

White Paper

Eclipse: Performance over the WAN

“The Standard That Can’t Be Beat”

Submitted By:

Mark S. Detrick, CCIE 6336
Systems Architect, SAN Extension and Routing
Mark.Detrick@McDATA.com

McDATA Corporation
Broomfield, CO
www.McDATA.com

Introduction

The McDATA Eclipse has a number of features that provide significant performance enhancements for transmitting storage data over WAN links. Those features include:

- FastWrite®
- Data Compression
- Storage Optimized TCP
- Rate Limiting
- Eclipse FC to IP Architecture & Methodology

The purpose of this white paper is to describe in technical detail the operation and benefit of each of these features. Ultimately, these features facilitate throughput of storage data across a WAN faster than competing products. For customers that wish to move data reliably and as fast as possible over prudent affordably sized links the Eclipse proves superior in every performance category.

FastWrite®

The FastWrite® algorithm is the most mature in the market with over 4 years of engineering experience and customer feedback. FastWrite® is an algorithm developed by Nishan Systems in 2001 that reduces the number of round trips required to complete a SCSI write operation. FastWrite® can maintain throughput levels over links that have significant latency. The RDR (Remote Data Replication) application still experiences latency but does not experience reduced throughput due to that latency.

Typical SCSI behavior is shown in Figure 1 below. There are two steps to a SCSI write. First, the write command is sent across the WAN to the target. The first round trip is essentially asking permission of the storage array to send data. The target responds with an acceptance (FCP_XFR_RDY). The initiator waits until it receives a response from the target before starting the second step, sending the data (FCP_DATA_OUT).

This simplex communication over a WAN link, with any appreciable latency, aggregates the time it takes to complete the entire write procedure. In the example diagram below there are two round trips required to complete the write, therefore, the write will take at least 2 x RTT (Round Trip Time). Multiple write commands can be outstanding at any one time; however, every command must wait for a response before moving on. Multiple outstanding commands can help the throughput problem but will not solve it.

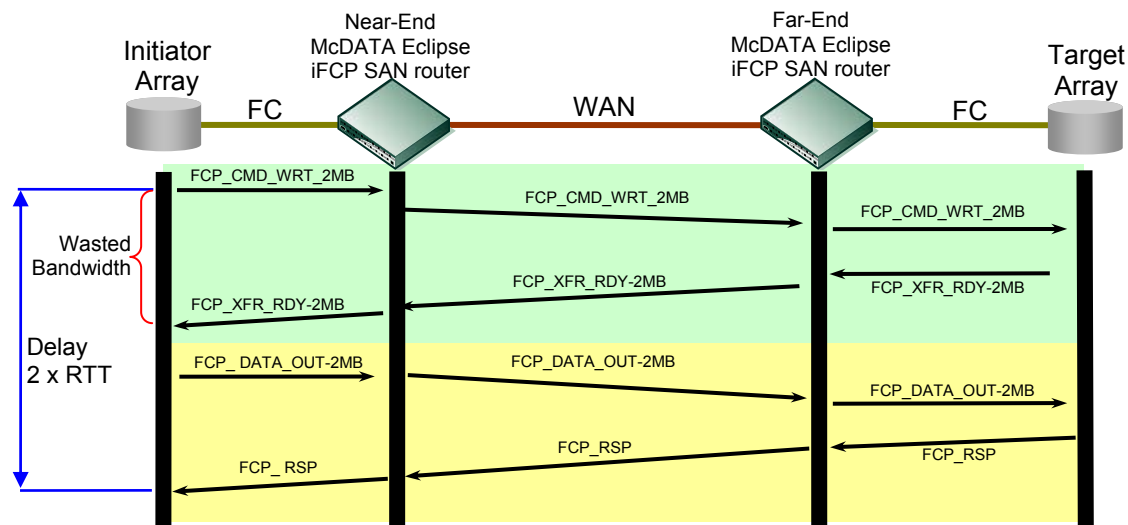


Figure 1: SCSI Write without FastWrite®

With the McDATA FastWrite[®] algorithm the Eclipse SAN router intercepts the originating write command and responds immediately (in terms of μ s) requesting the initiator to send the entire data set. The initiator will start to send the data. If the Eclipse buffers become full, because the egress is slower than the ingress, it will apply back pressure to the initiator by starving BB credits to the source. As the buffers empty out, BB credits are sent to the initiator to enable sending more data. The Eclipse will have a continuous supply of data in its buffers to completely fill the WAN driving optimized throughput.

The buffers in the Eclipse are very large and can accommodate an enormous quantity of TCP segments so that TCP can fill long fat pipes. There may be many outstanding segments at any one time. The buffer space for each TCP session is 8 MB. Connecting two controllers on each side will create two TCP sessions each with 8 MB of memory. This is enough memory to fill a GE link that extends half way around the world. RFC 1323 is used to scale the TCP congestion window. In addition, the TCP stack has been modified to optimize storage transmission and compensate for links that may generate Out of Order Packets or Bit Error Rates (BER).

The SAN router will start to send data across the link until all the bandwidth has been consumed. The receiving SAN router will act on behalf of the initiator and open a write exchange with the target over the local SAN or direct connection. Often, this technology allows a write to complete in a single round trip speeding up the process tremendously and mitigating link latency by 50%.

There is no possibility of data corruption with FastWrite[®] because the final response (FCP_RSP) is never spoofed, intercepted or altered in any way. It is this final response that the receiving device sends to indicate that the entire data set has been successfully received and committed. The near-end SAN router does not generate the final response in an effort to expedite the process, nor does it need to. If any single FC frame were to be corrupted or lost along the way the target would detect it and not send the final response. If the final response is not received within a certain amount of time the write will timeout and be resent, in any case the initiator will know that the write was unsuccessful and act accordingly.

IT professionals have a choice, with synchronous applications links can be up to twice the distance or response time can be halved. RDR/A applications benefit by smaller bandwidth links or more data per second, either way it results in greater ROI and lower TCO. Please refer to Figure 2 below.

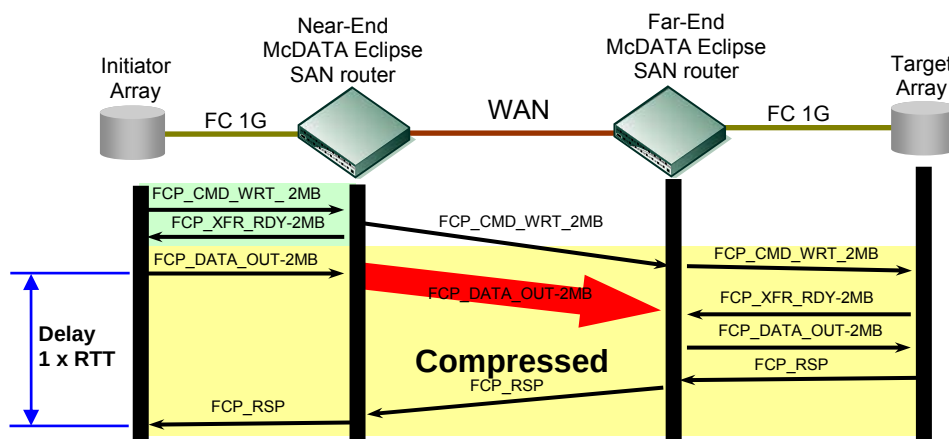


Figure 2: SCSI Write with FastWrite[®]

Eclipse FastWrite® vs. Cisco MDS 9216i Write Accelerate Testing

In a lab environment, the Eclipse EOSi 4.7 FastWrite® and 9216i 14/2 module running SAN-OS 2.1.1a Write Accelerate were tested for response time and throughput at various latencies. A common block I/O size was used (32 KB) and the standard MTU size of 1500 bytes. Because FastWrite® and Write Accelerate only work with SCSI write traffic the data was limited to writes. IOMeter was used to generate the SCSI traffic. Queue depth was set to 16.

In summary, the Eclipse had lower response time compared to the 9216i across the various latencies. The results are as follows:

Response Time

In Figure 3 below, the results demonstrated that the Eclipse has lower response time compared to the Cisco 9216i across the board. Obviously, lower response time is desirable. Having lower response time means transactions will complete faster resulting in more transactions per unit time.

The response time as reported by IOMeter is the amount of time it takes to complete the write command in its entirety. The traffic must leave the server, pass through the local and remote SAN router, be responded to by the storage system and return through the same path. The test equipment and conditions (storage, WAN simulator, IOMeter server) were the same for both platforms; therefore, the results are relative.

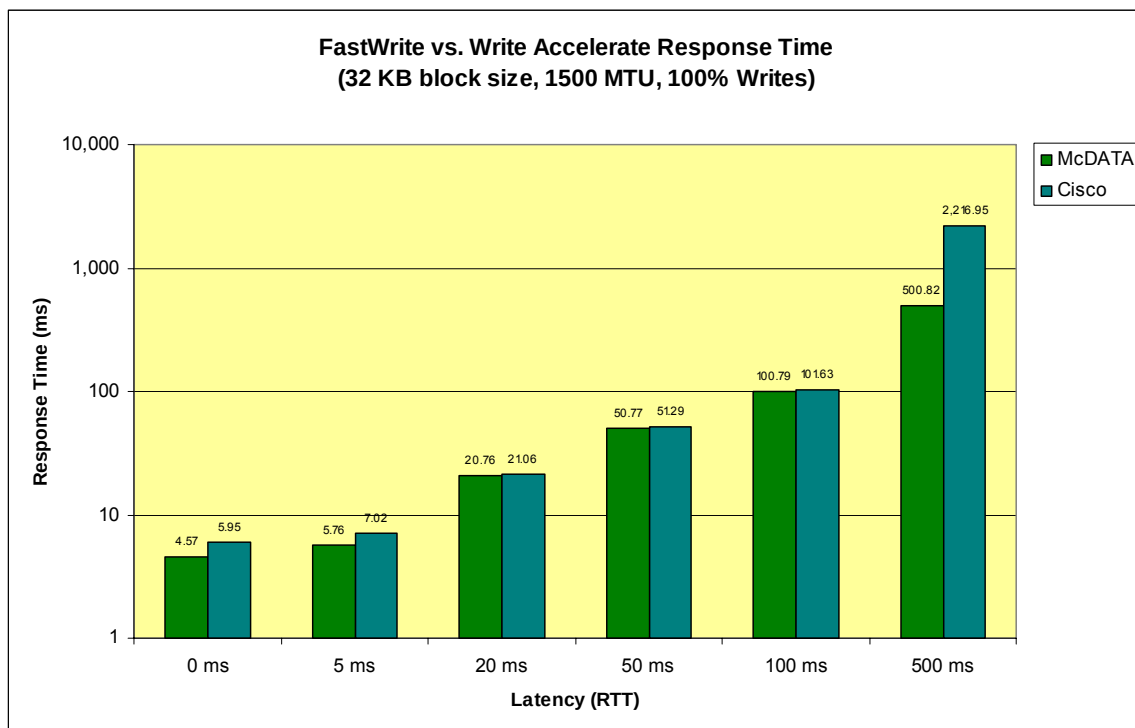


Figure 3: FastWrite® vs. Write Accelerate Response Time

During testing a comparison of FastWrite® disabled to FastWrite® enabled was made. This is followed by the same test for Write Accelerate. The following graphs outline the percentage degradation (up bars) or improvement (down bars) in response time. You will notice that the Eclipse causes no degradation of either response time or throughput at any latency and operates at latencies as high as 1 second (1000 ms), refer to Figure 4 below. This was not true for the 9216i 14/2, which had higher response times at zero latency and did not operate well at 500 ms and above.

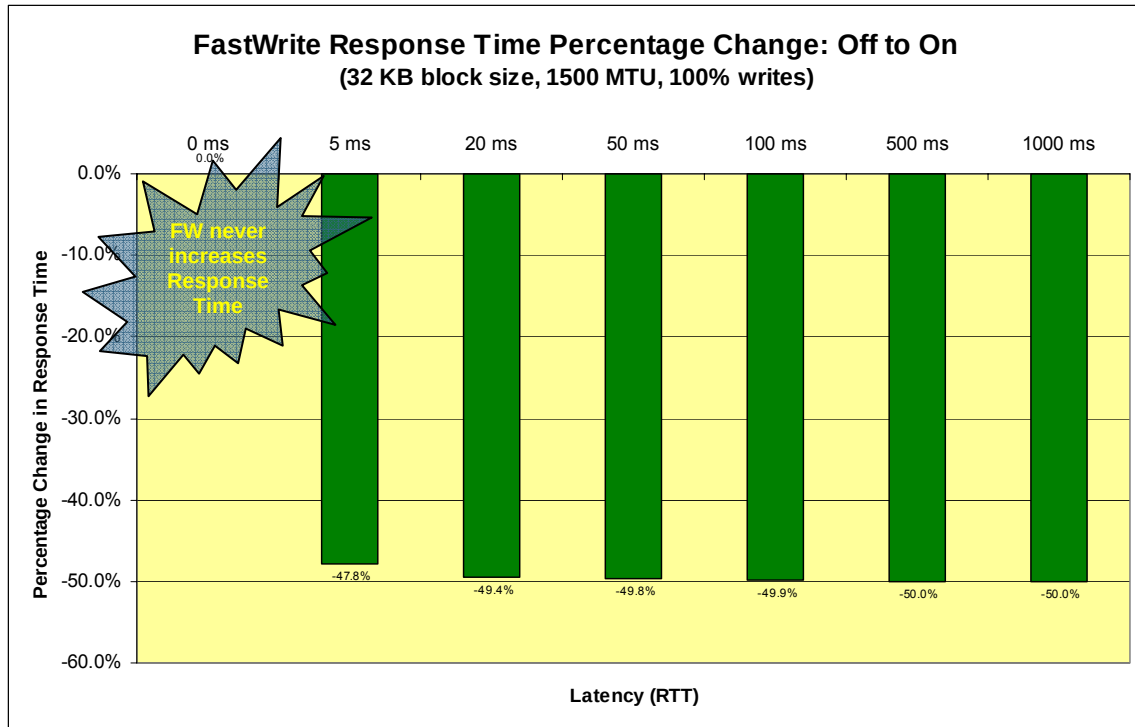


Figure 4: FastWrite® Response Time

The FastWrite® technology is perfectly demonstrated in this test. The graph shows that the response time has been reduced by half, as expected. The first round trip was eliminated. At small block sizes, response time is approximately 2 x RTT, with FastWrite® it is 1 x RTT, or 50%.

Cisco MDS 9216i 14/2 Module Write Accelerate showed an increase in Response Time with no latency. Beyond 300 ms the 9216i became very inefficient at transmitting traffic and could not pass traffic beyond 500 ms. At 1 second latencies the 9216i could not establish a connection, refer to Figure 5 below. It should be noted that at no time was the 9216i 14/2 module able to lower response time by as much as the Eclipse. The Eclipse showed consistent values of nearly 50% improvement. The greatest improvement in response time for the 9216i 14/2 module was 37.1%, a delta of 13%.

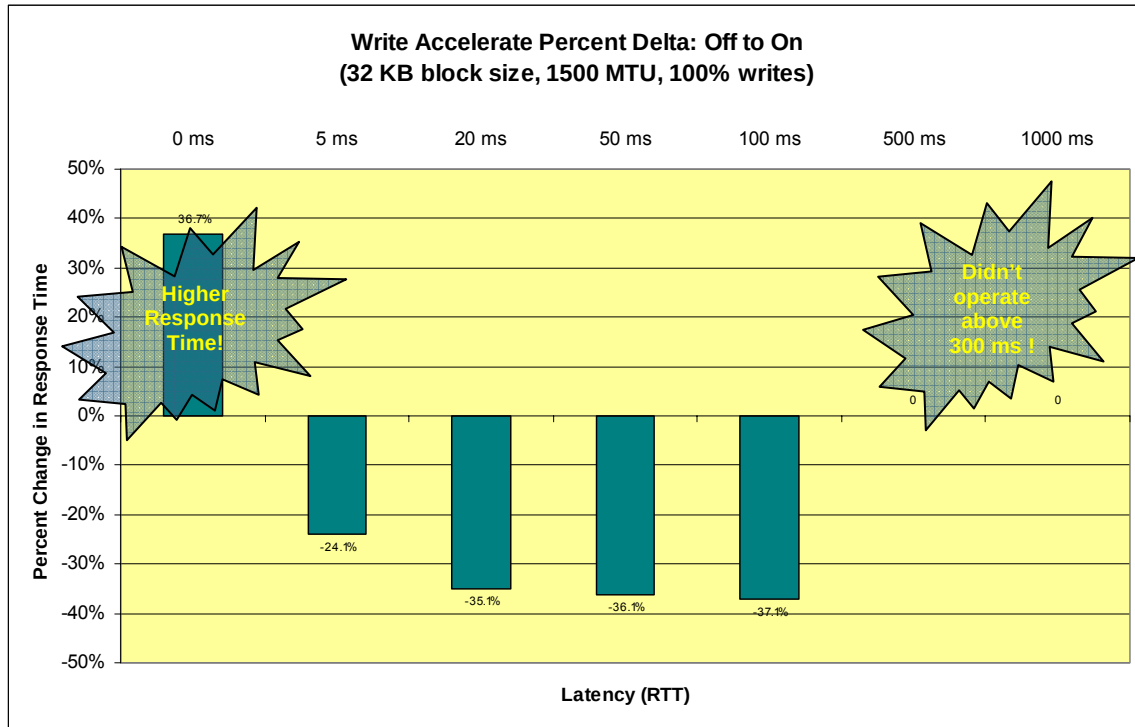


Figure 5: Write Accelerate Response Time

Throughput

The Cisco MDS 9216i 14/2 had lower throughput performance compared to the Eclipse at certain latencies. In fact, the Eclipse was superior in throughput at small latencies (0 to 5 ms) and at high latencies (above 100 ms), refer to Figure 6 below. From 20 to 100 ms of latency the 9216i 14/2 matched the Eclipse performance. While the throughput from 20 to 100 ms was the same, the response time was significantly better for the Eclipse.

Beyond 300 ms the 9216i has trouble passing traffic efficiently. At longer latencies, beyond 500 ms, the 9216i was not able to pass data or in some cases could not establish a connection. These two problems did not occur on the Eclipse. The Eclipse was able to transmit traffic at latencies between 500 and 1000 ms. The write enhancement technology in the Eclipse is powerful and more mature.

In the graphs below, the bars indicate the percentage change with write enhancement enabled compared to disabled. Throughput is indicated by down and up bars, reduced and increased throughput respectively.

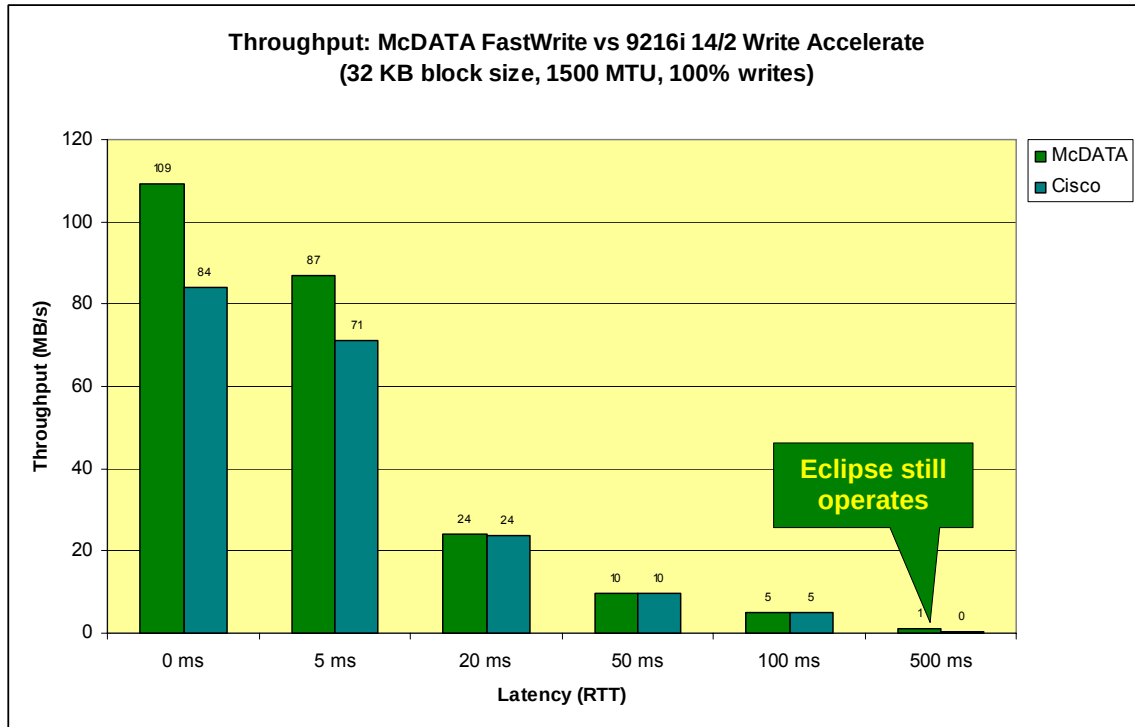


Figure 6: FastWrite® vs. Write Accelerate Throughput

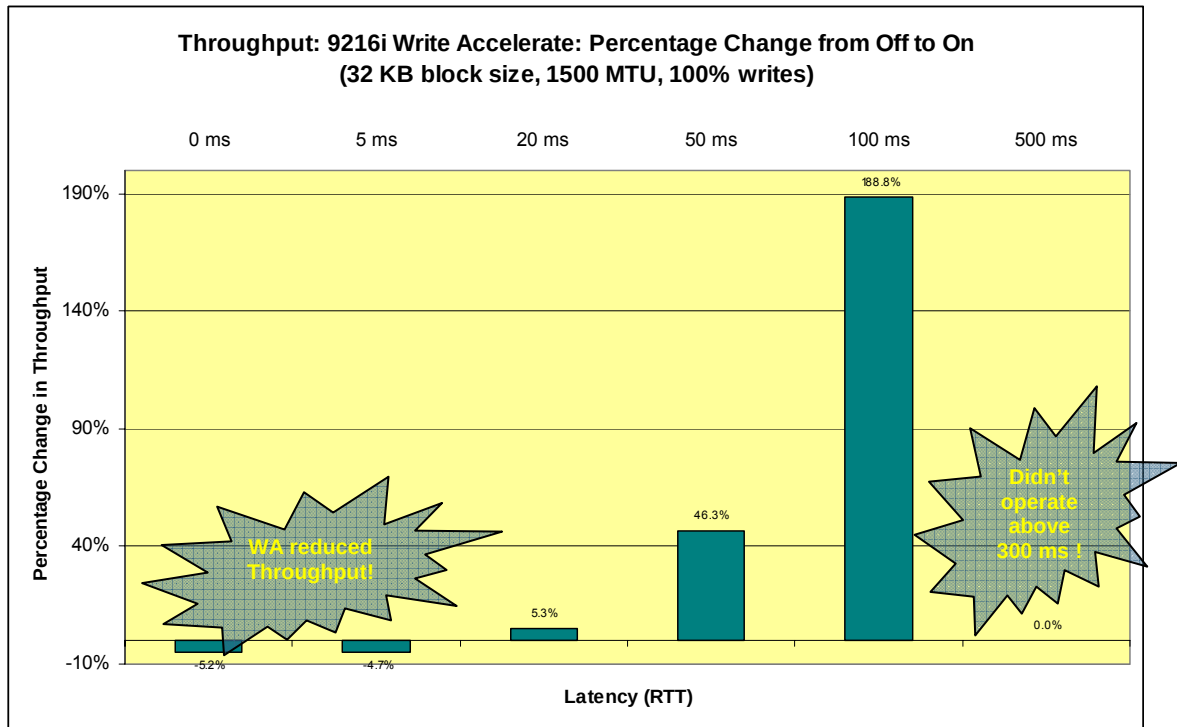


Figure 7: Write Accelerate Throughput Change

At latencies less than 20 ms, the Cisco MDS 9216i 14/2 module reduced throughput by approximately 5%, refer to Figure 7 above. At 5 ms the Eclipse is useful for RDR/S applications with improved response time and greater throughput. In a metro network the Eclipse can cut synchronous response time by nearly 50%. On the other hand, the 9216i 14/2 will reduce throughput and only moderately improve response time (24%).

Beyond 300 ms the 9216i 14/2 module has trouble passing traffic efficiently. At longer latencies, beyond 500 ms, the 9216i couldn't pass data or in some cases couldn't establish a connection. These two problems did not occur on the Eclipse. This indicates a more capable and stable TCP stack on the Eclipse.

In Figure 8 below, the percentage increase in throughput comparing FastWrite[®] disabled to enabled is shown. Substantial increases in throughput were observed. The Eclipse at no time reduced throughput of data. The Eclipse was able to accommodate a greater amount of throughput at all latencies and maintain lower response times.

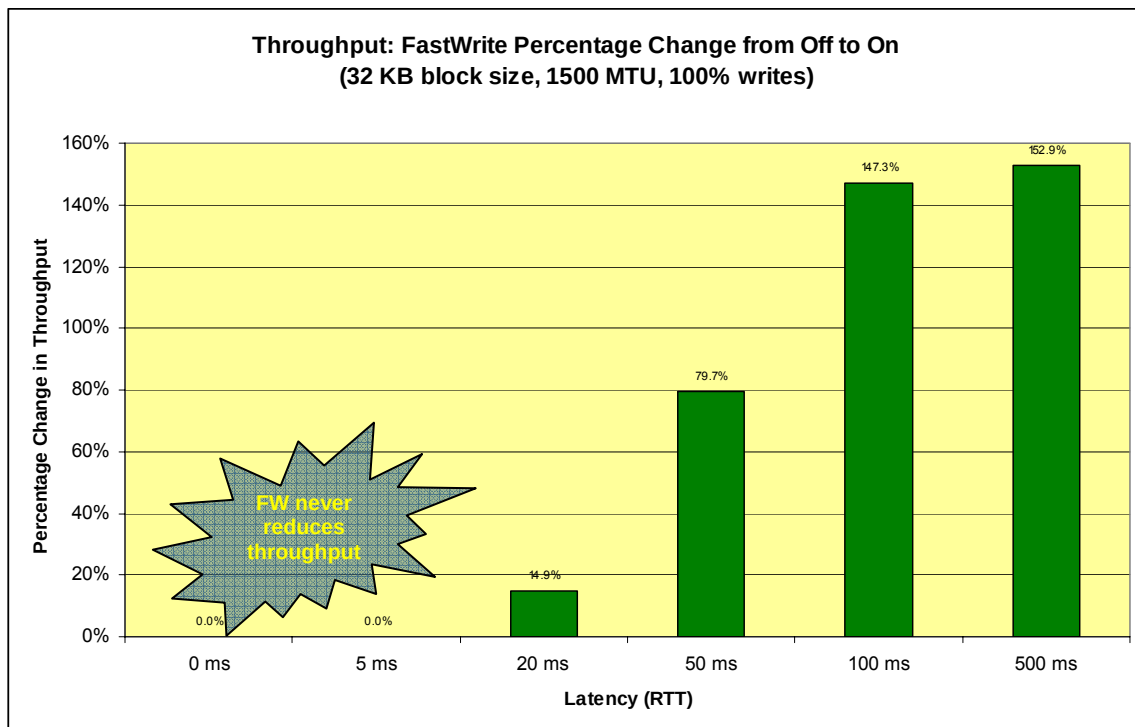


Figure 8: FastWrite[®] Throughput Improvement with FastWrite[®]

FastWrite® Case Study

Wachovia Bank used Eclipse with FastWrite® to migrate data from Chicago to Winston/Salem, approximately 700 miles (1,127 km), over an OC-3 (17.7 MB/s), refer to Figure 9 below. They had 4.2 TB of data to migrate. The link had a 23 ms RTT (Round Trip Time) and the RDR application was EMC SRDF Adaptive Copy. Compression averaged about 2:1.

The replication took 40 hours to complete at a rate of 109 GB per hour (FC ingress). OC-3 PoS (Packet over SONET) can accommodate about 170 GB/hour with no latency. Without FastWrite® the effects of latency causes the throughput to diminish to 3.8 GB/hour as demonstrated by the testing at Wachovia. FastWrite® increased the efficiency of the SCSI transactions over the link such that 109 GB/hour could be realized.

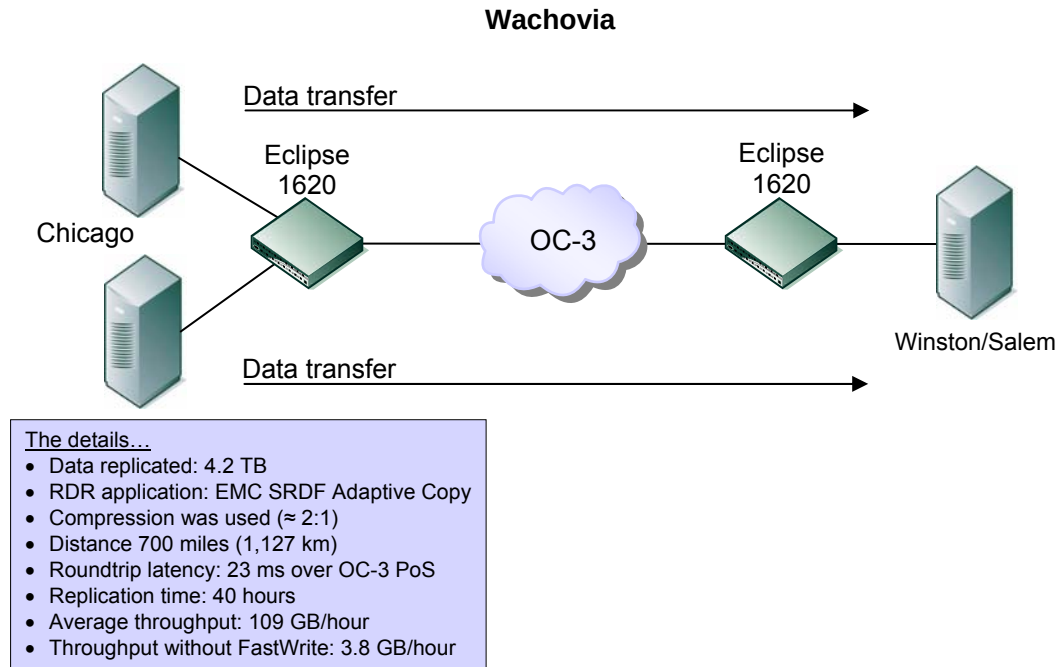


Figure 9: FastWrite Production Environment Case Study

Data Compression

Data compression can minimize WAN bandwidth requirements or increase overall RDR throughput resulting in possibly a less expensive WAN link. Maximum compression rate on the Eclipse is 26 MB/s (Fast LZO with History) for software based compression and 174 MB/s (FC ingress) for hardware based compression. Software compression is done on the dedicated RM7000 processor of an iFCP port. It is not done on the shared CPU of the SAN router itself. Hardware compression is done on a FPGA within the architecture of each intelligent port. Each iFCP port has its own hardware based processor. Hardware based compression uses the LZO algorithm. The RM7000 also handles the tasks of the TCP stack and FastWrite[®].

Compression is done only on the payload portion of the packets and only when the data field is 512 bytes or larger. There are diminishing returns when considering time to compress vs. uncompressed transmission with payloads of 512 bytes or less. Compression is done on the byte stream data prior to forming TCP segments; therefore, all data packets will utilize the MTU (see methodology section later in this document). If compression was done on the payload after forming the packets, there would be the same number of packets and overhead but smaller payload size, which results in a greater percentage of overhead to data and less efficiency.

The algorithm choice is based on link characteristics such as bandwidth and the number of required TCP sessions. If history is kept, additional memory is required for each TCP session limiting the total number of sessions to eight. History is a McDATA enhancement that increases the compression ratio.

Effective throughput will vary depending on the compressibility of the data. The compressibility of the data constantly changes as the byte stream passes through the compression engine. Effective throughput is the data rate out of the egress port multiplied by the compression ratio at that instance.

The Eclipse has the intelligence to monitor data for prior compression. The Eclipse will not attempt to recompress compressed data. Data that has been compressed prior to reaching the Eclipse may include: ZIP, JPEG, PDF, GIF, TIFF, MP3 and MPEG. The result may be an increase in data instead of a reduction. If data is not compressible then the throughput will be the same as the egress data rate, which is a compression ratio of 1:1. If data can be compressed then the throughput will be a positive multiple of the data rate out the egress port, which is a compression ratio of $n:1$ where $n > 1$.

The physical throughput out of the iFCP port will be consistent and sustainable at the set port rate limit. In other words, the Eclipse will fill the link at a constant and sustainable rate. Of course the data source, the RDR application, will need to provide data at the provisioned rate. Often on fatter pipes, the RDR application can't supply data traffic as fast as the Eclipse can send it.

There is no such thing as a data sheet spec for compression ratios unless either the maximum is stated or the spec is based on a known data set made available from the open source community such as Canterbury Corpus. A known data set will provide a baseline ratio across compression methods. Compression ratio varies from byte to byte as data streams through the algorithm. Therefore, it is impossible to state a compression ratio that means anything to normal enterprise data. The only way to determine the actual compression ratio is to try it on the data itself.

The maximum compression ratio achievable on the Eclipse is 25:1. For practical and comparison purposes this specification has little value.

*Appendix A provides additional details on the Eclipse compression algorithms.

Eclipse Compression vs. Cisco Compression Testing

In a lab setting, the Eclipse EOSi 4.7 was tested against the Cisco 9216i 14/2 module running SAN-OS 2.1.1a. Data representative of common enterprise data was passed through the products. The data set was consistent between tests, making the results relative. Compression ratio and throughput were recorded.

Both the McDATA Eclipse and Cisco 14/2 module have software based compression algorithms. These algorithms run on processors located on each port for the Eclipse and one on the 14/2 module. SW compression is used for low bandwidth applications. The throughput is limited because the processor has only a finite number of cycles available. The throughput and compression ratio depends on the algorithm. The Eclipse has a variety of algorithms for various link types.

Below is a table of Eclipse and 14/2 module algorithms listing the throughput and compression ratios observed during testing. Throughput is from the perspective of the FC ingress data rate, not the egress IP traffic. The I/O block size was 32 KB and the MTU was 1500 bytes. The results are as follows:

Table 1: Compression Ratio and Throughput Chart

Compression Ratio and Throughput		
Algorithm	Throughput (MB/s)	Compression Ratio
Eclipse HW LZO (HW-LZO)	173.99	1.58
Eclipse SW LZO (SW-LZO)	22.79	1.58
Eclipse LZO w/ History (LZO-H)	19.48	1.96
Eclipse Fast LZO w/ History (FLZO-H)	24.08	1.58
Eclipse Deflate	4.59	2.42
9216i 14/2 Mode 1	178.84	1.69
9216i 14/2 Mode 2	5.98	2.43
9216i 14/2 Mode 3	3.67	2.58

Cisco has no SW based compression that accommodates throughput above Ethernet (10 Mbps) speeds. MDS 9216i 14/2 must utilize HW based compression for all DS3 and faster links. A comparison of algorithms relative to link bandwidth is in the chart below.

First, an explanation of the calculations below is in order. Links, in the path of the egress port, have finite bandwidth. Because the throughput results are in terms of ingress data rate it is important to analyze the algorithm's ability to provide adequate throughput.

For example, if the link can accept 10 MB/s and the compression ratio is 2:1 the ingress rate would be 20 MB/s (Link Speed x Ratio). Can the algorithm keep up with 20 MB/s? Each algorithm has its own compression ratio used in the computations. The actual compression ratio will vary depending on the data. We have to rely on an average for the data set used in the testing. Your results may vary and tests should be performed on a snapshot of the actual data to be compressed. A higher compression ratio will increase the ingress data rate relative to the maximum egress bandwidth, which can result in the algorithm reaching its maximum throughput or surpassing its ability to keep up. Because the compression ratio changes continually this threshold can be reached from time to time even when on average the maximum throughput is not exceeded. The chart below provides calculations for the most popular links using the compression ratios obtained during testing.

Table 2: Compression Relative to Link Bandwidth

Compression Algorithm Relative to Link Bandwidth											
Link Type	Egress TP (MB/s)	Eclipse Algorithm	Eclipse Comp Ratio	Eclipse Ingress TP	Effective Ingress TP (MB/s)	9216i Algorithm	9216i Comp Ratio	9216i Ingress TP	Effective Ingress TP (MB/s)	Compression Ratio Delta	Throughput Delta (MB/s)
Ethernet	1.2	Deflate	2.42	4.59	2.90	Mode 3	2.58	3.67	3.10	14/2 6.6% higher	14/2 0.19 faster
DS3 ATM	4.4	LZO-H	1.96	19.48	8.62	Mode 1	1.69	178.84	7.44	Eclipse 13.8% higher	Eclipse 1.19 faster
DS3 CC	5.3	LZO-H	1.96	19.48	10.39	Mode 1	1.69	178.84	8.96	Eclipse 13.8% higher	Eclipse 1.43 faster
E3	4.1	LZO-H	1.96	19.48	8.04	Mode 1	1.69	178.84	6.93	Eclipse 13.8% higher	Eclipse 1.11 faster
FE	11.7	FLZO-H	1.58	24.08	18.49	Mode 1	1.69	178.84	19.77	14/2 7.0% higher	14/2 1.29 faster
OC-3 POS	17.7	HW-LZO	1.58	173.99	27.97	Mode 1	1.69	178.84	29.91	14/2 7.0% higher	14/2 1.95 faster
OC-12 POS	71.0	HW-LZO	1.58	173.99	112.18	Mode 1	1.69	178.84	119.99	14/2 7.0% higher	14/2 7.81 faster
GE	117.1	HW-LZO	1.58	173.99	173.99	Mode 1	1.69	178.84	178.84	14/2 7.0% higher	14/2 4.85 faster

The Eclipse will never deprive the egress queue from traffic based on compression processing. In fact, the Eclipse will always ensure the egress queue is adequately filled to prevent the link from going unutilized regardless of compression method, helping to maintain greater throughput.

Cisco did outperform the Eclipse over some link types, however, only nominally, never more than 7.0% for compression ratio. These are small differences easily compensated for by other advantages the Eclipse offers, for instance, FastWrite[®], architectural efficiency and SO-TCP.

The Eclipse had higher throughput by 27%, relative to link bandwidth, for DS3 (clear channel and ATM) and E3. Cisco at Ethernet speed was 16% greater throughput, 11% at FE, OC-3 & OC-12; and 4% at GE.

Storage Optimized TCP (SO-TCP)

Over the last couple of decades TCP has evolved such that one of its primary purposes is to play traffic cop for the Internet. The developers of TCP had to make certain assumptions about the network. It forces everybody accessing the Internet to be a good network citizen by providing flow control and congestion avoidance. If congestion is experienced and packets get dropped, TCP backs off dramatically allowing traffic to dissipate and flows to ramp up slowly preventing another congestion event from rapidly occurring again. The Internet would not be able to pass traffic efficiently without these congestion controls.

When an organization transmits storage they typically do not use the Internet. Storage transmission often uses dedicated links or circuits such as TDM (DS3), DWDM, dark fiber, SONET and ATM. Dedicated metro Ethernet links (10baseT, FE & GE) are also popular. With dedicated bandwidth different assumptions can and should be made by TCP, assumptions that standard TCP doesn't make. These networks are fundamentally different than the Internet.

McDATA Eclipse firmware EOSi 4.7 introduces new TCP features that allow these assumptions to be changed to better suit dedicated bandwidth found in enterprise environments. Primarily, when a segment is retransmitted over dedicated bandwidth there is no need for TCP to back off the data rate as much, nor does it need to worry about other contentious flows.

Why does iFCP and iSCSI use TCP at all? TCP offers the following benefits to data transmission:

- Data & header corruption can be detected
- Out of order segments can be placed back in order
- Segments lost in transit are detected and retransmitted
- Duplicate segments are detected and discarded

No matter how the IP network is architected, if TCP/IP can operate across it, FC frames will arrive at the destination via TCP with the same integrity and in the same order as they were received into the local FC port. There is no concern about FC frames getting corrupted, out of order, duplicated or lost in transit in the TCP/IP network.

Byte-by-byte, the data stream arrives at the application (an upper layer to TCP) the same as it is sent. Characteristics of TCP include:

- Reliable Transport
- Connection Oriented
- Byte Stream Oriented
- Unstructured Stream
- Full Duplex

Reliable Transport refers to TCP ability to detect missing transmitted data and resend that data. Sequence numbers are used to indicate bytes sent and acknowledged. If a segment arrives and

the sequence number is not the next expected number, it will trigger a duplicate acknowledgement (ACK) to be sent back. How these duplicate ACK are handled is explained below.

TCP also has checksum integrity checking. If the checksum calculated upon arrival is not the same as the checksum that arrived with the segment, it is invalid and the segment is dropped. iFCP has additional end-to-end CRC checking in addition to TCP.

In contrast, UDP is a best effort protocol. If an UDP datagram were to be dropped in transit the UDP protocol would not be able to detect it. However, the application level could be performing functions similar to TCP and detect a missing UDP datagram; of course, this would depend on the application.

Connection Orientated means TCP sets up a conversation with the remote side and vice-versa (full duplex). This is commonly known as the "3-Way Handshake". This must complete prior to any data being sent.

TCP sends a segment and waits for an ACK. If no ACK or the wrong ACK is received the sender will start a process to recover from possible lost segments. Because of this behavior TCP must buffer the segments until it receives the proper ACK. There is a retransmit timer used to facilitate resending segments if an ACK is not received promptly. This timer is continually adjusted based on an algorithm and ACK RTT.

UDP does not set up a connection with the remote side. UDP sends a datagram, after which, the sender's responsibility ends. Simply, UDP hopes the datagram makes it to the other side. This is referred to as "Best Effort". UDP doesn't have to buffer its data. Once the data is sent there is no way to retransmit it again from layer 4.

Unstructured Byte Stream refers to transporting a stream of bytes verses discrete data fields. It is the job of the application to determine where data fields start and end within the byte stream, hence, the data is unstructured from the perspective of TCP. When many data fields must be transported they are organized into a byte stream by the application and sent down the OSI model to TCP (layer 4). On the opposite side the same byte stream will be presented to the application. It is the responsibility of the application to parse out those data fields.

The Eclipse places the iFCP frames into a byte stream for TCP to transport. TCP does not know or care where one iFCP frame starts and another ends. It is all just one long stream of data.

TCP operates equally in both directions at the same time making it a full duplex protocol. Typically, RDR applications are writing in one direction, therefore, there will be a heavy load in one direction and a light load in the opposite direction. The return direction will be carrying acknowledgements and Eclipse control traffic back to the originating side.

TCP Congestion Window Scaling (RFC 1323)

Originally, when TCP was introduced and accepted for use in the Internet it was used by these popular protocols: HTTP, FTP, SMTP and Telnet. 64 KB congestion windows (cwnd) were adequate. This was a time when most users had dialup access and T1 (1.544 Mbps) was considered a big link. Over time, bandwidth demands and link speeds have grown by orders of magnitude. Many folks have broadband connections to their homes and commonly organizations have links like DS3 (45 Mbps), OC-3 (155 Mbps), OC-12 (622 Mbps) and GE; and in rare cases 10 gigabits per second (i.e., OC-192 and 10 GE).

These days, 64 KB congestion windows are far from adequate. 64 KB windows are not capable of utilizing all the bandwidth in the link. This is a similar situation to having too few buffer-to-buffer credits for a FC link. The TCP cwnd must be large enough to fill a long fat pipe. The amount of data that can fit in a pipe at any one point in time is the product of:

$$\text{Bandwidth} \times \text{RTT} = \text{Data in the Pipe}$$

For example, an OC-3 link with a RTT of 20 ms can hold:

$$155 \text{ Mbps} \times 20 \text{ ms} = 3.1 \text{ Mbits or } 388 \text{ KB}$$

You can see how 64 KB cwnd would leave a void in the link causing poor utilization of bandwidth resulting in poor throughput. After the 64 KB is sent, no more data can be sent until the 64 KB is ACKed. This leaves a period of idle unutilized link.

RFC 1323 suggests using a scaling factor to increase the size of the cwnd. The Eclipse has always implemented RFC 1323 increasing the size of the cwnd to 8 MB. Why 8 MB? Assume a GE link; also assume that there are two TCP sessions. One session for each set of controllers connected across the link, controller set A and controller set B. This is common and best practice. The bandwidth is split equally between the two TCP sessions.

$$8 \text{ MB} \div \frac{1}{2} \text{ GE} = 128 \text{ ms RTT}$$

128 ms is typically the latency for data to travel nearly halfway around the earth and back. That means an Eclipse can fill a long fat pipe to almost anyplace on earth. If the bandwidth is less, the distance would be longer. If there are more TCP sessions, the distance would be longer.

Do not confuse cwnd size with buffer space on the Eclipse. Each direction has 96 MB of buffer space for TCP to retain segments until acknowledged.

Summary: Why the Eclipse implements RFC 1323 and large buffers:

- Large buffers: TCP must maintain segments until they are ACKed. If the segment has to be retransmitted it must pull it from memory, it will not be able to retrieve it from the application.
- 64 KB is the maximum cwnd without implementing RFC 1323
- Larger cwnd are required for Long-Fat pipes in order to utilize the entire bandwidth
- RFC 1323 can scale to 1 GB cwnd (for use with OC-192 or 10GE pipes)
- Currently, Eclipse cwnd scales to 8 MB per TCP connection. This cwnd size has been sufficient for all customer deployments including a link from Tokyo, Japan to Sterling, VA.

Below are descriptions on Fast Retransmit and Slow Start. A fundamental understanding of these algorithms will help you understand why the various adjustments to SO-TCP had to be made and under what circumstances. Ultimately, SO-TCP gives the Eclipse an enormous advantage over competing products. SO-TCP maintains high throughput in the face of a dirty link (high BER or congestion) and paths that produce Out Of Order Packets (OOOP).

Fast Retransmit

With standard TCP, for every out of order segment that arrives, a Duplicate (DUP) ACK is immediately returned. After the third DUP ACK, suggesting a missing segment, a fast retransmit is triggered immediately without waiting for the retransmit timer to expire.

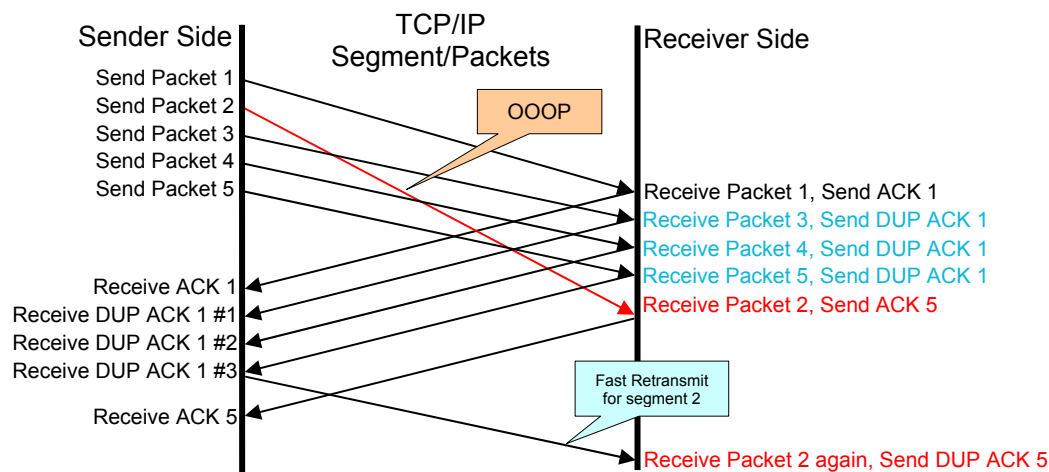


Figure 10: Fast Retransmit due to Out of Order Packet

What happens in many service providers networks is often a mystery. The reason a segment may arrive out of order is either because the IP packet carrying the segment got caught in transit, while others did not (think about a dual path and congestion on one of the paths), or because the segment was lost in transit for one reason or another. If the segment was lost in transit then Fast Retransmit will get the process started again quickly. If the segment arrives out of order then the Fast Retransmit wasted bandwidth and unnecessarily reduced throughput, see Figure 10 above.

After the Fast Retransmit is sent, fast recovery governs the transmission of new data until a non-DUP ACK arrives. Fast Recovery corrects the cwnd for the segments that arrived at the receiving side, which in turn generated duplicate ACKs initiating the process. It then starts to send additional retransmissions or new segments, which one depends on the degree of original packet loss.

The problem with standard Fast Retransmit/Fast Recovery for storage applications is that when initiated it causes the cwnd to be reduced, resulting in less throughput.

The Congestion Window is reduced in size when a Fast Retransmit is triggered.

- RFC 2581 (Congestion Algorithms) and RFC 3782 (Fast Recovery Algorithm)
- 3 or more duplicate ACK assumes that segment was lost in transit
 - Perform Fast Retransmit - congestion Window is decreased by about 50%
 - Perform Fast Recovery

McDATA Eclipse addresses this issue with EOSi 4.7 SO-TCP Optimized Fast Recovery.

Slow Start and Congestion Avoidance

With standard TCP, Slow Start and Congestion Avoidance are used to begin transmission into a network with unknown conditions. TCP slowly probes the network to determine available capacity. This avoids congesting the network with an inappropriately large burst of data. Slow Start is used when initiating traffic into a network or after repairing loss detected by the retransmit timer. Refer to Figure 11 below.

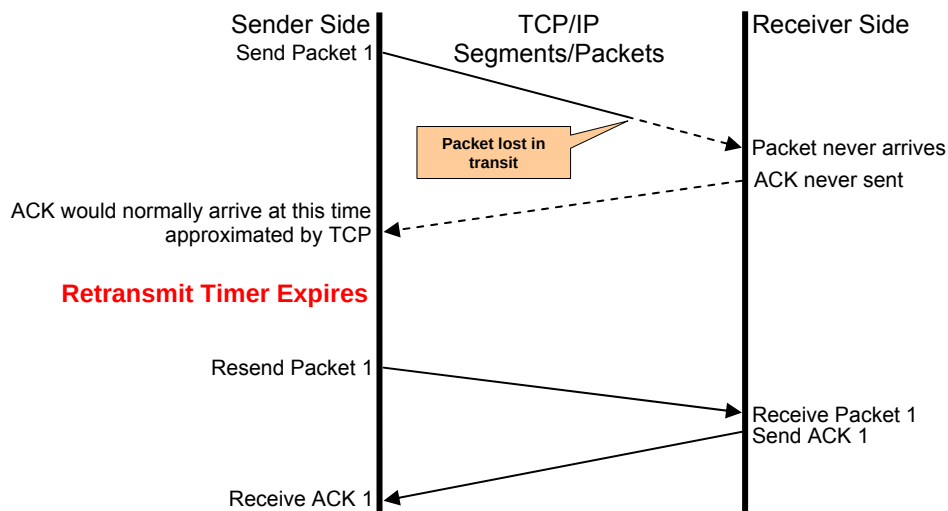


Figure 11: Retransmit Timer Expiration

Slow Start in the Eclipse is essentially an exponential algorithm that increases data flow relatively fast. Congestion Avoidance increases data at a very slow rate; refer to the Figure 12 below. The purpose is to recover throughput relatively quickly but not over congest the network again such that overall throughput suffers from repeated cyclical Slow Starts.

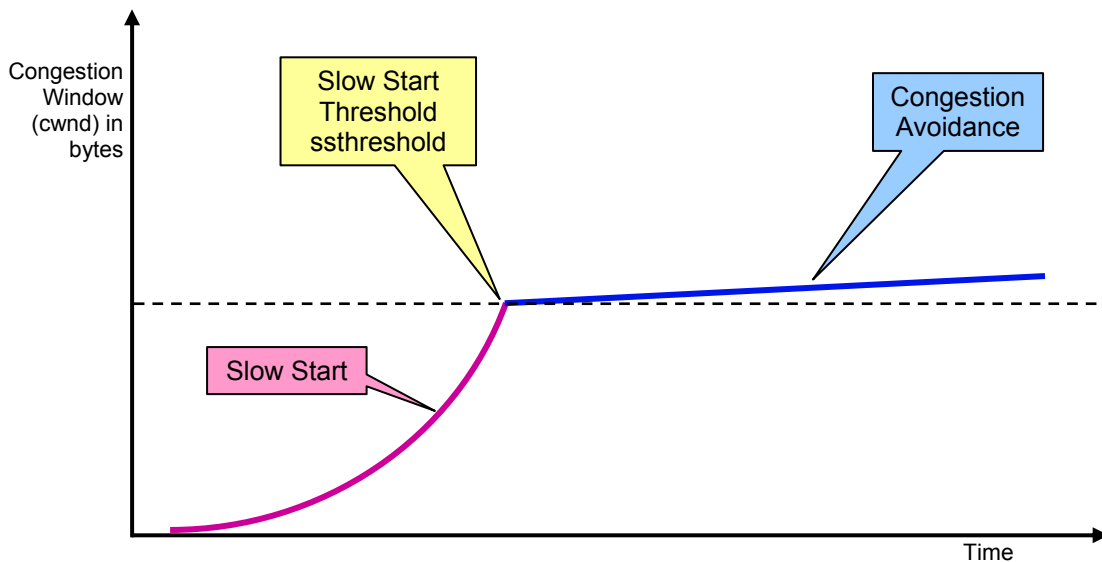


Figure 12: TCP Slow Start and Congestion Avoidance

There is a threshold that marks the change in transmission from Slow Start to Congestion Avoidance, this threshold is called ssthreshold. When the cwnd is less than ssthreshold, Slow Start is used and when it is greater than ssthreshold, Congestion Avoidance is used. The ssthreshold is not a constant value; it changes depending on network conditions and events.

Slow Start is initiated when the Retransmit Timer expires. The Retransmit Timer expires when Fast Retransmit fails to retransmit the missing segment. This can happen when no more segments are sent after the missing segment. It is the subsequent segments that generate the duplicate ACKs and trigger the Fast Retransmit. With standard TCP, if the segment does not get ACKed within 1 second plus RTT the Retransmit Timer expires.

Standard TCP cuts the cwnd to two times the senders maximum segment size (2 x SMSS). For an MTU of 1500 the SMSS is 1460 bytes. This turns out to be a very small amount of data for storage being sent over a long fat pipe.

SO-TCP: A Suite of TCP Enhancements

Storage Optimized TCP is a suite of TCP enhancements:

- Reorder Resistance
- Optimized Fast Recovery
- Quick Start
- Congestion Avoidance - Disable
- Optimized Retransmit Timer

Reorder Resistance

Problem: Some networks can produce Out of Order Packets (OOOP). An OOOP occurs when they/it arrive/s at the destination after packets that were subsequently sent. Obviously, this puts segments out of order. Standard TCP is built to reorder out of order segments, at least to a small degree.

OOOP density refers to how many contiguous packets are out of order. If the density is one or two packets out of place then TCP can efficiently handle this. If 3 or more packets are out of place then retransmits are generated causing inefficiencies. Most OOOP displacements occur within 32 frames. Causes of OOOP include:

- Some routers can cause OOOP rates as high as 2% due to their parallel switching architecture.
- Per Packet Load Balancing can cause OOOP. For example, if two paths are being used and one path experiences congestion holding up packets in buffers while the packets on the opposing path are not detained, the packets will be out of sequence. This should not be confused with flow based load balancing.

Solution: Fast Recovery causes the sender's congestion window to be cut by approximately 50% each time it is invoked. McDATA's SO-TCP improves overall throughput by tolerating more duplicate ACKs. Instead of invoking Fast Retransmit and Fast Recovery on the 3rd duplicate ACK the Eclipse can be configured to wait for 8 to 128 duplicate ACKs.

Keep in mind that Fast Retransmit will not be triggered if new segments are no longer being sent and the reorder resistance level has yet to be reached. Fast Retransmit can fill the holes in the bytes stream (missing segments) provided segments continue to be sent, which result in DUP ACKs. If a segment gets dropped and the reorder resistance is high (32 to 128) it will take no longer than 4 ms to trigger a Fast Retransmit from the reception of the first DUP ACK due to a "Backstop" timer, regardless of how many DUP ACKs have or have not been received. This may be necessary on some slower links, i.e., when 32 to 128 DUP ACKs can't be returned before 4 ms. With the Backstop timer the first hole will always be filled promptly. However, there may be multiple holes in the byte stream at the receiving side. If segments are no longer being sent subsequent holes require expiration of the Retransmit Timer to be filled.

Selective Acknowledgement (SACK) should be used in conjunction with Reorder Resistance. If a Fast Retransmit is not generated until a larger number of segments are received, statistically, additional segments may be missing. SACK will notify the sender of the missing segments and not just the next expected segment. This improves efficiency by sending only the exact segments needed resulting in less retransmitted data and faster recovery. As it turns out, by default SACK should be used, however, enabled is not the default.

For paths that are known to cause OOOP this feature will dramatically improve throughput. For paths that can't cause OOOP this feature should not be used.

Optimized Fast Recovery

Problem: Every time a Fast Retransmit is triggered, Fast Recovery occurs subsequently. Fast Recovery cuts the cwnd by approximately 50% and then initiates Congestion Avoidance.

Solution: McDATA SO-TCP improves throughput by limiting the cwnd reduction to 12.5% instead of 50%. Fast Recovery starts at the ssthreshold, therefore, it is the slow climb portion of the TCP curve. Less reduction allows faster recovery to previous data rates.

Keep in mind that storage traffic is not typically passing over a network in which there are many other contentious flows. There is no need to be the good network citizen backing off to the degree that standard TCP does.

If the storage traffic is utilizing a network in which it shares bandwidth with other applications, this feature should be evaluated further for interaction with other applications. The 12.5% is significant enough to allow congestion situations to dissipate. The other flows using standard TCP will continue to back off by 50%. Prioritized traffic using QoS mechanisms in the network and algorithms like WRED (Weighted Random Early Discard) will help to determine which flows will experience the most congestion events.

A congestion event is any event that triggers the Fast Retransmit or Slow Start algorithms.

Quick Start

Problem: When a Slow Start initiates TCP backs off too far for enterprise storage applications, in fact, it backs off all the way to approximately 1 segment size (SMSS). Most storage applications have dedicated links and don't require the protection of normal TCP slow start. Only shared networks need this protection.

Solution: McDATA SO-TCP improves overall throughput by taking into consideration that most enterprise storage applications will utilize dedicated links. Eclipse SO-TCP sets the initial cwnd value to 64 KB (≈45 segments) vs. standard TCP in which the initial window size is set to one. Furthermore, there is a reduction of only 25% of the Congestion Window when slow start occurs.

Recovery from a retransmit timer expiring is much faster with Quick Start. There is no drawback to using this feature provided that the link is dedicated. A shared network path may experience more congestion when using Quick Start because TCP will not back off as aggressively.

Congestion Avoidance - Disable

Problem: Congestion Avoidance is used after both Fast Retransmits and Slow Starts to slowly, very slowly, increase the rate of the data. On the Internet, immediately bursting to high levels after a congestion event would cause congestion again. In an enterprise, storage applications more often than not use dedicated links or circuits. If your storage application has end-to-end dedicated bandwidth Congestion Avoidance is not necessary. It is acceptable to resume the data at the peak levels experienced just prior to the congestion event. A congestion event is any event that triggers the Fast Retransmit or Slow Start algorithms.

There are reasons other than congestion why a segment may be lost. For example, a segment may be lost due to a bit error and the CRC failing upon arrival or some sort of hardware malfunction (failing optic). These issues result in packets being dropped.

Standard TCP grows the cwnd using the flowing formula. It is calculated with the arrival of every ACK. It is a very slow increase over time.

$$CWND = CWND + ((SMSS * SMSS) \div CWND)$$

Eclipse improved this by adding 128 bytes per ACK; however, this was still not sufficient in some cases.

Solution: McDATA SO-TCP improves throughput by allowing the user to disable Congestion Avoidance. After a congestion event occurs the data rate is returned immediately to prior levels.

Optimized Retransmit Timer

Problem: Standard TCP does not expire the Retransmit Timer until 500 to 1000 ms plus RTT. If there is a situation in which the next FC exchange can't start until the last exchange completes, FC frames will stop being sent by the storage system. If there are no FC frames being sent there will also be no segments sent by iFCP. If nothing is sent there is no way to initiate a Fast Retransmit. The first hole in the byte stream will be filled by the "Backstop" timer provided at least one DUP ACK was received to start the timer. Otherwise, lost segments must wait for the Retransmit Timer to expire. One second + RTT is too long to wait for storage applications.

Solution: McDATA SO-TCP improves throughput by restarting data flow faster. The Optimized Retransmit Timer now expires in 150 ms + RTT. The lost segment can be detected and corrected faster. Slow Start can be invoked quicker. Data can resume promptly.

Test Results: Eclipse vs. 9216i 14/2

In a practical setting, comparing the Eclipse SO-TCP to the Cisco MDS 9216i over a link with 1.0% packet loss, the McDATA Eclipse clearly outperforms the Cisco MDS 9216i. Please refer to Figure 13 below. 50 ms of latency with ±5 ms jitter was used. 32 KB block size, 1500 MTU and 50% reads and 50% writes. No compression or write enhancement algorithms were used.

The Eclipse outperformed the Cisco 9216i 14/2 across the board. The smallest percent advantage was 26.7% at 50 and 100 ms. The largest percent advantage that was not zero latency was 38.6% at 5 ms.

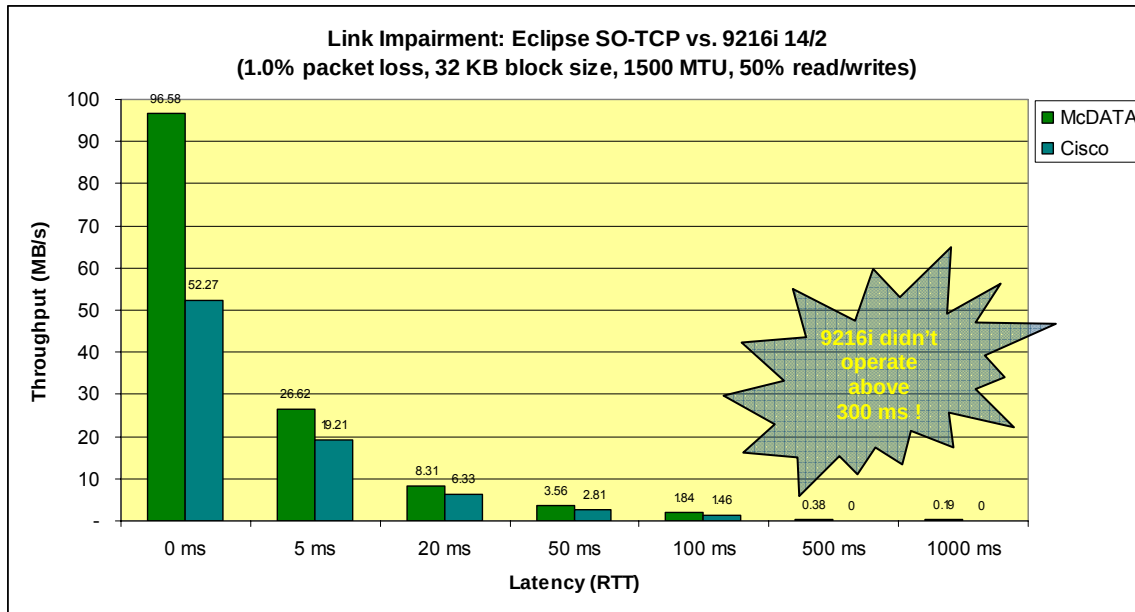


Figure 13: Link Impairments: Eclipse SO-TCP Outperforms the Cisco 9216i 14/2

Rate Limiting

When the ingress speed is greater than the egress speed on any network device buffers are going to fill. The rate at which they fill depends on the difference between the ingress and egress rates. The greater the difference the faster they will fill. Once the finite buffer space within the device becomes full, packets will no longer be accepted and dropped. Dropped packets result in TCP managing flow control (Congestion Avoidance). Flow control at layer 4 is extremely inefficient and causes throughput to diminish drastically (see SO-TCP section). Please refer to Figure 14 below.

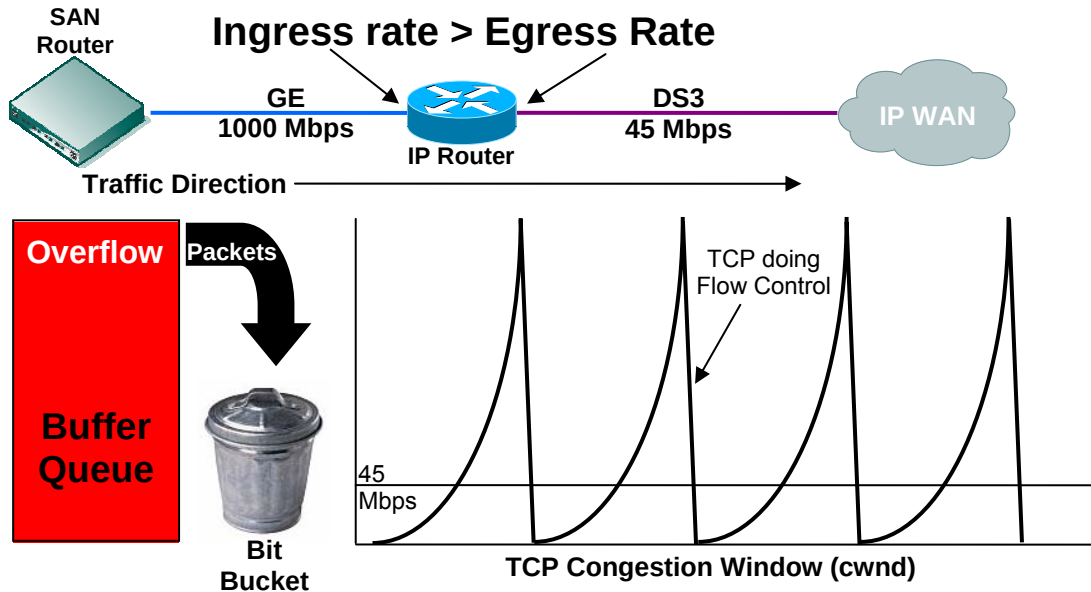


Figure 14: Ingress Faster than Egress

A considerable effort should be made to prevent packets from being dropped so that TCP does not have to administer flow control. This will result in greater overall data rates. The best and simplest way to do this is to ensure the ingress data rate is just slightly less than the egress rate of the slowest link in the path. In this situation, buffers will not fill up resulting in dropped packets, hence, TCP does not invoke flow control and throughput is sustained and maximized, refer to Figure 15 below.

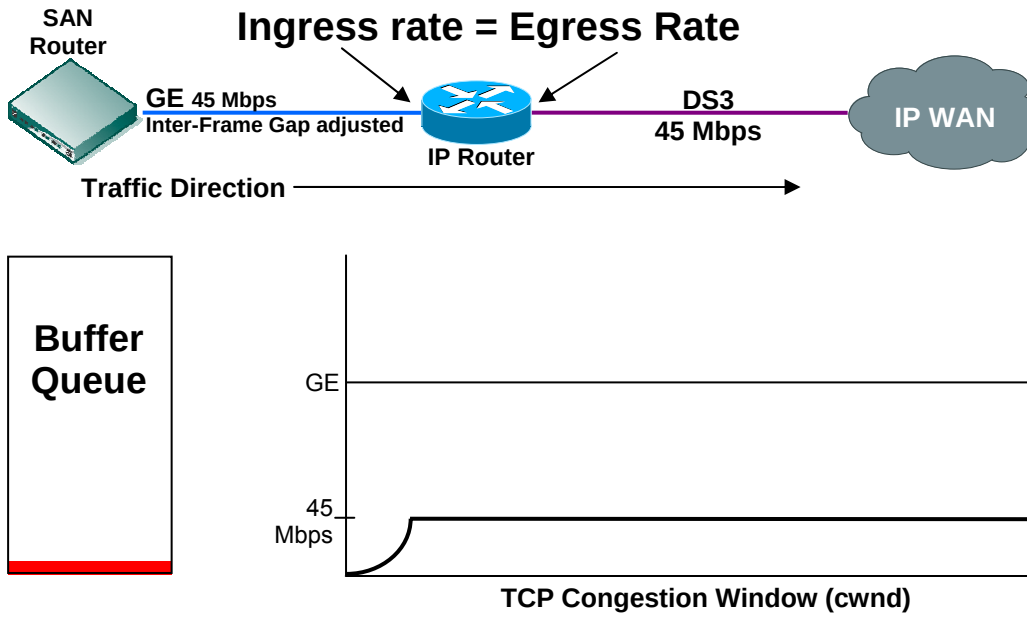


Figure 15: Ingress limited to Egress Rate

Efficient Architecture

The methodology used to place FC frames into IP packets varies among SAN extension products. McDATA Eclipse has diligently engineered the most efficient process on the market.

The process starts with encapsulating the FC frame. Most likely, this is similar among the competing products. This is done by prepending the FC Frame Encapsulation header (RFC 3643) shown in Figure 16 below. This same header is used for both iFCP and FCIP encapsulation. After the header is added the FC frame becomes either an iFCP frame or FCIP frame depending on the value in the protocol field.

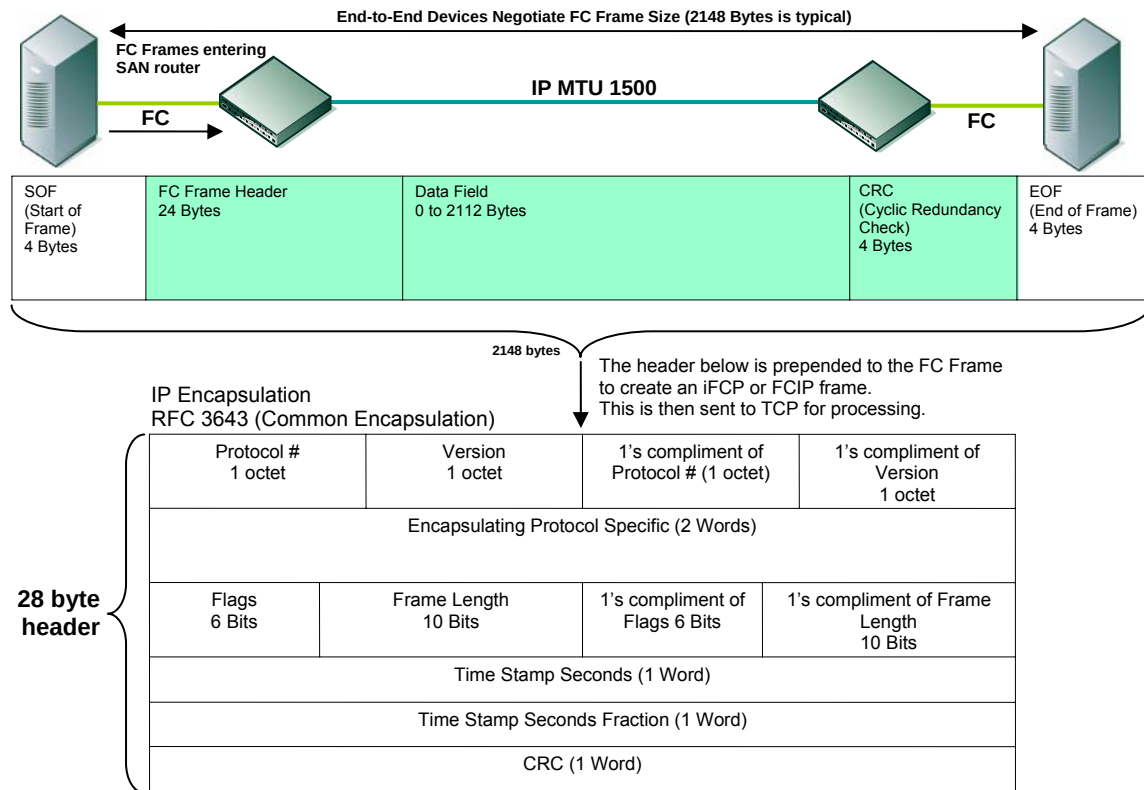


Figure 16: Common Encapsulation

The McDATA Eclipse method of FC frame to IP packet places the iFCP frames into a byte stream, refer to Figure 17 below. TCP does not care where the iFCP frames start or end. There is not a one to one relationship of FC frame to TCP segment or IP packet. The iFCP frames can extend across multiple IP packets, or there can be more than one FC frame in a single jumbo IP packet, either way it does not matter. iFCP frames don't have to start at the beginning of the segment, in fact, they can start arbitrarily anywhere within the segment. The Eclipse knows how to parse the iFCP frames out from the byte stream upon arrival at the remote side.

Compression is performed on that byte stream. Now there are a smaller number of bytes to place into the TCP segments. The segments can be filled to the maximum segment size (MSS) which equals:

$$MSS = IP\ MTU - TCP\ Header - IP\ header$$

For an MTU of 1500 the MSS = 1500 - 20 for the TCP header - 20 for the IP header, a total of 1460 bytes.

On the Eclipse every IP packet is filled to its MTU. MTU is the Maximum Transmission Unit and it is the largest size IP packet that can be accepted by the network. IP packets over the MTU will be dropped by the network and an ICMP error message will be returned to the originator indicating so.

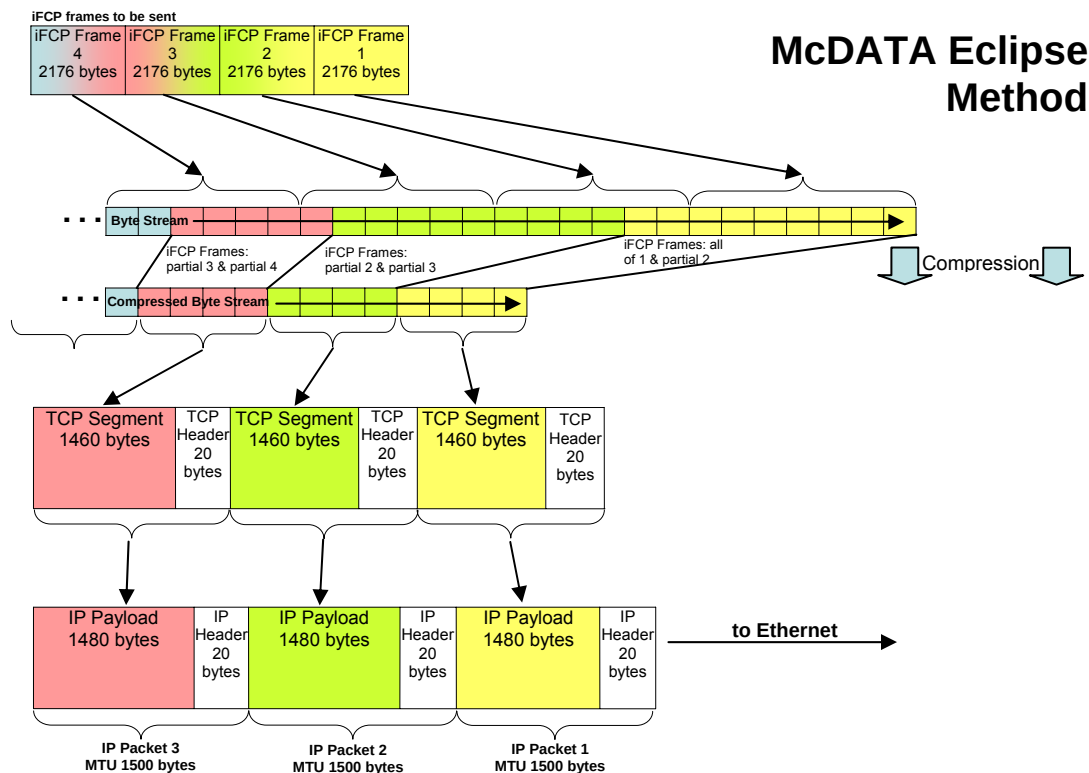


Figure 17: McDATA Method of FC to IP

Packets larger than 1500 bytes are referred to as Jumbo Frames. All networks today support 1500 byte IP MTU, it is the standard. Only some networks support jumbo frames and most service providers do not. The Eclipse supports jumbo frames up to 4096 bytes.

The motivation for sending packets at the MTU size is that it increases the ratio of overhead to data. Overhead is a tax that must be paid when sending data over a network. The object is to reduce the tax. Send more data and have less overhead. If the packet size increases while the overhead stays constant this accomplished that goal. Jumbo Frames were instituted for this purpose. Sending data in IP packets filled considerably less than the MTU essentially pays more tax because more packets have to be sent for the same amount of data, which means more overhead.

The Cisco 14/2 9216i method of placing FC frames into IP payload is much less efficient. The 9216i has a one to one relationship of FC frame to IP packet. If the packet is not completely full it can't be filled with a portion of the following FC frame. When 9216i is confined to the standard 1500 byte MTU, the FCIP frame must be sent using more than one IP packet. Remember that FC frames are 2148 bytes and FCIP adds 28 more bytes to that. The FCIP frame must be fragmented to be transported across the IP network. For each additional IP packet created by fragmentation more overhead is added. Every new packet adds 40 bytes for the TCP + IP headers. This increases the tax. The second IP packet gets sent partially full. More tax.

Furthermore, the 9216i supports Jumbo Frames up to 9K in size, but can't fill them more than the size of one FCIP frame, which is about 2K. Keep in mind, the Eclipse can fill a 4K Jumbo Frame regardless of the size of the iFCP frame.

The Cisco MDS 9216i uses a method of compression called IPcomp. IPcomp is performed on the IP packet (the Eclipse compresses the TCP byte stream). This exacerbates the tax problem. Now all packets are partially filled, not just the second one. 9216i 14/2 packets are filled only 25% to 50% of the MTU. This can be observed on a FCIP trace. Assuming approximately 2:1 compression the tax is now doubled. Please refer to Figure 18 below. This method of placing FC into IP packets is not efficient compared to the McDATA Eclipse.

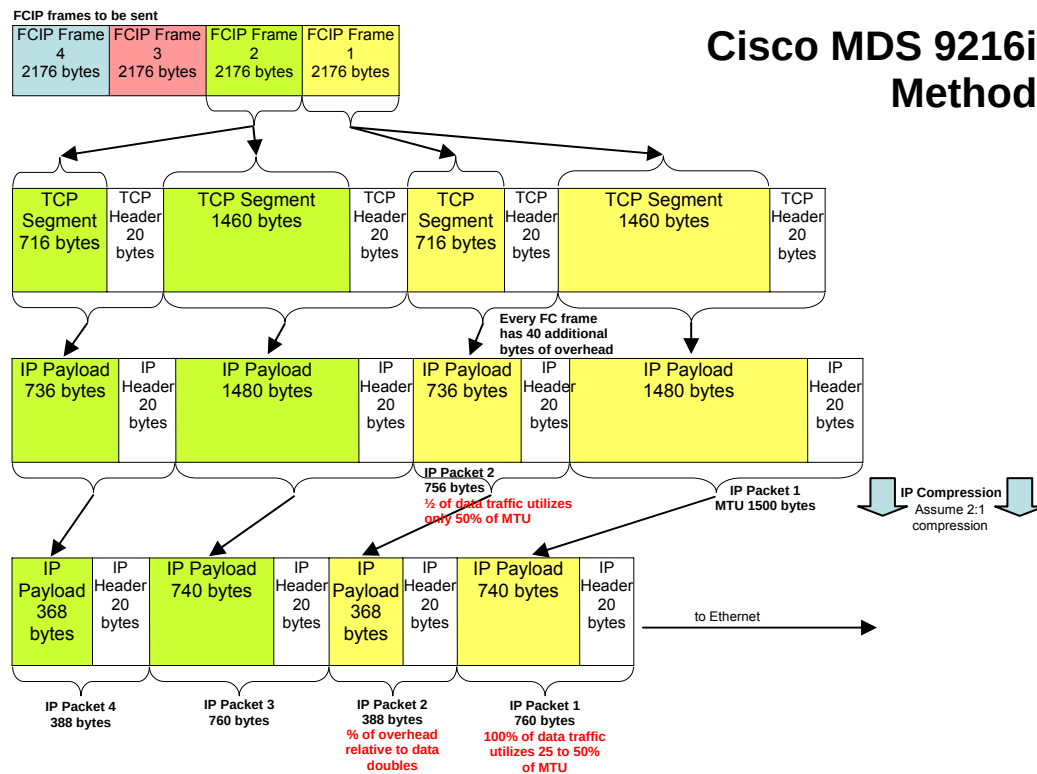


Figure 18: Cisco Method of FC to IP

The graph below, Figure 19, shows test results of maximum throughput of the Eclipse vs. 9216i 14/2 module. The Eclipse is faster at both 1500 and 4096 MTU across all latencies. At 1500 MTU and no latency the Eclipse is 20% faster (109.3 vs. 87.7 MB/s) than the Cisco MDS 9216i 14/2 with traffic flowing in one direction. See the next section below for comments on bidirectional traffic flow. At zero latency the difference in performance can't be attributed to latency. Data source and destination are identical for all tests. The difference in performance must be attributed to the 9216i 14/2.

The difference could be explained due to the fundamental differences in methodologies. When the 9216i is offered jumbo frames it becomes more efficient, as expected. Nonetheless, it continues to perform slower than the Eclipse by 2.6% (113.0 vs. 110.1 MB/s).

The test used a 32 KB block size and 100% writes. No compression or write enhancement algorithms were used. Eclipse SO-TCP was disabled and the link had no impairments. Plainly, the test was just FC to IP throughput. Iometer was used to generate the data.

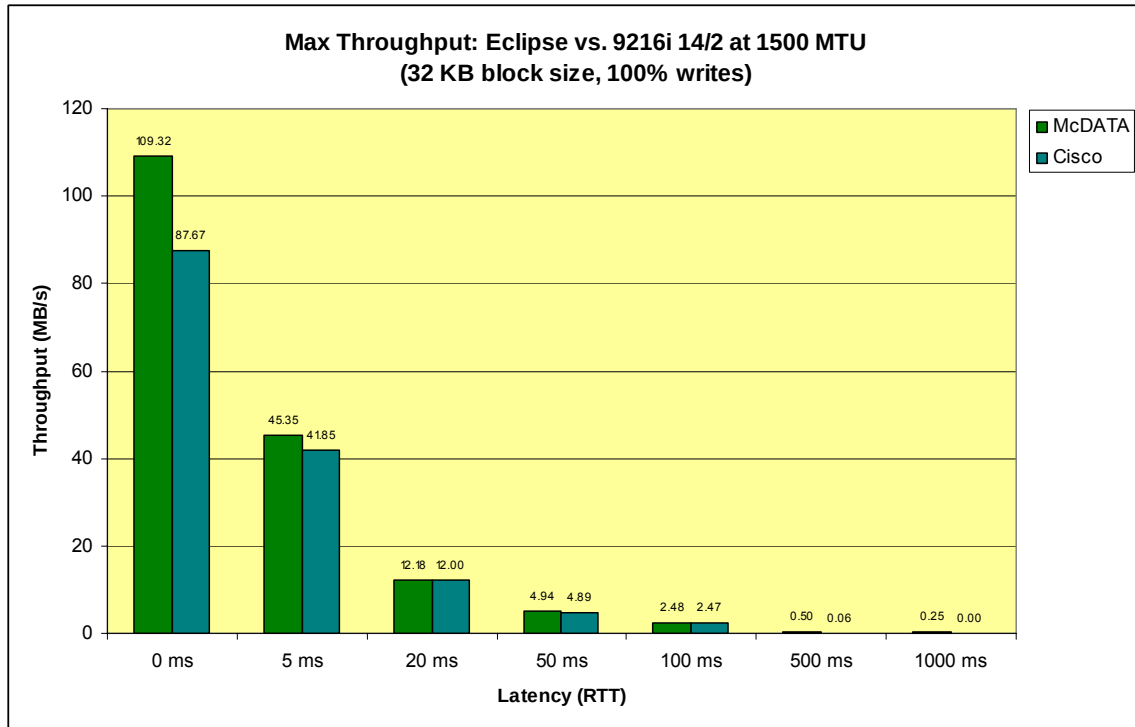


Figure 19: Eclipse vs. 9216i 14/2 Efficiency at 1500 MTU

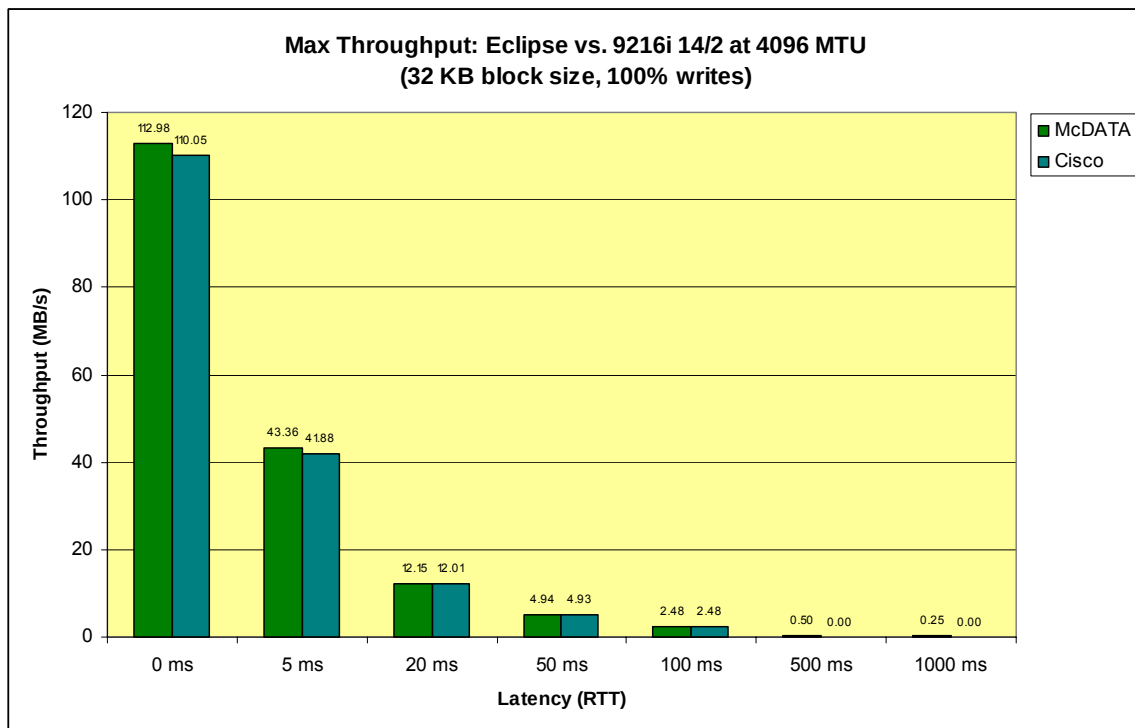


Figure 20: Eclipse vs. 9216i 14/2 Efficiency at 4096 MTU

Full Duplex Operation

The Eclipse has a non-blocking architecture. Multiple flows of data between various ports will all operate at line rate in both directions. The ability to operate in both directions simultaneously is called *full duplex*. While the Eclipse can operate with bidirectional traffic at line rate, the Cisco MDS 9216i 14/2 module can not. In fact, the 14/2 module can only operate at approximately 60% of line rate when bidirectional flows are present, refer to Figure 21 below. This attests to the robust architecture engineered into the Eclipse.

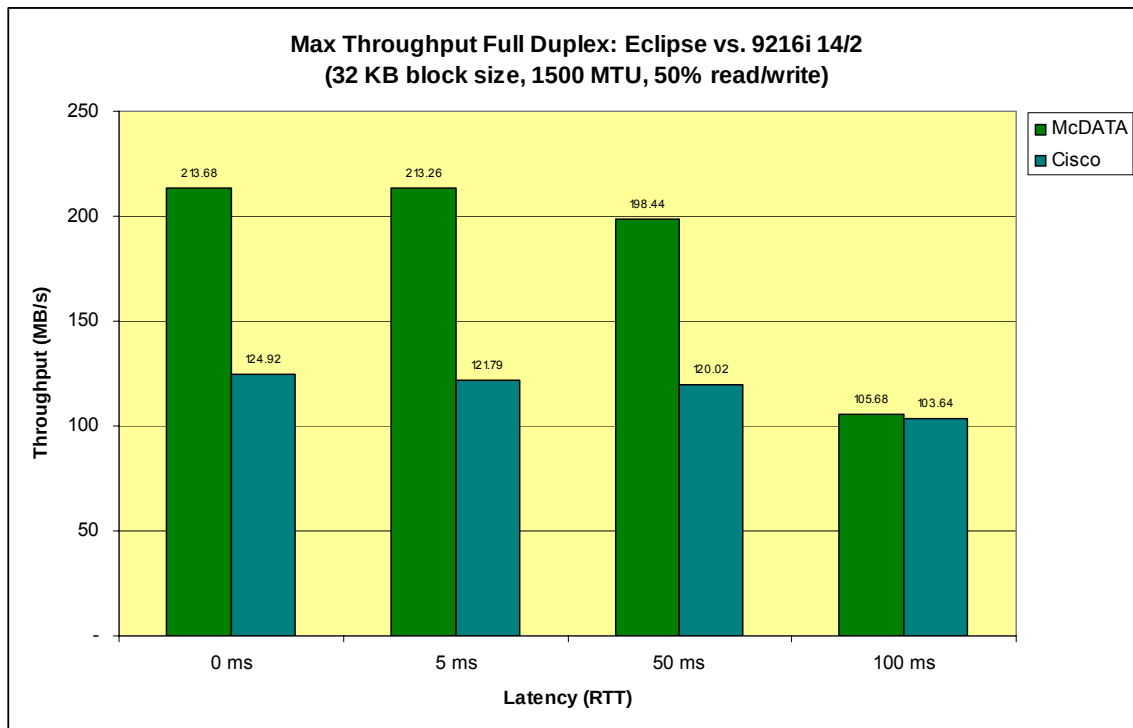


Figure 21: Full Duplex Data Rates

An example of an environment in which this may cause concern is server communications (read/write I/O) in a metro network (low latency and high bandwidth). Using the Cisco MDS 9216i 14/2 would result in poor performance by 40% compared to the Eclipse. Another example is bidirectional RDR. For these reasons, the Cisco MDS 9216i 14/2 module is not appropriate for many performance architectures. Remember, the response time of the MDS 9216i 14/2 is also higher.

At 100 ms the throughput between the products starts to converge. This is because of normal throughput degradation that occurs when features like FastWrite[®] are turned off and latency increases. Essentially, the Eclipse data rate is slowing down to match the 9216i 14/2 data rate. This can be prevented by using FastWrite[®]. Again, remember that the 14/2 Write Accelerate is not as effective as FastWrite[®].

Summary

In every instance, with the exception of some compression situations, the Eclipse outperformed the Cisco MDS 9216i 14/2 module. The Eclipse is a more mature product. Clearly, FastWrite[®] outperformed Write Accelerate. SO-TCP provides the strongest performance over links with impairments. The architecture and methodology of the Eclipse is superior in efficiency. Compression holds its own or is at a nominal disadvantage. Overall, the performance features of the Eclipse more than make up for any compression deficiencies.

Appendix A: Compression Algorithms

LZO (Lempel Ziv Oberhammer)

LZO is a dictionary based algorithm which searches for strings of characters duplicated within the block of data being compressed. The duplicated strings are removed from the data stream and replaced by an encoded string indicating the location of the earlier string. Characters which are not part of the duplicated string (called “literals”) are passed directly to the output with special encoding to distinguish them from duplicate string encodings.

The LZO implementation performs the compression over a block of data generating a compressed data block, which is self contained. In other words, all of the information needed to decompress the data is present in the compressed data. There is no history maintained either by the receiver (for decompression) or the sender (for compression).

This algorithm performs similar to *LZO with History* except that this algorithm can accommodate more TCP sessions, up to 64. This algorithm is recommended when many TCP sessions are going to be used.

LZO is an open source algorithm.

LZO is used for both the software and hardware based algorithms. HW LZO is used for links faster than Fast Ethernet (100 Mbps), i.e., OC-3 and faster.

Fast LZO with History

The *Fast LZO with History* algorithm uses the LZO described above with an additional history cache. A history cache is maintained on each side of the link and is used to more effectively compress and decompress data. This algorithm has an average throughput increase of approximately 12% over LZO. The compression ratio is the same as LZO.

The maximum number of TCP sessions is 8.

Fast LZO with History can accommodate Fast Ethernet links (100 Mbps).

LZO with History

This LZO algorithm incorporates history and Huffman encoding. Because of the added encoding this algorithm utilizes more computing resources and results in a lower maximum bandwidth that can be accommodated. The combination of history and Huffman encoding offers an average of approximately 24% higher compression ratio over LZO.

8 is the maximum number of TCP sessions.

This algorithm is recommended for DS3 (ATM or Clear Channel) and E3 (ATM or Clear Channel).

Deflate

Deflate is a dictionary based algorithm combined with Huffman encoding. The algorithm maintains history to achieve a higher compression ratio. In addition, a hash table is used in performing the string searches and is saved as part of the state.

Deflate has approximately 53% higher compression ratio over LZO.

8 is the maximum number of TCP sessions.

Deflate is a processor intensive algorithm with the highest compression ratio. Because of the limits on processing resources this algorithm is restricted to Ethernet (10 Mbps) speeds or less.

Appendix B: RDR Block Sizes

EMC SRDF

Depending on the Symmetrix ucode version and model number, the Symmetrix uses either use 1k or 2k FC frames. Enginuity 5671 uses 2k frames.

All versions of Symmetrix 6 SRDF transfer no less than 4,160 bytes, which is one sector (4,096 bytes), plus 64 bytes of additional internal information. No more than 32,832 bytes, which is one full track (8 sectors) plus 64 bytes of additional internal information. This is data per FC exchange.

The amount of data transfer is completely dependent upon the host. For example, if a write from a host updates only one sector, 4,160 bytes will be transferred. If the write from the host was to span 2 sectors, then the data transfer would be 2 sectors (8,192 bytes) plus 64 bytes of additional internal information for a total of 8,288 bytes. As a result, each exchange will always consist of some number of 1k or 2 k frames as well as a single frame with a 64 byte payload.

Symmetrix 7 can transfer up to 65,600 bytes per exchange due to the increased track size associated with the Symmetrix 7.

EMC MirrorView

For all versions of MirrorView, the sizes of the updates are completely dependent upon the size of the write sent by the host.