the transmitted data also must be received and processed by the host stack. In addition, TCP is required to maintain state information for every data connection created. This state information includes data such as the current size and position of the windows for both sender and receiver. Every time a packet is received or sent, the position and size of the window change and TCP must record these changes.

Protocol processing consumes more CPU power to receive packets than to send packets. A standard NIC must buffer received packets, and then notify the host system using interrupts. After a context switch to handle the interrupt, the host system processes the packet information so that the packets can be associated with an open TCP connection. Next, the TCP data must be correlated with the associated application and then the TCP data must be copied from system buffers into the application memory locations.

TCP uses the checksum information in every packet that the IP layer sends to determine whether the packet is error-free. TCP also records an acknowledgment for every received packet. Each of these operations results in an interrupt call to the underlying OS. As a result, the host CPU can be saturated by frequent interrupts and protocol processing overhead. The faster the network, the more protocol processing the CPU has to perform.

In general, 1 MHz of CPU power is required to transmit 1 Mbps of data. To process 100 Mbps of data—the speed at which Fast Ethernet operates—100 MHz of CPU computing power is required, which today's CPUs can handle without difficulty. However, bottlenecks can begin to occur when administrators introduce Gigabit Ethernet and 10 Gigabit Ethernet. At these network speeds, with so much CPU power devoted to TCP processing, relatively few cycles are available for application processing. Multi-homed hosts with multiple Gigabit Ethernet NICs compound the problem. Throughput does not scale linearly when utilizing multiple NICs in the same server because only one host TCP/IP stack processes all the traffic. In comparison, TOEs distribute network transaction processing across multiple TOE-enabled network adapters.

## TOEs provide options for optimal performance or flexibility

Administrators can implement TOEs in several ways, as best suits performance and flexibility requirements. Both processor-based and chip-based methods exist. The processor-based approach provides the flexibility to add new features and use widely available components, while chip-based techniques offer excellent performance at a low cost. In addition, some TOE implementations offload processing completely while others do so partially.

### Processor-based versus chip-based implementations

The implementation of TOEs in a standardized manner requires two components: network adapters that can handle TCP/IP processing operations, and extensions to the TCP/IP software stack that offload specified operations to the network adapter. Together, these components let the OS move all TCP/IP traffic to specialized, TOE-enabled firmware—designed into a TCP/IP-capable NIC—while leaving TCP/IP control decisions with the host system. Processor-based methods also can use off-the-shelf network adapters that have a built-in processor and memory. However, processor-based methods are more expensive and still can create bottlenecks at 10 Gbps and beyond.

The second component of a standardized TOE implementation comprises TOE extensions to the TCP/IP stack, which are completely transparent to the higher-layer protocols and applications that run on top of them. Applications interact the same way with a TOE-enabled stack as they would with a standard TCP/IP stack. This transparency makes the TOE approach attractive because it requires no changes to the numerous applications and higher-level protocols that already use TCP/IP as a base for network transmission.

The chip-based implementation uses an application-specific integrated circuit (ASIC) that is designed into the network adapter. ASIC-based implementations can offer better performance than off-the-shelf processor-based implementations because they are customized to perform the TCP offload. However, because ASICs are manufactured for a certain set of operations, adding new features may not always be possible. To offload specified operations to the network adapter, ASIC-based implementations require the same extensions to the TCP/IP software stack as processor-based implementations.

By relieving the host processor bottleneck, TOEs can help deliver the performance benefits administrators expect from iSCSI-based applications running across high-speed network links.

### Partial versus full offloading

TOE implementations also can be differentiated by the amount of processing that is offloaded to the network adapter. In situations where TCP connections are stable and packet drops infrequent, the highest amount of TCP processing is spent in data transmission and reception. Offloading just the processing related to transmission and reception is referred to as partial offloading. A partial, or *data path*, TOE implementation eliminates the host CPU overhead created by transmission and reception.

However, the partial offloading method improves performance only in situations where TCP connections are created and held for a long time and errors and lost packets are infrequent. Partial offloading relies on the host stack to handle control—that is, connection setup—as well as exceptions. A partial TOE implementation does not handle the following:

- TCP connection setup
- Fragmented TCP segments
- Retransmission time-out
- Out-of-order segments

The host software uses dynamic and flexible algorithms to determine which connections to offload. This functionality requires an OS extension to enable hooks that bypass the normal stack and implement the offload heuristics. The system software has better information than the TOE-enabled NIC regarding the type of traffic it is handling, and thus makes offload decisions based on priority, protocol type, and level of activity. In addition, the host software is responsible for preventing denial of service (DoS) attacks. When administrators discover new attacks, they should upgrade the host software as required to handle the attack for both offloaded and non-offloaded connections.

TCP/IP fragmentation is a rare event on today's networks and should not occur if applications and network components are working properly. A store-and-forward NIC saves all out-of-order packets in on-chip or external RAM so that it can reorder the packets before sending the data to the application. Therefore, a partial TOE implementation should not cause performance degradation because, given that current networks are reliable, TCP operates most of the time without experiencing exceptions.

The process of offloading all the components of the TCP stack is called full offloading. With a full offload, the system is relieved not only of TCP data processing, but also of connection-management tasks. Full offloading may prove more advantageous than partial offloading in TCP connections characterized by frequent errors and lost connections.

## TOEs reduce end-to-end latency

A TOE-enabled TCP/IP stack and NIC can help relieve network bottlenecks and improve data-transfer performance by eliminating much of the host processing overhead that the standard TCP/IP stack incurs. By reducing the amount of time the host system spends processing network transactions, administrators can increase available bandwidth for business applications. For instance, in a scenario in which an application server is connected to a backup server, and TOE-enabled NICs are installed on both systems, the TOE approach can significantly reduce backup times.

Reduction in the time spent processing packet transmissions also reduces latency—the time taken by a TCP/IP packet to travel from the source system to the destination system. By improving end-to-end latency, TOEs can help speed response times for applications including digital media serving, NAS file serving, iSCSI, Web and e-mail serving, video streaming, medical imaging, LAN-based backup and restore processes, and high-performance computing clusters.

Part two of this article, which will appear in an upcoming issue, will include benchmark testing and analysis to measure the benefits of TOEs.

**Sandhya Senapathi** (sandhya_senapathi@dell.com) is a systems engineer with the Server OS Engineering team. Her fields of interest include operating systems, networking, and computer architecture. She has an M.S. in Computer Science from The Ohio State University.

**Rich Hernandez** (rich_hernandez@dell.com) is a technologist with the Dell Product Group. He has been in the computer and data networking industry for more than 19 years. Rich has a B.S. in Electrical Engineering from the University of Houston and has pursued postgraduate studies at Colorado Technical University.

FOR MORE INFORMATION

RDMA Consortium:
http://www.rdmaconsortium.org/home