ciscopress.com

Overcoming Transport and Link Capacity Limitations Through WAN Optimization

Date: Jun 19, 2008 By Joel Christner and Ted Grevers. Sample Chapter is provided courtesy of Cisco Press.

This chapter examines how WAN optimization capabilities overcome performance barriers created by WAN conditions.

This chapter includes the following topics:

- Understanding Transport Protocol Limitations
- Understanding Transmission Control Protocol Fundamentals
- Overcoming Transport Protocol Limitations
- Overcoming Link Capacity Limitations

The previous chapters have discussed how to align network resources with business priority and application requirements, as well as techniques that can be applied within the network and accelerator devices to improve the overall performance of an application over the network. Techniques employed in the network, such as quality of service (QoS), can help align available network capacity with application throughput requirements or adjust queuing and scheduling for a specific application that might be latency sensitive. Application data caching, read-ahead, prediction, and CDN capabilities can help mitigate the unnecessary utilization of bandwidth for redundant object transfers and also mitigate latency by handling the majority of the workload locally or in a more optimized fashion.

You can, however, employ more generic layers of optimization, which can work across multiple applications concurrently. This type of optimization, commonly called *WAN optimization*, generally refers to functions that are commonly found in accelerator devices (such as standalone appliances or router-integrated modules) that overcome performance limitations caused by transport protocols, packet loss, and capacity limitations.

WAN optimization capabilities make the WAN a more tolerable place for applications to live by removing the vast majority of performance barriers that the WAN creates. For instance, advanced network compression can be applied to improve performance by minimizing the amount of data that needs to traverse the WAN. A secondary benefit of this is that, in many cases, fewer exchanges of data need to occur over the WAN as well, thereby mitigating the latency associated with the number of roundtrip exchanges that would have been necessary. TCP optimization, on the other hand, is commonly used to allow nodes to more efficiently use available resources and minimize the impact of loss and latency in a WAN.

This chapter examines how WAN optimization capabilities overcome performance barriers created by WAN conditions. Keep in mind that, in terms of accelerator products, you can use WAN optimization capabilities in conjunction with other optimization technologies that are application-specific, as described in Chapter 4, "Overcoming Application-Specific Barriers." Furthermore, assuming the architecture of the accelerator is transparent, you can use these optimization technologies in conjunction with network-oriented functions that provide visibility and control.

Understanding Transport Protocol Limitations

What is a transport protocol and how does it create performance limitations? Most people wonder how TCP (or other transport protocols) could ever become a bottleneck, simply because it always seems to just "work." In an internetwork, a layer must exist between applications and the underlying network infrastructure. This layer, called the *transport layer*, not only helps to ensure that data is moved between nodes, but also helps nodes understand how the network is performing so that they can adapt accordingly.

While the transport layer is an unlikely candidate for application performance woes, it can become a problem, primarily because the transport protocols in broad use today were designed in 1981. Today's application demands and network topologies differ greatly from the networks of the early 1980s. For instance, 300 baud was considered blazing fast at the time that TCP was created. Congestion was largely due to a handful of nodes on a shared network of limited scale rather than the complex, high-speed, hierarchical networks such as the Internet, which is plagued with oversubscription, aggregation, and millions of concurrent users each contending for available bandwidth. Applications in 1981 were commonly text-oriented applications (and largely terminal-oriented), whereas today even the most ill-equipped corporate user can easily move files that are tens upon hundreds of megabytes in size during a single transfer.

Although the network has changed, TCP remains relevant in today's dynamic and ever-changing network environment. TCP has undergone only minor changes in the past 25 years, and those changes are in the form of extensions rather than wholesale rewrites of the protocol. Although there are some more modern transport protocols that have roots in TCP, many are considered developmental projects only and currently have limited deployment in the mainstream.

Another important consideration relative to today's enterprise networking and application environments is the cost and available capacity of LAN technology versus declining WAN technology costs. In effect, network bandwidth capacity has steadily increased over the past 20 years; however, the cost of LAN bandwidth capacity has dropped at a rate that is more dramatic per bit/second than the rate at which the cost of an equivalent amount of WAN bandwidth capacity has dropped.

The ever-increasing disparity between WAN and LAN bandwidth presents challenges in the form of throughput. Applications and access to content have become more enabled for LAN users as LAN bandwidth has increased; however, the same level of access to those applications and content has not become more enabled for WAN users given the far different cost versus bandwidth capacity increase metrics that the WAN has undergone. Put simply, the rate of bandwidth increase found in the WAN is not keeping pace with the LAN, and this creates performance challenges for users who are forced to access information over the WAN. In this way, the LAN has enabled faster access to a much more network-intensive set of data. At the same time, the WAN has not grown to the same degree from a capacity perspective or become achievable from a price perspective to allow the same level of access for nearly the same cost.

WAN optimization techniques (discussed later in this chapter, in the section "Accelerators and Compression") are considered adjacent and complementary to the techniques presented in earlier chapters; that is, they are implemented apart from network optimization (QoS and optimized routing) and application acceleration (caching, CDN, and other optimizations such as read-ahead). For instance, an object-layer application cache can leverage compression technologies during content distribution or pre-position jobs to improve throughput and ensure that the compression history is populated with the relevant content if the transfer of the objects in question takes place over the WAN.

In the opposite direction, a user has a read-write type of relationship with an object that has been opened through an accelerator's object cache, where that object has been validated against the origin server (in the case of a cached file). If that file is saved and written back to the origin server, the compression history and protocol optimizations (such as write-back optimization) can be leveraged to improve the write performance while saving bandwidth.

Compression techniques can be leveraged to minimize the bandwidth consumed and eliminate previously seen repetitive byte patterns. This not only helps to ensure that precious WAN bandwidth is conserved but also serves to improve the performance of the user experience because less bandwidth is needed. Consequently, fewer packet exchanges must occur before the operation completes.

Coupling compression and the application acceleration techniques discussed in previous chapters with optimizations to the transport protocol ensures that the WAN is used efficiently and the user experience is significantly optimized. In many cases, WAN users experience performance levels similar to those experienced when operating in a LAN environment. WAN optimization helps to overcome the constraints of the WAN while maintaining WAN cost metrics, preserving investment, and providing a solution for consolidating distributed server, storage, and application infrastructure.

Put simply, the network is the foundation for an application-fluent infrastructure, and an optimized foundation provides the core for application performance. Transport protocol optimization and compression (that is, WAN optimization) ensure that resources are used effectively and efficiently while overcoming performance barriers at the data transmission layer. Application acceleration works to circumvent application layer performance barriers. These technologies are all independent but can be combined cohesively to form a solution, as shown in Figure 6-1.

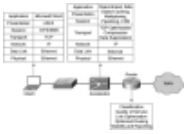

Figure 6-1 Application Acceleration and WAN Optimization Hierarchy

Understanding Transmission Control Protocol Fundamentals

TCP is the most commonly used transport protocol for applications that run on enterprise networks and the Internet today. TCP provides the following functions:

- Connection-oriented service between application processes on two nodes that are exchanging data
- Guaranteed delivery of data between these two processes
- Bandwidth discovery and congestion avoidance to fairly utilize available bandwidth based on

utilization and WAN capacity

NOTE

For more information about TCP, see RFC 793, which you can access through the following link: http://www.rfc-editor.org/rfcsearch.html

Before data can be sent between two disparate application processes on two disparate nodes, a connection must first be established. Once the connection is established, TCP provides guaranteed reliable delivery of data between the two application processes.

Connection-Oriented Service

The TCP connection is established through a three-way handshake agreement that occurs between two sockets on two nodes that wish to exchange data. A *socket* is defined as the network identifier of a node coupled with the port number that is associated with the application process that desires to communicate with a peer. The use of TCP sockets is displayed in Figure 6-2.

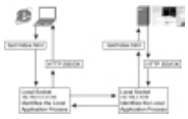

Figure 6-2 TCP Socket

During the establishment of the connection, the two nodes exchange information relevant to the parameters of the conversation. This information includes

- **Source and destination TCP ports:** The ports that are associated with application processes on each of the nodes that would like to exchange application data.
- **Initial sequence number:** Each device notifies the other what sequence number should be used for the beginning of the transmission.
- **Window size:** The advertised receive window size; that is, the amount of data that the advertising node can safely hold in its socket receive buffer.
- **Options:** Optional header fields commonly used for extensions to TCP behavior. For instance, this could include features such as window scaling and selective acknowledgment that were not included as part of the TCP RFC but can augment TCP behavior (an authoritative list of TCP options can be found at http://www.iana.org/assignments/tcp-parameters).

For instance, if an Internet user wants to use Internet Explorer to access http://www.cisco.com, the user's computer would first have to resolve the name www.cisco.com to an IP address, and then attempt to establish a TCP connection to the web server that is hosting www.cisco.com using the well-known port for HTTP (TCP port 80) unless a port number was specified. If the web server that is hosting www.cisco.com is accepting connections on TCP port 80, the connection would likely establish successfully. During the connection establishment, the server and client would tell one another how much data they can receive into their socket buffer (window size) and what initial sequence number to use for the initial transmission of data. As data is exchanged, this number would increment to allow the receiving node to know the appropriate ordering of data. During the life of the connection, TCP employs checksum functionality to provide a fairly accurate measure of the integrity of the data.

Once the connection is established between two nodes (IP addresses) and two application process identifiers (TCP ports), the application processes using those two ports on the two disparate nodes can begin to exchange application layer data. For instance, once a connection is established, a user can submit a GET request to the web server that it is connected to in order to begin downloading objects from a web page, or a user can begin to exchange control messages using SMTP or POP3 to transmit or receive an e-mail message. TCP connection establishment is shown in Figure 6-3.

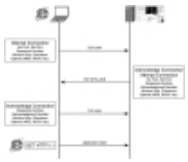

Figure 6-3 TCP Connection Establishment

Guaranteed Delivery

Once transmission commences, application data is drained from application buffers on the transmitting process into the node's socket buffer. TCP then negotiates the transmission of data from the socket transmission buffer to the recipient node (that is, the draining of the buffer) based on the availability of resources on the recipient node to receive the data as dictated by the initially advertised

window size and the current window size. Given that application data blocks may be quite large, TCP performs the task of breaking the data into segments, each with a sequence number that identifies the relative ordering of the portions of data that have been transmitted. If the node receives the segments out of order, TCP can reorder them according to the sequence number. If TCP buffers become full for one of the following reasons, a blocking condition could occur:

- **TCP transmit buffer becomes full:** The transmit buffer on the transmitting node can become full if network conditions prevent delivery of data or if the recipient is overwhelmed and cannot receive additional data. While the receiving node is unable to receive more data, applications may be allowed to continue to add data to the transmit buffer to await service. With the blockade of data waiting in the transmit buffer, unable to be transmitted, applications on the transmitting node may become blocked (that is, momentary or prolonged pauses in transmission). In such a situation, new data cannot be written into the transmit buffer on the transmitting node unless space is available in that buffer, which generally cannot be freed until the recipient is able to receive more data or the network is able to deliver data again.
- **TCP receive buffer becomes full:** Commonly caused by the receiving application not being able to extract data from the socket receive buffer quickly enough. For instance, an overloaded server, i.e. one that is receiving data at a rate greater than the rate at which it can process data, would exhibit this characteristic. As the receive buffer becomes full, new data cannot be accepted from the network for this socket and must be dropped, which indicates a congestion event to the transmitting node.

Figure 6-4 shows how TCP acts as an intermediary buffer between the network and applications within a node.

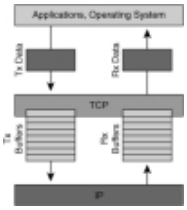

Figure 6-4 TCP Buffering Between the Network and Applications

When data is successfully placed into a recipient node TCP receive buffer, TCP generates an acknowledgment (ACK) with a value relative to the tail of the sequence that has just been received. For instance, if the initial sequence number was "1" and 1 KB of data was transmitted, when the data is placed into the recipient socket receive buffer, the recipient TCP stack will issue an ACK with a value of 1024. As data is extracted from the recipient node's socket receive buffer by the application process associated with that socket, the TCP stack on the recipient will generate an ACK with the same acknowledgment number but will also indicate an increase in the available window capacity. Given that applications today are generally able to extract data immediately from a TCP receive buffer, it is likely that the acknowledgment and window relief would come simultaneously.

The next segment that is sent from the transmitting node will have a sequence number equal to the previous sequence number plus the amount of data sent in the previous segment (1025 in this example), and can be transmitted only if there is available window capacity on the recipient node as dictated by the acknowledgments sent from the recipient. As data is acknowledged and the window value is increased (data in the TCP socket buffer must be extracted by the application process, thereby relieving buffer capacity and thus window capacity), the sender is allowed to continue to send additional data to the recipient up to the maximum capacity of the recipient's window (the recipient also has the ability to send dynamic window updates indicating increases or decreases in window size).

This process of transmitting data based on the recipient's ability to receive and previously acknowledged segments is commonly referred to as the TCP *sliding window*. In essence, as the recipient continues to receive and acknowledge or otherwise notify of an increase in window size, the window on the transmitting node shifts to allow new data to be sent. If at any point buffers become full or the window is exhausted, the recipient must first service data that has been previously received and acknowledge the sender before any new data can be sent. An example of this process is shown in Figure 6-5.

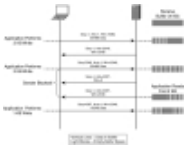

Figure 6-5 TCP Operation

Additionally, TCP provides a flag that allows an application process to notify TCP to send data

called a *push*, or PSH, instructs TCP to immediately send all data that is buffered for a particular destination. This push of data also requires that previous conditions be met, including availability of a window on the recipient node. When the data is transmitted, the PSH flag in the TCP header is set to a value of 1, which also instructs the recipient to send the data directly to the receiving application process rather than use the socket receive buffer.

Nodes that are transmitting also use the acknowledgment number, sequence number, and window value as a gauge of how long to retain data that has been previously transmitted. Each segment that has been transmitted and is awaiting acknowledgment is placed in a retransmission queue and is considered to be unacknowledged by the recipient application process. When a segment is placed in the retransmission queue, a timer is started indicating how long the sender will wait for an acknowledgment. If an acknowledgment is not received, the segment is retransmitted. Given that the window size is generally larger than a single segment, many segments are likely to be outstanding in the network awaiting acknowledgment at any given time.

From a transport layer perspective, the loss of a segment might not prevent transmission from continuing. However, given that the application layer is really dictating transport layer behavior (for instance, an upper-layer protocol acknowledgment), the loss of a segment may indeed prevent transmission from continuing.

The purpose of this retransmission queue is twofold:

- It allows the transmitting node to allocate memory capacity to retain the segments that have been previously transmitted. If a segment is lost (congestion, packet loss), it can be transmitted from the retransmission queue, and remains there until acknowledged by the recipient application process (window update).
- It allows the original segment, once placed in the retransmission queue, to be removed from the original transmission queue. This in effect allows TCP to be continually extracting data from the local transmitting application process while not compromising the transmitting node's ability to retransmit should a segment become lost or otherwise unacknowledged.

An example of TCP retransmission management is shown in Figure 6-6.

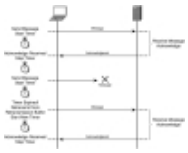

Figure 6-6 TCP Retransmission Management

Bandwidth Discovery

The TCP sliding window can act as a throttling mechanism to ensure that transmission of data is done in such a way that it aligns with the available buffer capacity and window of the two devices exchanging data. There are also mechanisms in TCP that allow it to act as a throttling mechanism based on the capacity of the network and any situations encountered in the network.

In some cases, the nodes exchanging data are able to send more data than the network can handle, and in other cases (which is more prominent today), the nodes are not able to send as much data as the network can handle. In the case of nodes being able to transmit more data than the network is prepared to handle, congestion and loss occur. TCP has mechanisms built in to account for these situations. In this way, TCP is *adaptive* in that it bridges the gap between transmission requirements and limitations, receive requirements and limitations, congestion, and loss, for both the application process *and* the network. This throttling mechanism also provides a balancing act between applications and the network and works to continually leverage what capacity in the network is available while attempting to maximize application throughput.

Discovering the steady state for an application process, meaning the balance between available network capacity and the rate of data exchange between application processes and socket buffers, is wholly subjective because application behavior is largely dictated by the function of input, meaning the rate at which the application attempts to send or receive data from the socket buffers. Network throughput is generally more deterministic and objective based on a variety of factors (which would otherwise make it appear nondeterministic and fully subjective), including bandwidth, latency, congestion, and packet loss.

The terms congestion and loss are used separately here, even though congestion and loss are generally married. In some situations, congestion can simply refer to a delay in service due to saturated buffers or queues that are not completely full, but full enough to slightly delay the delivery of a segment of data. These factors are deterministic based on network utilization and physics rather than a set of input criteria, as would be the case with an application. In any case, applications are driving the utilization of network bandwidth, so the argument could be made that they go hand in hand.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.