Application Scaling for Cloud Computing

Using multicore processors to build intelligent application delivery switches for next-generation data centers

Large data centers such as Google, Yahoo, and Salesforce.com use application delivery switches to load balance incoming network traffic. This traffic comes from multi-Gigabit data streams to more than 100 application servers running hundreds of virtual machines. Handling this traffic demands a very high-performance, intelligent switch capable of processing thousands of compute-intensive Transmission Control Protocol (TCP) and Secure Sockets Layer (SSL) operations per second.

The strategy employed by many application delivery switch providers like F5, Cisco, ACE, and Citrix to solve this processing throughput problem leverages the performance advantages of embedded multicore processors with hardware acceleration. This case study describes how TCP and SSL can be successfully implemented on a multicore processor to provide scalable cloud computing.

The Processor Performance Challenge

Among the most compute-intensive tasks application delivery switches must perform are TCP and SSL. TCP enables the end-to-end transport of data between the user (browser) and the website (web server). Its primary tasks are to establish a connection between the two end points, transfer data in an orderly manner, and terminate the session when complete. In addition, TCP manages the size of the data segment, the flow of data, and the speed at which data is exchanged, as well as any network traffic bottlenecks.

Another key technology most application delivery switches must contend with is Transport Layer Security (TLS), or the older Secure Sockets Layer (SSL). TLS/SSL is used by HTTPS to allow secure, private transmission of data, such as credit card or bank account numbers, or other personal information over the Web. A secure, https site is identified by the yellow “lock” icon in the upper left hand corner of the browser.

Processing TCP requests and establishing secure sessions put a significant demand on processors. A high-level overview of how TCP and TLS work will give the reader an appreciation for the brute processing force that application delivery switches need to perform these tasks and maintain an acceptable level of service.

Inside TCP

TCP protocol operations are divided into three phases: *Connection Establishment* phase where connections are established in a multi-step handshake process; *Data Transfer* phase; and *Connection Termination* phase which closes established virtual circuits and releases all allocated resources after data transmission is finished.

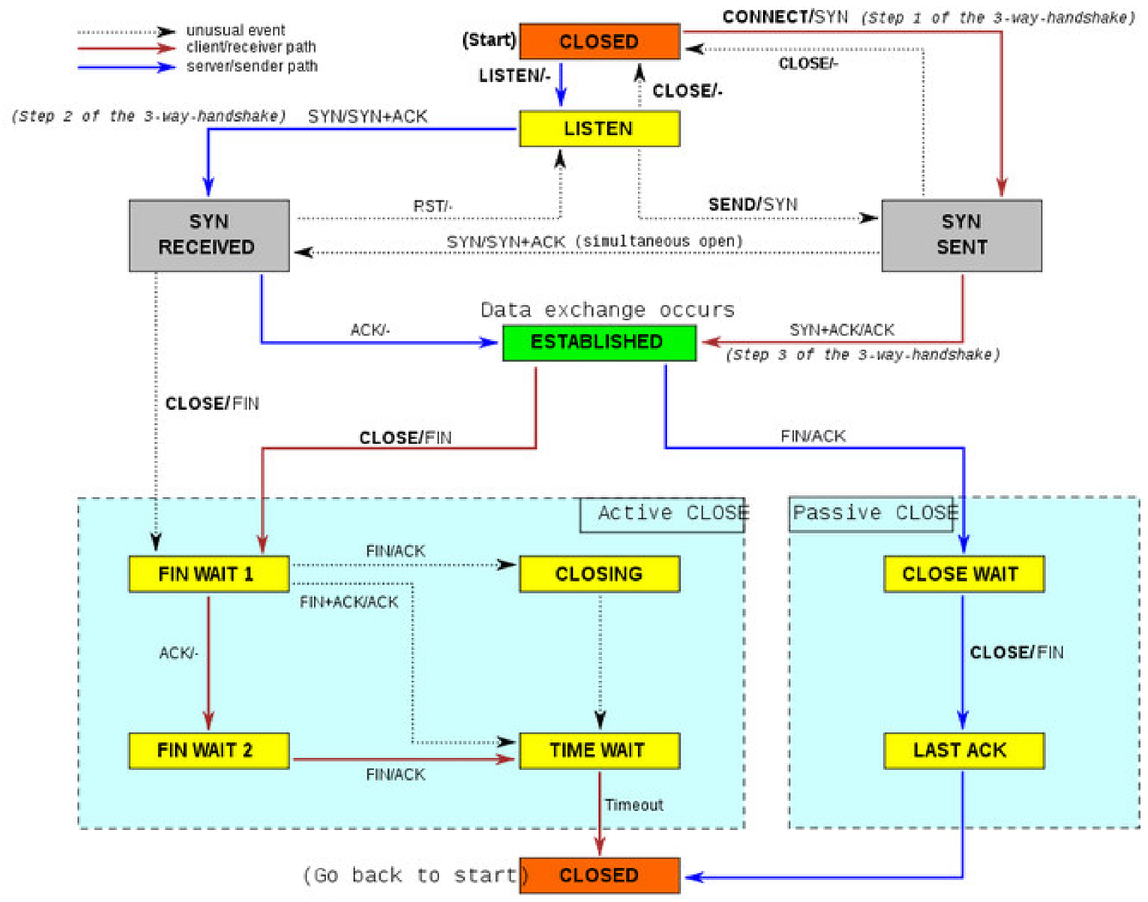


Fig. 1 For a more detailed description of TCP, visit http://en.wikipedia.org/wiki/File:Tcp_state_diagram_fixed.svg

In the connection phase, the server first binds to a port to open it up for a connection. This is called a passive open. Once established, the client can initiate an active open by sending a SYN to the server and setting the segment's sequence number to a random value. The server then replies with a SYN-ACK and the acknowledgement number is reset to one greater than the received sequence number. Upon receipt of the server's SYN-ACK, the client sends an ACK back to the server. The sequence number is reset to the received acknowledgement value and the acknowledgement number is reset to one more than the received sequence number. Now acknowledgements have been received by both parties and data can be transferred over the connection.

To ensure reliable transmission, TCP employs a sequence number to identify every byte of data and its order. This allows the data to be reconstructed in the correct order even if it becomes disordered, fragmented, or some packets are lost. If packets are lost, TCP can request the retransmission of packets. In order to prevent the sender from sending data too fast for the TCP receiver to reliably receive and process it, TCP uses flow control protocol.

Finally, each side of the connection terminates its side independently by transmitting a FIN packet, which the other end acknowledges with an ACK.

Inside TLS

At a high level, the SSL/TLS server and client agree on the version of SSL protocol to employ, select cryptographic algorithms, authenticate each other by exchanging and validating digital certificates, create a shared secret key, and then use that key for the symmetric encryption of messages between the two. TLS requires even more processing power than TCP, particularly for the key exchange operation, as described below.

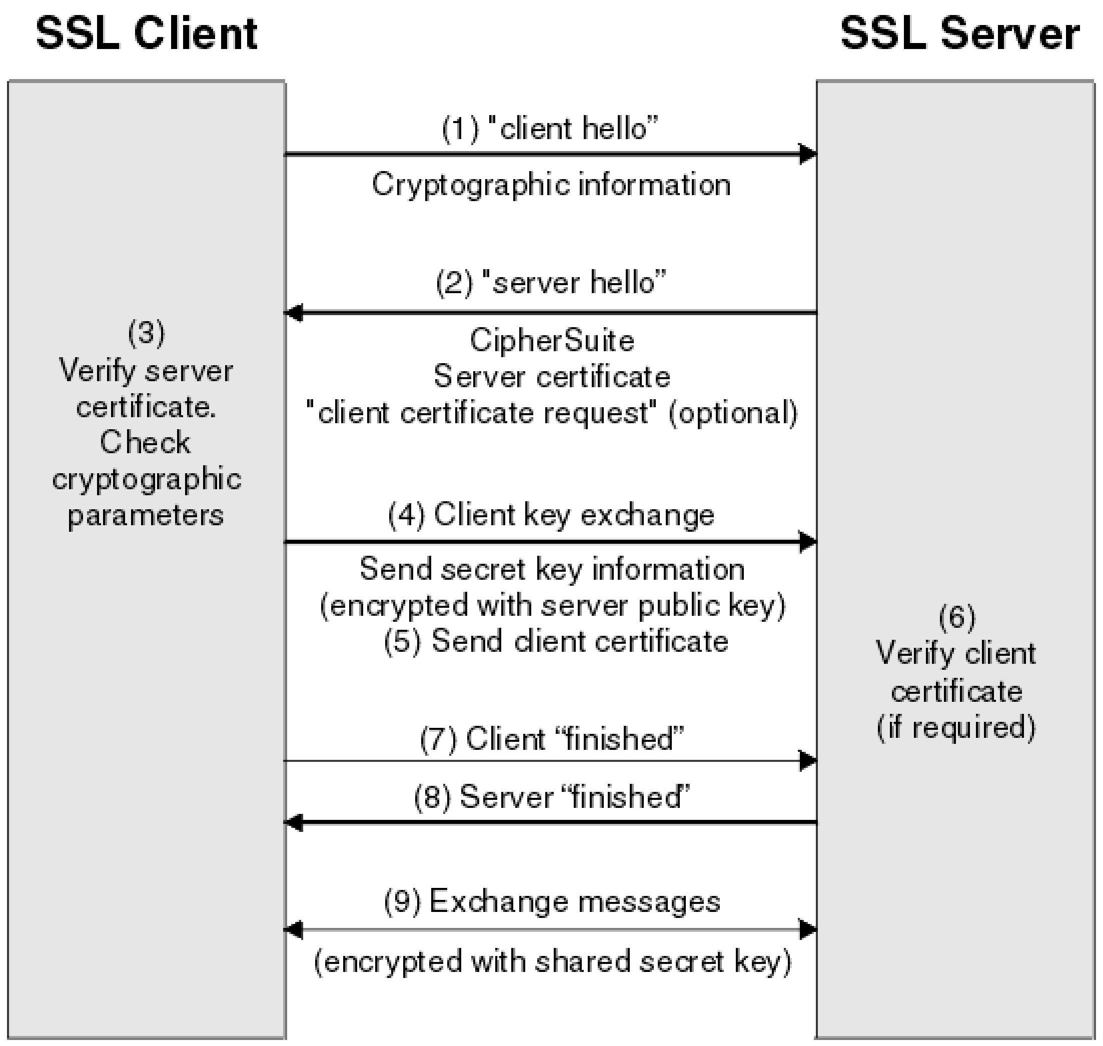

SSL Client
SSL Server
(1) "client hello"
Cryptographic information
(2) "server hello"
CipherSuite
Server certificate
"client certificate request" (optional)
(3) Verify server certificate. Check cryptographic parameters
(4) Client key exchange
Send secret key information (encrypted with server public key)
(5) Send client certificate
(6) Verify client certificate (if required)
(7) Client "finished"
(8) Server "finished"
(9) Exchange messages
(encrypted with shared secret key)

Fig. 1 Flow chart of SSL handshake

An SSL/TLS session begins with an exchange of messages called the SSL/TLS handshake. The handshake enables the server to authenticate itself to the client by using public-key techniques, and then allows the client and the server to cooperate in the creation of symmetric keys used for rapid encryption, decryption, and tamper detection during the session that follows. SSL protocol uses both public- and symmetric-key encryption. Symmetric-key encryption is significantly faster than public-key encryption; however, public-key encryption provides better authentication techniques.

The key exchange operation within SSL/TLS relies on asymmetric encryption techniques (commonly Diffie-Hellman or RSA) to generate a shared secret key, which avoids the key distribution problem. The SSL client sends a random byte string that enables both the client and the server to compute the secret key to be used for encrypting subsequent message data. The random byte string is also encrypted with the server's public key. This key exchange operation is the most processor-intensive step within SSL—made more intensive with hundreds or thousands of key exchanges being established simultaneously within a large eCommerce site.

Once the symmetric keys have been established, bulk transfer of encrypted data can be performed securely. The encryption operation, like the key exchange operation, is a processing-intensive process that can impact performance.

Strategy for Handling TCP and TLS in an Application Delivery Switch

In the past, developers of application delivery switches relied on single-core processors. More recently, it became clear to developers that the single-core approach lacked the scalability needed to meet increasing processor demands.

The path forward for application delivery switches was multicore processors (i.e., multiple cores on a single processor) that could divide compute tasks across many cores. Multicores have two significant advantages over a single core. The first is higher total processing power at lower frequencies. At one execution step per cycle, for example, a single-core processor operating at 2 GHz is capable of processing 2 billion steps per second. But a 16-core processor operating at just 750 MHz on each core (a fraction of the single processor) can process 12 billion steps per second—while using less power and generating less heat.

The other advantage multicores have is the ability to parallelize tasks. Application delivery switches manage many TCP flows and TLS/SSL sessions simultaneously, so there is inherent parallelism. Each core can be assigned to process a flow or session and multiple cores can process multiple sessions/flows in parallel.

Limitations of Multicore

Many multicore implementations today utilize a general purpose processor with two or four cores. A simultaneous multi-processing operating system (SMP OS) manages and schedules processing among the cores. Software is used to partition the data into threads. The SMP OS then schedules the execution of these threads. Shared data structures and serialization among the threads are handled through semaphores or through some software locking mechanisms, such as spin-locks.

Migrating from a single core to two or four cores does increase performance; however, the marginal increase quickly drops as the number of cores increases. A plot of overall performance against the number of cores utilized reveals that performance plateaus and then even decreases as the core count increases further.

One of the reasons for this performance limitation has to do with scheduling. SMP systems rely on the OS to schedule all the execution. The overhead of scheduling and managing all the program threads and processes increases as more cores and more threads are scheduled.

In addition, SMP OS preemptively swaps out execution threads and swaps in other threads, so that all the threads make forward progress based on the set of priorities managed by the SMP OS. On a multicore system, the SMP OS swaps threads in and out with all the cores as execution resources. For example, a thread can be executing on core x for some number of cycles, get swapped out, and later resume execution on core y. When this thread was executing on core x, a sizable portion of its working data set was cached within the core x cache. When it was swapped out and later resumed execution at core y, the related cache content in core x became useless. Moreover, when it resumed execution on core y, it experienced additional cache misses initially. Such SMP OS behavior impacts cache performance and reduces the overall application performance.

Another source of significant overhead in the SMP multicore model is connected with synchronization and serialization through semaphores and spin-locks. Semaphores and spin locks are the most common mechanisms used to protect critical sections in the code, to protect shared data structures, to ensure atomic execution sequence, etc. When multiple threads compete against the same lock, the threads that lose will simply continue to compete until they get the lock. Progress is stalled and bus bandwidth is consumed in the meantime. As more cores and threads are deployed, they compete against one another, decreasing overall performance.

To allow multicore processing to truly scale to satisfy the needs of high-end application delivery switches, developers had to find ways to reduce overhead related to SMP OS scheduling, synchronization, and serialization.

Optimized Multicore Processors

The latest generation of multicore processors is ideally suited for demanding applications like application delivery switches. Specifically designed for low power and high density, processors with up to 16 integrated cores are currently available, with 32 cores reaching the market soon.

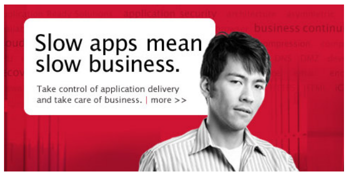
Coordinating tasks efficiently among many cores could be an arduous challenge for software developers. The latest multicore processors minimize software development complexity by implementing an industry-standard Instruction Set Architecture (ISA), such as MIPS64. They also incorporate a number of hardware features and optimizations that facilitate performance scaling—even when many cores are deployed.

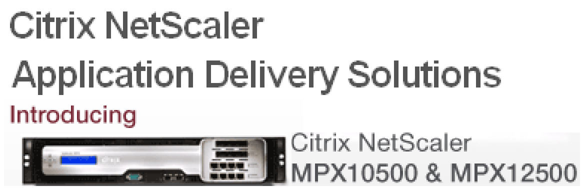
One of the most important of these features is hardware scheduling. By recognizing flows and scheduling the processing of packets to individual cores, the hardware scheduler eliminates the need to run an SMP OS on these cores. The result is no SMP OS scheduling overhead.

In addition, the processing is left to run to completion, rather than swapping it in and out. Cache performance is maximized, while overhead from swapping processing tasks in and out is minimized. This approach is often described as running on bare-metal, because an OS is not utilized on these cores. Because the scheduler can recognize flows, it can also ensure that packets belonging to the same flow are processed in the correct ingress order. In addition, processing sequences requiring atomicity can be scheduled so that the scheduler can guarantee atomic operations rather than having software maintain atomic sequence. With the task scheduler maintaining packet order and atomic sequences, there is no need for locking or synchronization/serialization, which accounted for a large portion of overhead in earlier multicore implementations.

Another key component of the latest multicore processors is integrated application hardware acceleration engines that streamline TCP protocol processing, security processing, and content processing. While earlier multicore processors performed these functions in software, newer multicore processors use acceleration engines to offload many mechanical or compute-intensive tasks, providing significantly higher performance at much lower power consumption. In addition, hardware acceleration, such as packet buffer management, can eliminate atomic sequence for managing buffers in a linked list.

Offloading a process that involves atomic sequences eliminates the need for software locking. For example, a hardware packet buffer manager can allocate and de-allocate buffers. Software just performs a simple read to allocate a buffer and a simple write to de-allocate. Without such hardware, software would implement a shared buffer linked list. In this case, multiple read/write transactions are needed to take a buffer out of the shared linked list, and re-insert a buffer. These transactions need to be atomic, because the linked list is a shared resource to all the execution threads. As this example illustrates, hardware offload not only provides an efficient implementation, but also helps multicore performance scaling by avoiding software solutions that would require locking.


Slow apps mean slow business.
Take control of application delivery and take care of business. | more >>



About Cavium Networks

Cavium Networks is a leading provider of highly integrated semiconductor products that enable intelligent processing in networking, communications, wireless, storage, video and security applications. Cavium Networks offers a broad portfolio of integrated, software-compatible processors ranging in performance from 10 Mbps to 20 Gbps that enable secure, intelligent functionality in enterprise, data-center, broadband/consumer and access and service provider equipment. Cavium Networks processors are supported by ecosystem partners that provide operating systems, tool support, reference designs and other services. Cavium Networks principal offices are in Mountain View, CA with design team locations in California, Massachusetts and India. For more information, please visit: http://www.caviumnetworks.com.