Network Storage System Testing and Optimization

Firas Charrouf

Supervisor: Dr. Mike MacGregor

University of Alberta

Electrical and Computer Engineering Department

Edmonton, April 2008

Sumary

1- Objective

This project aims to test and optimize the performance of the YottaYotta distributed storage system over two geographically separated sites.

The setup for the actual tests will consist of two sites with two YottaYotta systems in each site. It is envisioned that the two sites are connected via DummyNet for WAN emulation. The bandwidth and latency connecting the two sites will be adjusted according to the test requirements.

The project can be summarized through the following steps:

- Familiarization with product:
  - Introduction to YY CLI commands.
  - Introduction to YY firmware commands.
  - Fiber Channel switch configuration.

- Block level testing
  - Long sequential reading.
  - Long sequential writing.

- Data analysis
  - Front-end (this term will be explained later on this report) throughput.
  - WAN traffic throughput.

2- Goals

After a familiarization with the products and technologies involved in the testing environment, the proposed setup using the YottaYotta network storagers (GSX 3000) was implemented and the first performance tests could be started.

We were able to implement and gather throughput measurements for the following performance tests[1]:

Test	Configuration Involved
Large Sequential Read	Default Settings
	Pre-fetch
	Volume Stripe
	TCP settings
	COM window
Large Sequential Write	Default Settings
	WOF (Write Order Fidelity)
	COM window

All the tests were implemented using a T3 link under 0, 50 100 and 150 ms of round trip time latency.

In addition, we were able to optimize our YottaYotta network storage system setup while under low bandwidth regime (T3 link) to archive higher performance when compared with the same setup under default configuration. The tuning strategies used to optimize performance under low bandwidth regime consist mostly in adjusting optimal values for some key parameters (e.g. pre-fetch, network interface buffer size, etc) and maximizing WAN throughput over the T3 link by tuning COM[2] port parameters to avoid TCP window collapsing.

It was defined that to proceed with testing with PolyServe, a YottaYotta setup configuration that yields WAN reading and writing performance of at least 70% of the full T3 bandwidth over 5000 km of distance (50 ms of round trip time latency) should be archived. Therefore, this project does not cover performance testing with PolyServe stretch file system interacting through the YottaYotta layer with DR1 (distributed RAID 1) as proposed in the former project plan because block-level performance across the WAN at this point was still unacceptably low.

[1] The performance tests mentioned above will be explained with more details in the next sections of this report.
[2] COM is the YottaYotta proprietary protocol for inter node communication and will be explained later with more details.

3- Methodology

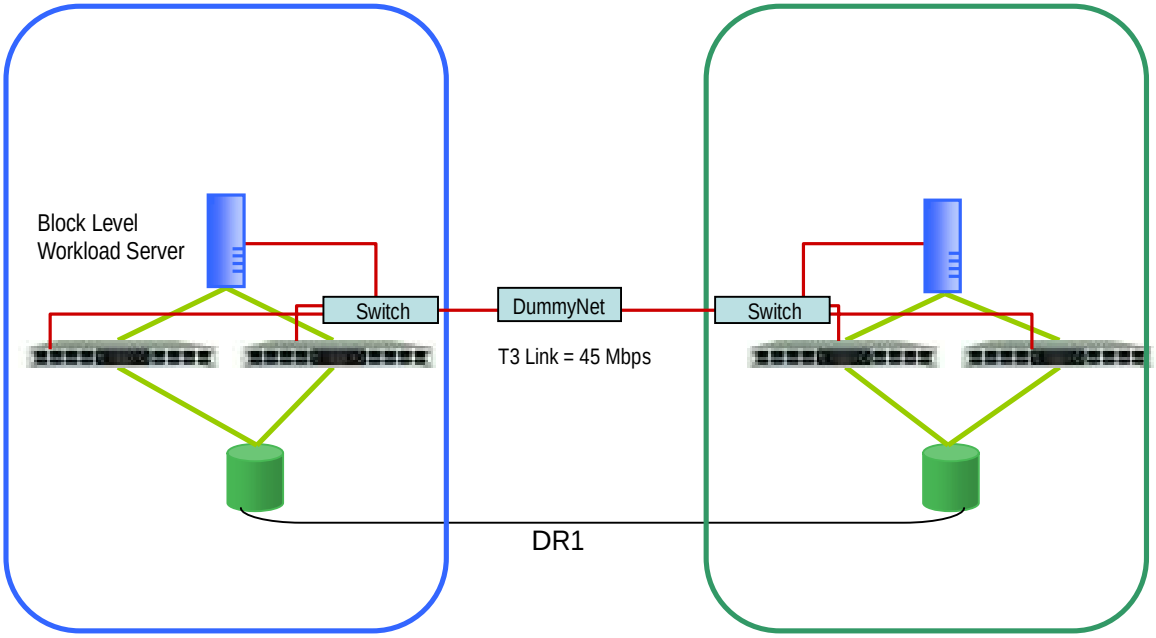
3.1- YottaYotta Network Storager

YottaYotta's GSX 3000 is a storage network appliance that resides between the data paths between servers and storage, providing scalable geographic storage replication, continuous data availability, and geographic SAN (storage area network) clustering.

GSX 3000 nodes can be clustered, with each additional GSX 3000 providing additional bandwidth resources, cache memory and performance. Our setup consists of a 2X2 distributed clustered system which means two GSX 3000 nodes on both two sites.

3.2- Network Configuration

The figure below represents our logical network topology:



The block level workload servers are used to implement block level writing and reading operations.

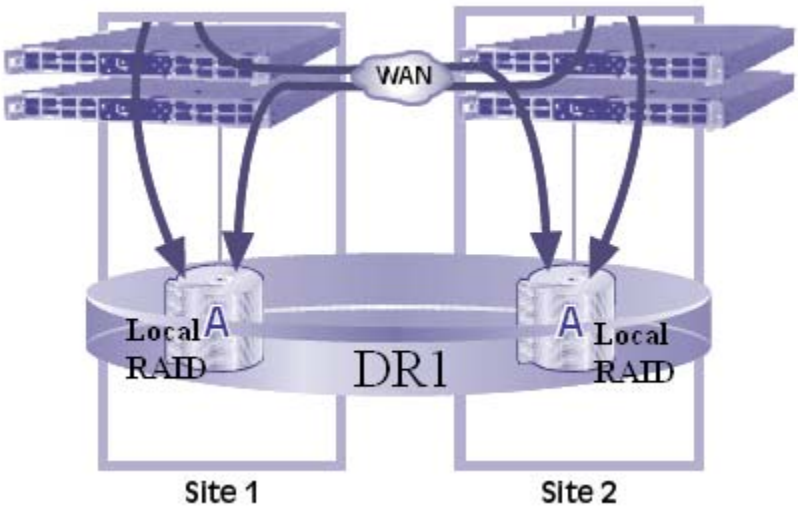
On both sites, the block level workload server is connected to the YottaYotta network storage nodes via fibre channel and each node is connected, also via fibre channel, to locally attached storage.

The communication between sites is performed by the YottaYotta network storage nodes through their Gigabit Ethernet interfaces.

DummyNet is used between the sites for WAN emulation (T3 link) and the network storagers are configured to create a distributed RAID 1 (DR1).

DummyNet is a flexible tool for bandwidth management and for testing networking protocols. It works with IPFW (IPFW is one of the FreeBSD firewalls) to simulate the effects of network bandwidth limitations, propagation delays and packet loss by intercepting packets using IPFW rules.

A DR1 is a RAID 1 which spans sites in geographically distributed (multi-site) system; it is not limited to one site. It is created with previously local RAIDs on each desired site as shown in the figure below:



3.3- Large Sequential Read Testing

The throughput performance for reading operations will depend on the network storagers cache state:

- Cold cache: Cache is empty and the disk is being hit. Tests on this state essentially measure how quickly the index data can be located and read from the disk
- Warm cache: Local network storage node cache contains the data which will be read in memory. It essentially tests how quickly the index data can be located and read from the local network storage node cache.
- Remote warm cache: The network storage nodes on the remote site contains the data which will be read cached in memory. It essentially tests how quickly the index data can be located and read over the WAN (T3 link) from the remote network storage node cache.

If the workload server of a site is reading data through the network storage nodes, the nodes of the site where the reading operation is being performed will, first of all, check

for this data on their own cache, and then if this data is not cached there, they will do the same on the remote cache (network stogeragers' cache of the other site). Finally, if the data is not there either, it will be read from the locally attached storage.

Therefore, the implemented large sequential reading tests were executed under cold, warm and remote warm cache state. To be able to do so and to collect the whole reading operation performance behavior, the following procedure was performed:

(1) Clean the cache of the servers and netstoragers (YottaYotta GSX 3000).
(2) Enable I/O performance monitoring[3] on netstoragers.
(3) Reset I/O performance monitoring statistics on all network storage nodes.
(4) In one of the sites, run dd[4] (single-threaded) on the workload server to read 1GB of data. Once the reading operation is completed, collect throughput statistics from dd output. That should represent the cold cache throughput performance.
(5) Collect I/O performance monitoring from netstoragers. This should provide the statistics referent to the cold cache reading cycle.
(6) Clean the cache on the workload server.
(7) Execute dd on the same server again to read the same 1 GB of data that was read in Step-4. This should give the warm cache throughput performance.
(8) Collect I/O performance monitoring from netstoragers. This should provide the statistics referent to the warm cache read cycle.
(9) On the other site's workload server, execute dd to read 100 MB of data.
(10) On the DummyNet machine, capture the WAN traffic using tcpdump.
(11) Once the reading operation is completed, collect throughput statistics from dd output. That should represent the remote warm cache throughput performance.
(12) Collect I/O performance monitoring from netstoragers. This should provide the statistics referent to the remote warm cache reading cycle.

The above process is performed three times to ensure that data results are consistent.

In addition, remote cache testing will depend on the network round trip time (RTT) latency. Consequently, all remote cache tests are implemented under 0, 50, 100 and 150 ms of RTT latency.

This large sequential testing is implemented initially using the GSX 3000 default settings. However, to archive enhanced throughput, some parameters and configuration were tuned to optimize performance. These parameters and configuration are summarized below:

[3] The network storager can generate statistics by creating "performance monitors" to monitor various aspects of the system's performance. This feature allows us to determine how a given port or volume is being used, how much I/O is being processed, CPU usage, and so on.
[4] dd is a common UNIX utility whose primary purpose is to copy a input to a output, applying any specified conversions.

Pre-fetch: It is a parameter that can be set on the network storage nodes to determine the amount of data pages to be loaded ahead from either the disk or remote cache to the local cache and improve large sequential operations.

Volume stripe: Using Linux MD, four exported YottaYotta volumes (single accessible storages created from the DR1) are striped to form a single MD RAID 0 and provide improved performance.

TCP buffer size and number of connection: A suitable number of TCP connection and an optimal buffer size, which depends on the RRT latency in use, can improve the traffic throughput over the T3 link.

COM window: COM bandwidth management is a YottaYotta dynamic tuning mechanism that supports throttling the number of outstanding messages for busy links and expanding the number of messages sent when response times for message acknowledgements are met. Provided messages are acknowledged within an acceptable time limit, the number of messages sent can be increased until link saturation is achieved (similarly to the TCP flow control mechanism). Fixing a static optimal COM window size can limit the amount of data transmitted by the sender providing a stable WAN throughput and avoiding the TCP window to collapse. In addition, it will avoid the sender to be increasing its sending rate until it exceeds the capacity of the network and packet loss occurs.

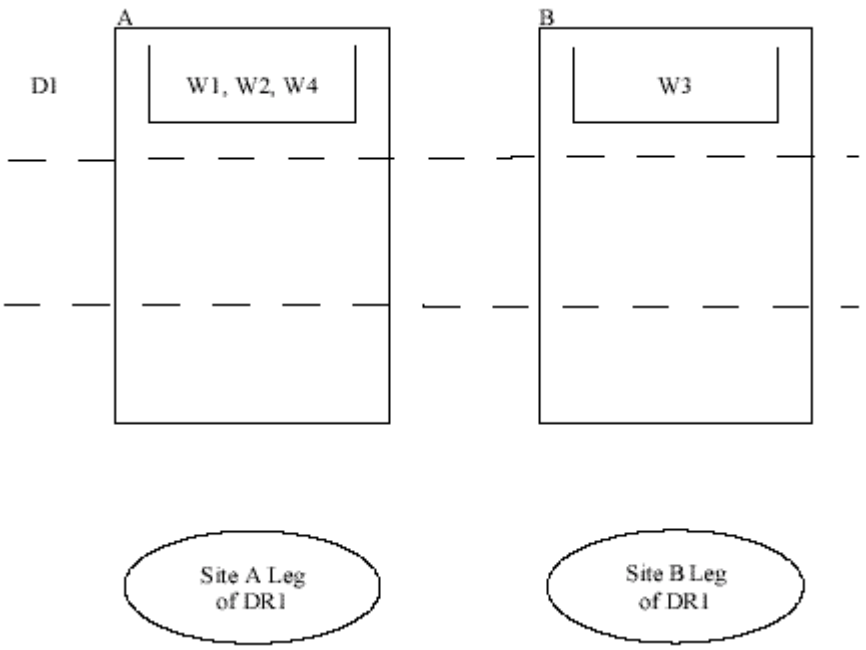
3.4- Large Sequential Write Testing

In opposite to reading operation performance, the writhing performance does not depend on the network storage nodes' cache state. The process of writing on asynchronous cache mode (default mode) start with the workload server writing data to the network storage nodes' cache of a site. Then, acknowledgements are returned immediately upon data receipt by the GSX 3000s of this site and the cached data is written to the disks (local and remote storage systems) later. The "dirty" data is kept in cache and periodically (each 60 seconds by default) flushed to the disks.

The throughput performance for writing operations will depend on whether the Write Order Fidelity (WOF) functionality is in use or not.

The following example (from a YottaYotta document, USPTO Performance Investigation, prepared by Gary Oikawa, Greg Czarniewski and Andrew Feldman) will clarify the process involved when implementing writing operations with WOF:

"The diagram below shows a two site system, with one leg of a DR1 at each site. Initially, each site is collecting writes into the currently open delta D1, shown below by the open box at the top of each site. The writes are collected locally at each network storage node at a site, subject to the usual cache coherency protocols.



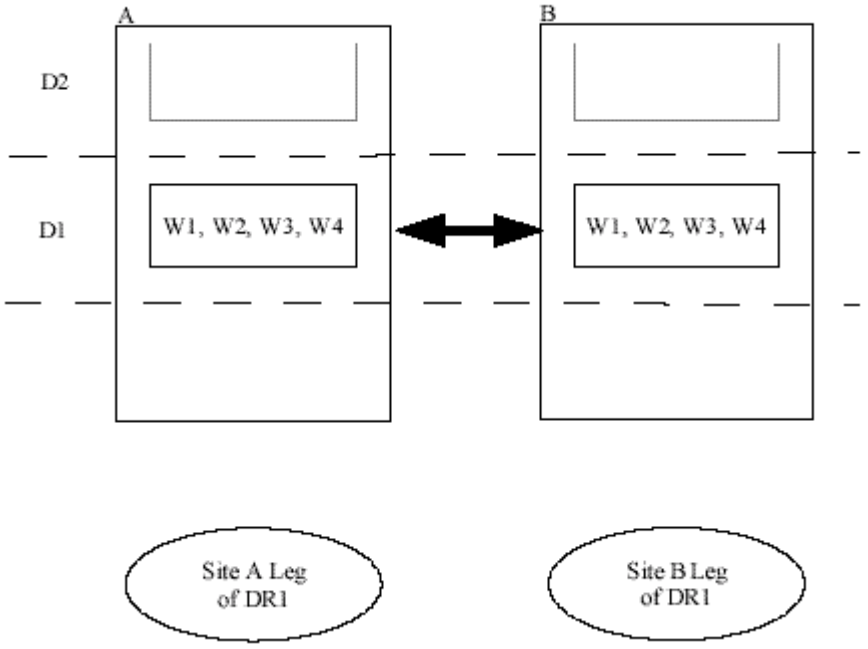
Eventually the system decides to close the current delta. It may do so for several different reasons:

- A user-configurable timer has expired. The administrator could use such a knob to bind the amount of data that will be lost in the event of a failure.

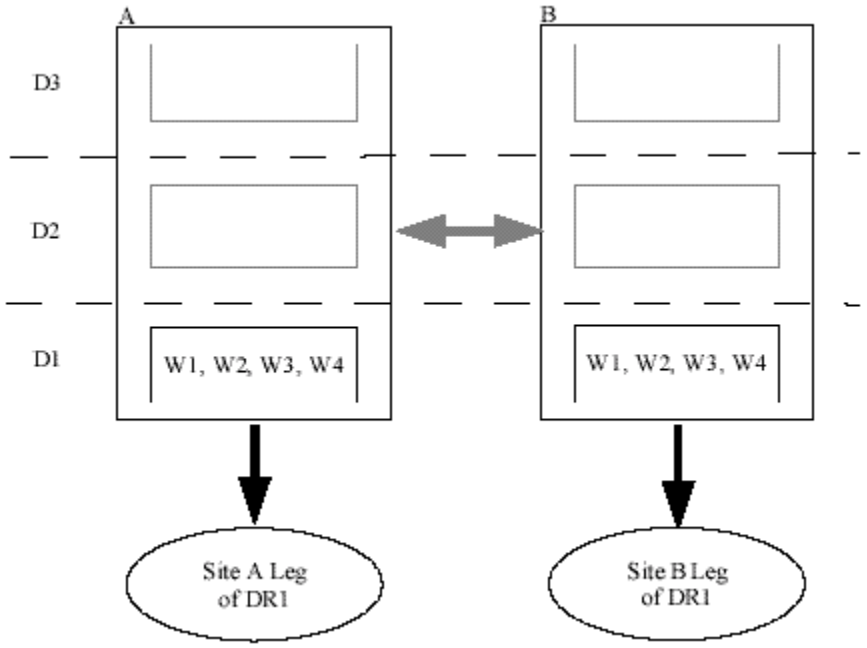
- Running out of resources on one or more network storage nodes. One of the nodes collecting the delta may decide it needs to close the delta early.

The system synchronizes the closing of the current delta across the sites so that the boundary of the delta respects dependent write ordering consistency. That is, the edge of each delta defines a consistent view of storage. Once the current delta is closed, a new delta is immediately opened to collect new writes from the application.

The system now begins to exchange the partial deltas collected at each site across the inter-site link so that each site has a complete copy of the closed delta. This step is shown below:



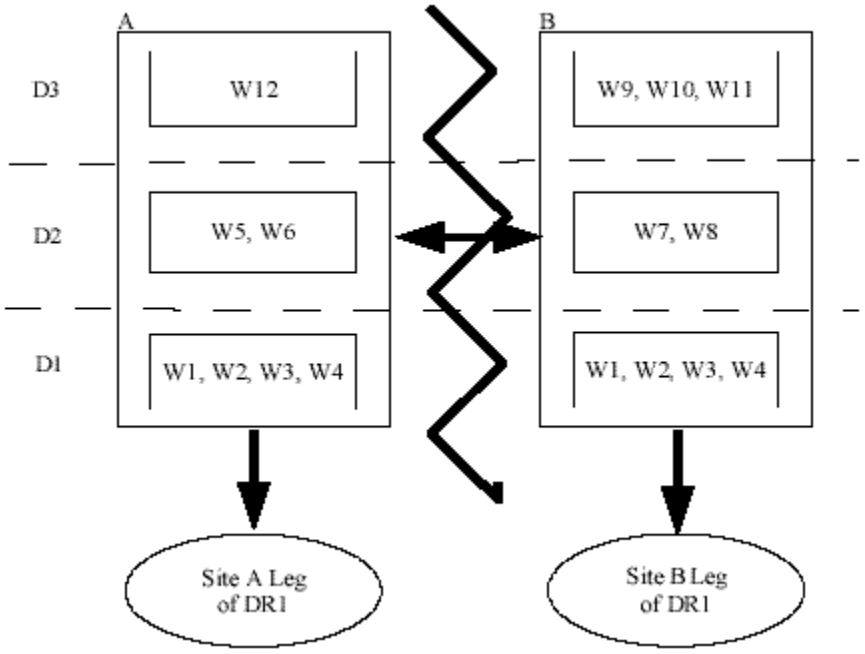
As each site is accumulating a complete delta, it also ensures that the delta is protected against local failure before it can be written to storage, or during writing to storage. Once both sites have exchanged their partial deltas, and made their local copies of the delta safe against failure, each site begins applying the delta to the local leg of the DR1 independently. When both sites agree to apply the delta to local storage, the delta becomes the next recovery point in case of failure.



Applying the delta to each leg of the DR1 is not an atomic operation, which means it can be interrupted by various kind of failure. Depending on the failure, it may be necessary to restart the process of applying the delta.

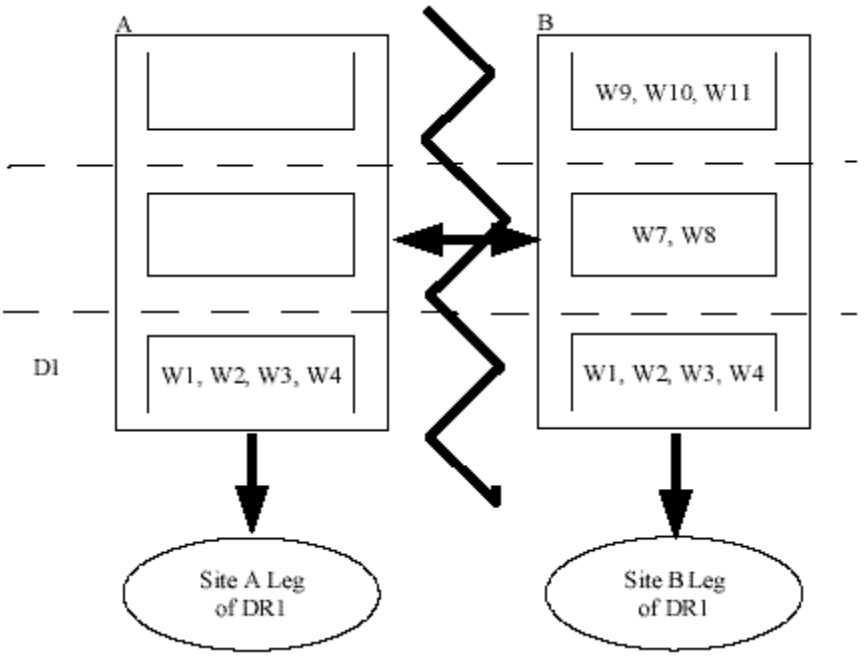
The final issue to discuss is the behavior of this WOF implementation in the presence of split brain. The term "split brain" refers to a distributed system losing communication across a WAN link, such that one or more sites is unable to communicate with one or more other sites. Subsequently, complete system knowledge is no longer completely distributed amongst all blades and sites. The key problem associated with split brain is the handling of the situation in terms of continuing operation, avoiding data loss, and re-establishing data consistency upon site recovery or link recovery.

The figure below shows all the processes described above proceeding concurrently when a link failure happens. Writes are being collected, a closed delta is being exchanged, and a committed delta is being applied to disk.



When the link fails, front-end[5] I/O will be suspended as usual, so no more writes are collected into D3. Fragments of D2 cannot continue being exchanged, and each site will continue to hold dirty data for that delta until the link heals or the administrator declares one site the "winner". However, the committed delta D1 can continue being applied to the legs. Note that the legs of the DR1 will be identical after this completes, even if the link is down.

[5] Between workload server and network storage nodes.



If the link failure is temporary, then normal operation resumes when the link heals. The figure above shows what happens when administrator declares site A the "winner". In this case, we have lost dirty data because site B holds the only copies of some writes that the system has acknowledged. Site A now discards the contents of deltas D2 and D3 and resumes servicing application I/O. The contents of storage, as seen from site A, are consistent as of the last successfully exchanged and committed delta D1. While writing only to the local leg of the DR1, site A is updating the bitmap logs so that the two legs of the DR1 can be synchronized when the link heals.

The WOF implementation is a substantial change to the flow of the workload server writes from the front-end through to back-end[6] physical storage. Several aspects of the implementation affect the net performance of the system as perceived by the host:

- Delta closure: All blades servicing a WOF group must synchronize to close the current delta. This has obvious performance implications, especially for multi-site configurations. However, it should be noted that during this synchronization interval, reads will carry on normally, and incoming writes will receive data from the host, but will not be acknowledged until the delta closure is complete.

- Efficient use of WAN link: By collecting changes into a delta, we should be able to stream these changes across the WAN link more efficiently than the smaller individual write operations.

- Fault-tolerance of committed deltas: Applying a committed delta to storage is not an atomic operation, which means it is vulnerable to failures. We must decide the

[6] Between network storage nodes and locally attached storage system.

> failures against which we protect committed deltas not yet applied to storage against. This is equivalent to choosing how strong we make our consistency guarantee. There are several implementation options in this area, each with different performance tradeoffs."

During large sequential write testing, the following procedure was executed to obtain throughput performance information:

(1) Clean the cache of the servers and netstoragers (YottaYotta GSX 3000).
(2) Enable I/O performance monitoring on netstoragers.
(3) Reset I/O performance monitoring statistics on all network storage nodes.
(4) In one of the sites, run dd (single-threaded) on the workload server to write 100 MB of data.
(5) On the DummyNet machine, capture the WAN traffic using tcpdump.
(6) Once the writing operation is completed, collect throughput statistics from dd output.
(7) Collect I/O performance monitoring from netstoragers. This should provide the statistics referent to the writing cycle before dirty data is fully flushed to disks.
(8) Wait around 5 minutes while dirty data is flush to disks (local and remote storage systems).
(9) On the DummyNet machine, stop the WAN traffic capturing using tcpdump.
(10) Collect I/O performance monitoring from netstoragers. This should provide the statistics referent to the writing cycle after dirty data is fully flushed to disks.

The above process is performed three times to ensure that data results are consistent.

This large sequential testing is implemented initially using the GSX 3000 default settings. However, to archive enhanced throughput, the COM window size was set statically to optimize performance as will be shown later on this report.

4- Results and Analysis

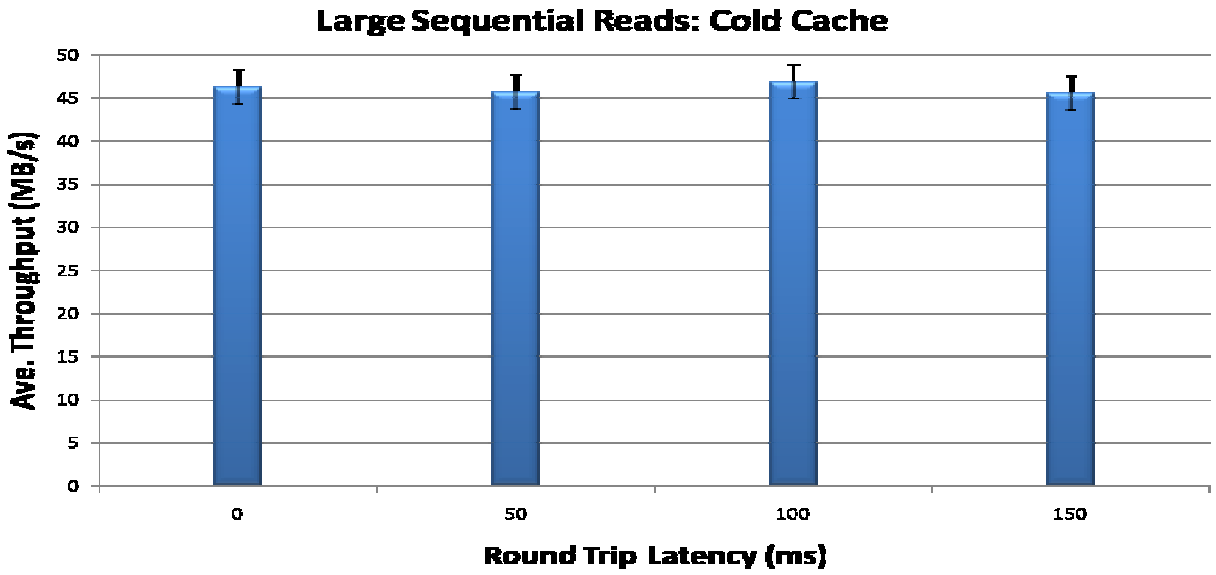
The following performance tests were implemented and I/O and WAN traffic throughput measurements were collected and analyzed:

Test	Configuration Involved
Large Sequential Read	Default Settings
	Pre-fetch
	Volume Stripe
	TCP settings
	COM window
Large Sequential Write	Default Settings
	WOF (Write Order Fidelity)
	COM window

The above tests were implemented using a T3 link under 0, 50 100 and 150 ms of round trip time latency and will be explained in detail the next sections.

4.1- Large Sequential Read Testing

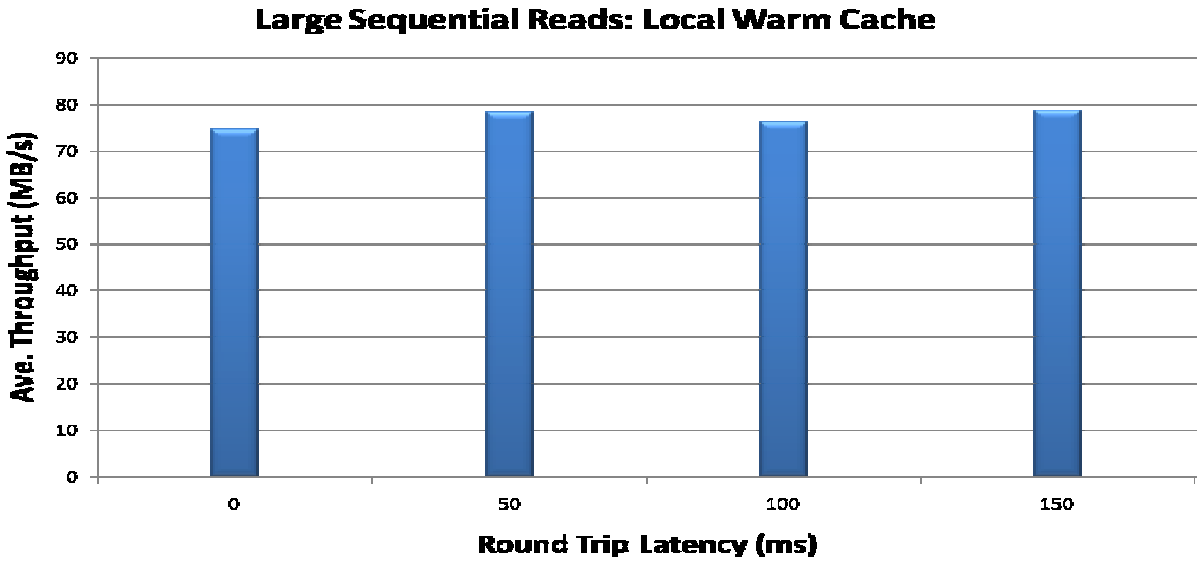
The first large sequential read testing were performed using the default configuration on the YottaYotta network storage nodes and following the procedure described previously on the Methodology chapter.



In the above graph for cold cache results, it is possible to notice that throughput performance maintains the same as a single site setup (non-distributed storage system) and do not vary with inter-site distance. This means that data is read from local storage disks, independently of the other site's system. Therefore, there is no inter-site communication (no data traffic on the network storage node's TCP ports - Gigabit Ethernet interfaces) during this phase of the testing as shown on the I/O performance monitoring output extracted from the YottaYotta netstorager:

Node front-end operations	Node front-end read	Node front-end write	Node TCP port received	Node TCP port sent	Node back-end operations	Node back-end read	Node back-end write
6704	976 MB	0 MB	0 MB	0 MB	31768	977 MB	0 MB

Analyzing this same table it is possible to confirm that the data is being read from locally attached disks because the amount of data from node's front-end (between workload server and node) reading and back-end (between node and local disks) reading are approximately the same. This means that all read data was supplied by local disks avoiding the necessity of inter-site communication.



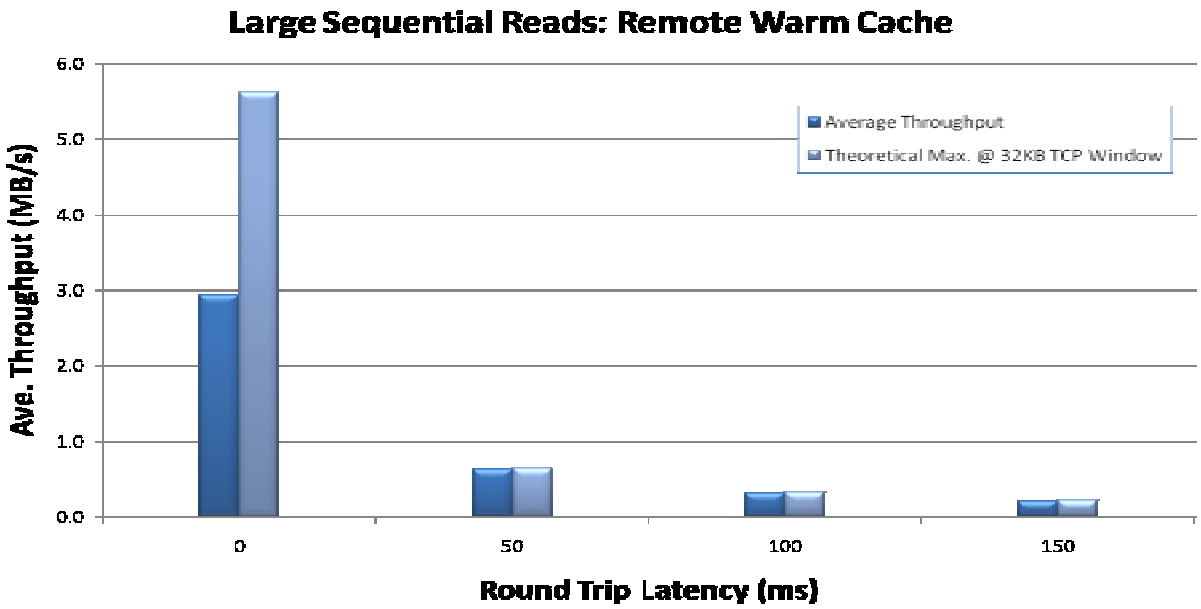
The above graph shows the performance results for local warm cache. They still stay as same as single site setup throughput performance and do not vary with inter-site distance. However, the performance is improved if compared with cold cache result since the process of reading data directly from cache is faster than reading from disks.

The table below shows the I/O performance monitoring output extracted from the YottaYotta node for local warm cache. In this case, there is no back-end reading because the whole data was supplied by network storager's cache.

Node front-end operations	Node front-end read	Node front-end write	Node TCP port received	Node TCP port sent	Node back-end operations	Node back-end read	Node back-end write
6000	976 MB	0 MB	0 MB	0 MB	0	0 MB	0 MB

As explained before, the reading process occurs in the following way: if the workload server of a site is reading data through the network storage nodes, the nodes of the site where the reading operation is being performed will, first of all, check for this data on their own cache, and then if this data is not cached there, they will do the same on the remote cache (network stogeragers' cache of the other site). Finally, if the data is not there either, it will be read from the locally attached storage.

Therefore, if the procedure described in the Methodology chapter is followed, it is possible to collect the remote warm cache results. These results are shown below:



The results show that the performance is negatively affected during remote warm cache and with round trip latency increase because read data is obtained over the WAN (T3 link) from remote network storage nodes as shown in table below:

Node front-end operations	Node front-end read	Node front-end write	Node TCP port received	Node TCP port sent	Node back-end operations	Node back-end read	Node back-end write
787	97 MB	0 MB	103 MB	2 MB	0	0 MB	0 MB

Even though the performance is degraded in remote warm cache state, it is really important to obtain data from remote site's cache before getting it from local disks because data on local disks could be out-of-data if there is dirty data not flushed to disks on remote cache. Therefore, it is necessary to check remote cache to provided data consistency between sites.

In the remote warm cache graph, the theoretical maximum performance is calculated considering an average receiver's TCP window size of 32 KB during the read testing using the following formula:

$$Throughput \leq \frac{RWIN}{RTT}$$

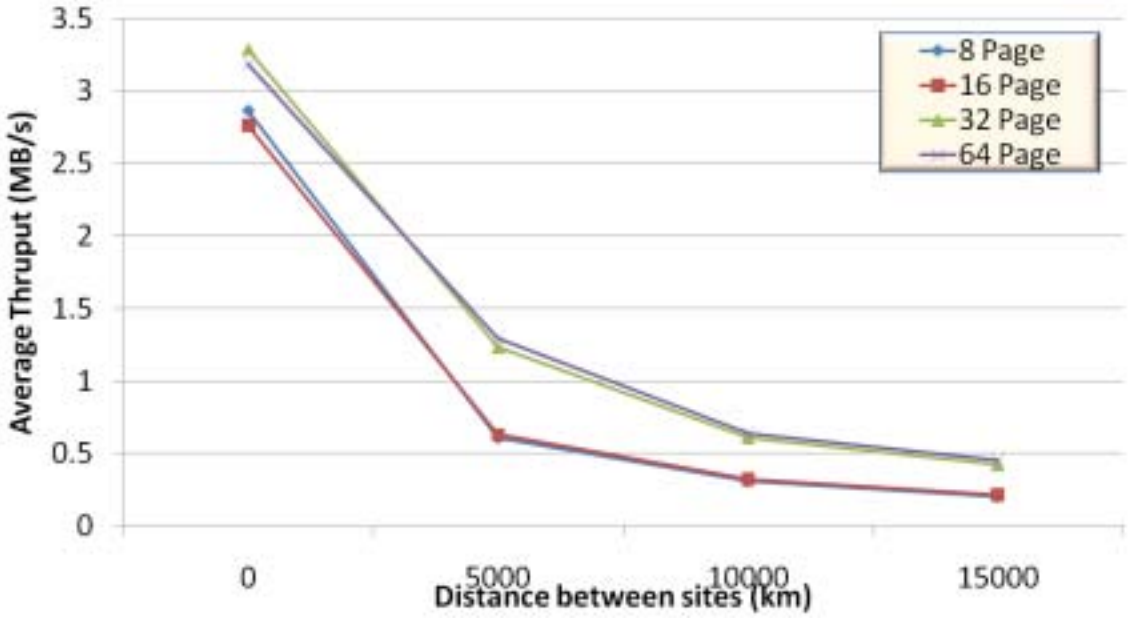
Where Throughput is the T3 link bandwidth (around 45 Mbps), RTT the round trip time latency and RWIN is the receiver's TCP window size. This formula is based in the fact

that TCP transmits data up to the window size before waiting the packets' acknowledge and because of that, full bandwidth of the network may not always get used.

For 0 ms of RTT latency, there is considerable discrepancy between theoretical maximum and measured performance. This is due an issue with the COM window and it will be covered in detailed during the large sequential read testing with fixed COM window size analysis.

In order to improve remote cache state reading performance, we implemented large sequential read testing using a range of pre-fetch settings (8, 16, 32 and 64 pages). Pre-fetch is the netstorager action of getting data well before ahead of the requested in a single operation. In this way, the netstorager will not need to always wait for the TCP to get its next operation's request.

The graph below represents the results for large sequential read testing using pre-fetch equals to 8, 16, 32 and 64 pages with RTT latency varying among 0, 50, 100 and 150 ms which correspond to the following distance between sites respectively 0, 5000, 10000, 15000 km:

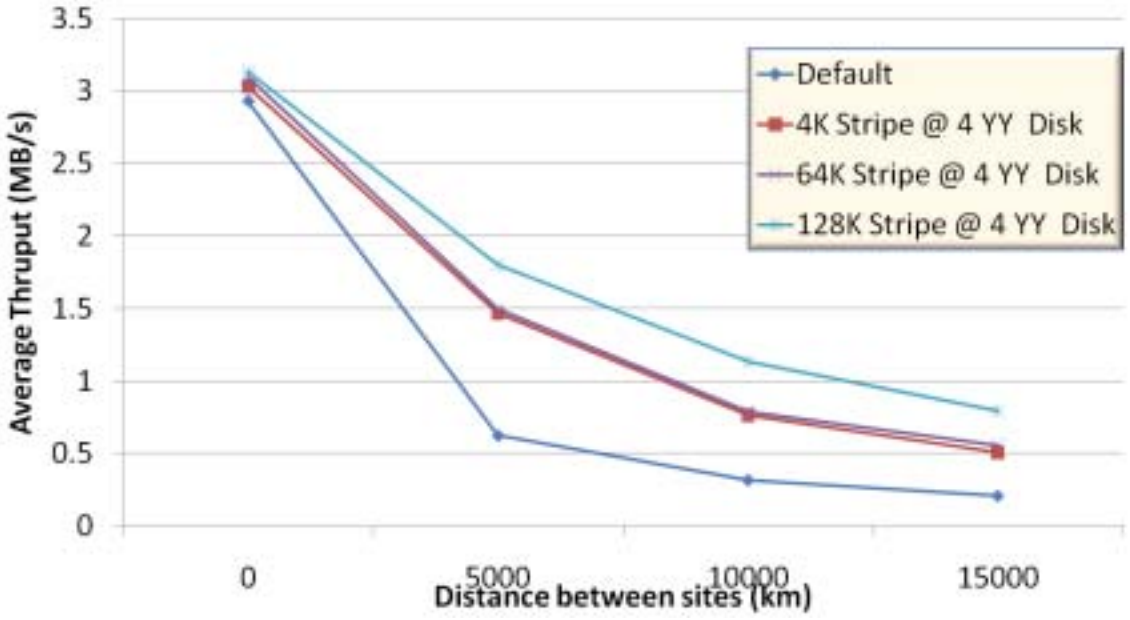


These results are for remote warm cache only because for cold cache and local warm cache the high performance presented previously while using default setting was really satisfactory and it is not a concern.

Increasing pre-fetch improves read performance over the WAN, but the improvement is non-linear. An 8-fold increase in pre-fetch only improves WAN performance by a factor of 2.2.

It was expected that an increase in pre-fetch would result in a comparable improvement in warm remote cache read performance. However, an eight-fold increase in pre-fetch improving remote read performance by little more than a factor of 2 wasn't expected. We decided to leave this issue unresolved momentarily as we explore other strategies to improve performance.

The next correspond to the large sequential read testing using Linux MD to stripe four exported YottaYotta volumes (single accessible storages created from the DR1) and form a single MD RAID 0 and provide improved performance.



These results are for remote warm cache only because, as mentioned before, for cold cache and local warm cache the high performance presented previously while using default setting was really satisfactory and it is not a concern.

The four YottaYotta volumes are striped by 4, 64 and 128 KB as shown above. The performance improvement is due to smaller stripes of the entire chunk allowing data to be read off the volumes in parallel, giving this type of arrangement higher bandwidth. However, RAID 0 does not implement error checking so any error is unrecoverable. More volumes in the array means higher bandwidth, but greater risk of data loss.

Striping the volumes by 128 KB seems to improve the performance more efficiently because the 128 KB stripes flow more suitable over TCP avoiding fragmentation for most transmitted segments.

To improve WAN traffic throughput and consequently improve reading throughput performance in remote warm cache state, the number of TCP connections per node between each node was set to one instead of two (default value).

At first, it might seem that increasing the number of TCP connections should theoretically improve performance. However, most of the TCP connections are not really transferring "reading data" during the testing; they are mostly transmitting Heartbeat information between nodes. For example, for two TCP connection per node between each node (default setting), six connections out of eight will be used to transfer Heartbeat information between nodes and just two connections will be used to transfer “reading data” as shown below:

Node1Site1	Node2Site1	Node1Site2	Node2Site2
192.168.20.2	192.168.20.3	192.168.30.2	192.168.30.3

Address A	Port A	Address B	Port B	Packets	Bytes	Packets A->B	Bytes A->B	Packets A<-B	Bytes A<-B
192.168.20.2	11000	192.168.30.3	1029	1139	92094	521	43242	618	48852
192.168.30.3	11000	192.168.20.2	1026	1139	92094	618	48852	521	43242
192.168.30.3	11000	192.168.20.3	1028	2015	164150	1001	81642	1014	82508
192.168.20.3	11000	192.168.30.3	1028	2015	164150	1013	82462	1002	81688
192.168.20.3	11000	192.168.30.2	1027	2287	186434	1158	94044	1129	92390
192.168.30.2	11000	192.168.20.3	1029	2292	186928	1131	92534	1161	94394
192.168.20.2	11000	192.168.30.2	1028	249734	72623877	126760	62785061	122974	9838816
192.168.30.2	11000	192.168.20.2	1027	254275	72772011	124281	9940510	129994	62831501

The information above is collect from the WAN traffic capture using tcpdump during testing under default setting and were processed using Wireshark.

The highlighted node and the highlighted connections are the ones that are transferring “reading data”. The other connections are just transferring Heartbeat information between nodes.

The performance will for sure improve if we can increase just the number of TCP connections which are transferring "reading data" and not the number of connection per node between each node on all blades because TCP adapts to share network bandwidth with other connections even if these connections are not transferring large amounts of data.

However, if we use only one TCP connection per node between each node as shown below:

Address A	Port A	Address B	Port B	Packets	Bytes	Packets A->B	Bytes A->B	Packets A<-B	Bytes A<-B
192.168.30.3	11000	192.168.20.2	1025	953	76494	475	37686	478	38808
192.168.20.4	11000	192.168.30.3	1026	2145	175278	1034	85140	1111	90138
192.168.20.3	11000	192.168.30.2	1026	2266	186756	1146	93784	1120	92972
192.168.20.2	11000	192.168.30.2	1027	335933	133757414	137519	117408974	198414	16348440

We will obtain better WAN traffic throughput since there will be less connections transferring “non-reading data” even though just one connection will be transferring “reading data”.

In addition, the TCP buffer size on the Gigabit Ethernet interfaces was adjusted to a tuned value, which depends on the RRT latency in use and is calculated using the following formula:

$$Throughput \leq \frac{\text{Buffer Size}}{RTT}$$

Where Throughput is the T3 link bandwidth (around 45 Mbps) and RTT is round trip time latency.

To get maximal throughput it is critical to use optimal TCP buffer sizes for the link in using, in our case, T3 link. If the buffers are too small, the TCP congestion window will never fully open up. If the receiver buffers are too large, TCP flow control breaks and sender can overrun the receiver, which will cause the TCP window to shut down.

Below are the front-end throughput testing results for 1 TCP connection per node between each node and optimal buffer size (this value depends on the latency in use). In addition, for comparison reasons, the same results for default settings:

RTT	**0 ms**		**Buffer size**	**Prefetch**	**TCP connections**
Iteration	**Cache state**		**1M**	**64**	**2**
	cold	**remote warm**			
1	72.6	2900			
2	72	3400			
3	73.1	3500			
Average	72.5666667 MB/s	3266.666667 KB/s			

Iteration	**Cache state**		**128k**	**64**	**1**
	cold	**remote warm**			
1	71.9	3200			
2	72.6	3200			

3	71.2	3400
Average	71.9 MB/s	3266.666667 KB/s

RTT 50 ms **Buffer size** **Prefetch** **TCP connections**

Iteration	**Cache state**	
	cold	**remote warm**
1	73.2	1300
2	73.6	1300
3	74.6	1300
Average	73.8 MB/s	1300 KB/s

1M **64** **2**

Iteration	**Cache state**	
	cold	**remote warm**
1	70.6	2100
2	70.9	2300
3	71	2100
Average	70.83333 MB/s	2166.666667 KB/s

256k **64** **1**

RTT 100 ms **Buffer size** **Prefetch** **TCP connections**

Iteration	**Cache state**	
	cold	**remote warm**
1	75.8	683
2	72	643
3	73.4	644
Average	73.73333 MB/s	656.6666667 KB/s

1M **64** **2**

Iteration	**Cache state**

512k **64** **1**

	cold	remote warm
1	71.1	1400
2	69.7	1400
3	69.1	1400
Average	69.96667 MB/s	1400 KB/s

RTT	**150 ms**	**Buffer size**	**Prefetch**	**TCP connections**

Iteration	**Cache state**	
	cold	**remote warm**
1	71.6	442
2	71.5	478
3	74.1	478
Average	72.4 MB/s	466 KB/s

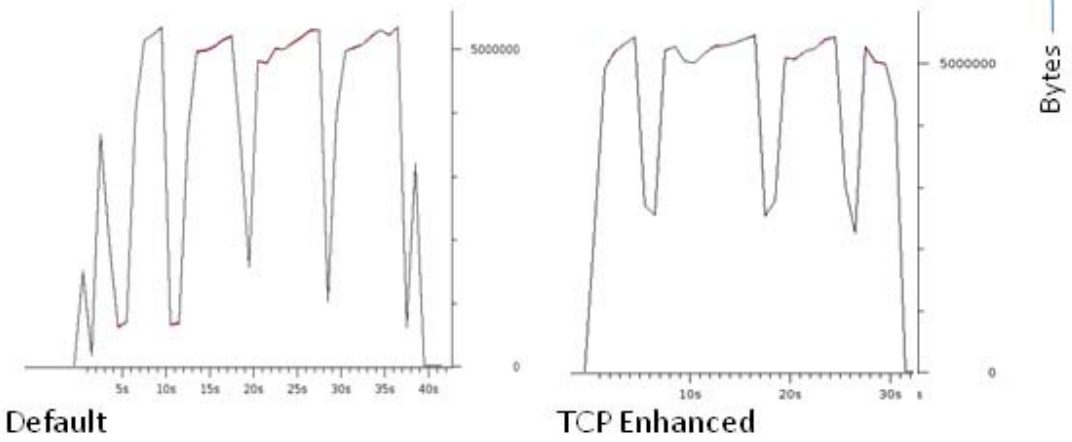
1M **64** **2**

Iteration	**Cache state**	
	cold	**remote warm**
1	68.5	1000
2	70	883
3	70	914
Average	69.5 MB/s	932.3333333 KB/s

768k **64** **1**

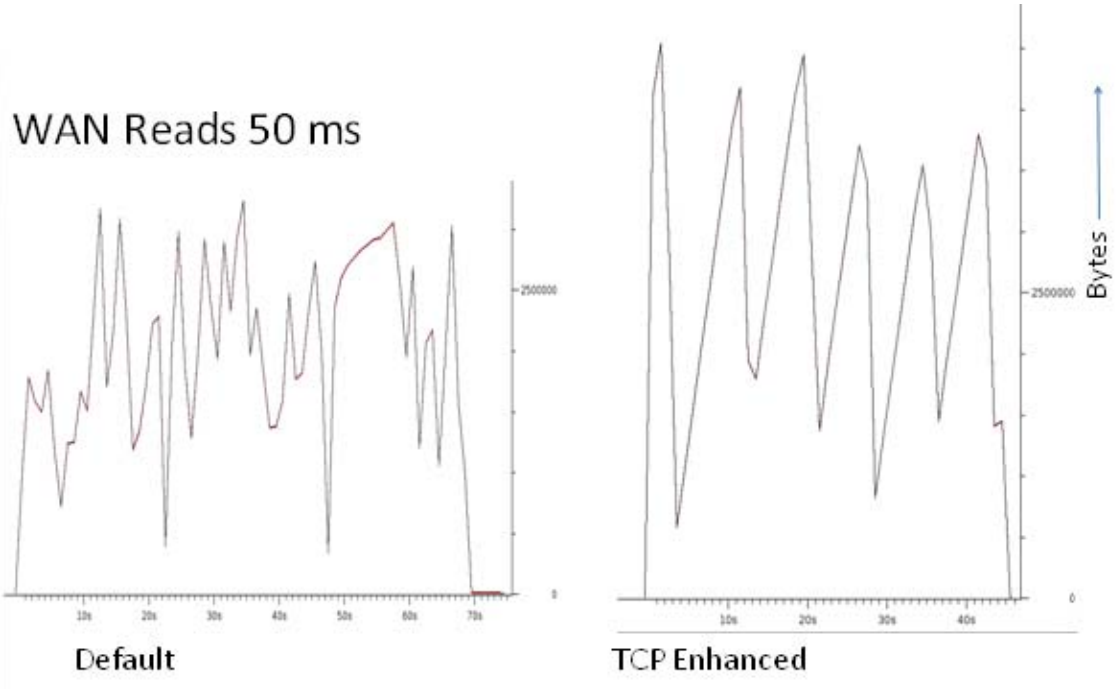
It is possible to notice a clear front-end performance improvement while using enhanced TCP settings (one TCP connection and optimal buffer size). This is due to the WAN traffic throughput performance improvement as we can see by analyzing the WAN throughput over time captured using tcpdump (where the y axis is in Bytes and x is in seconds):

WAN Reads 0 ms

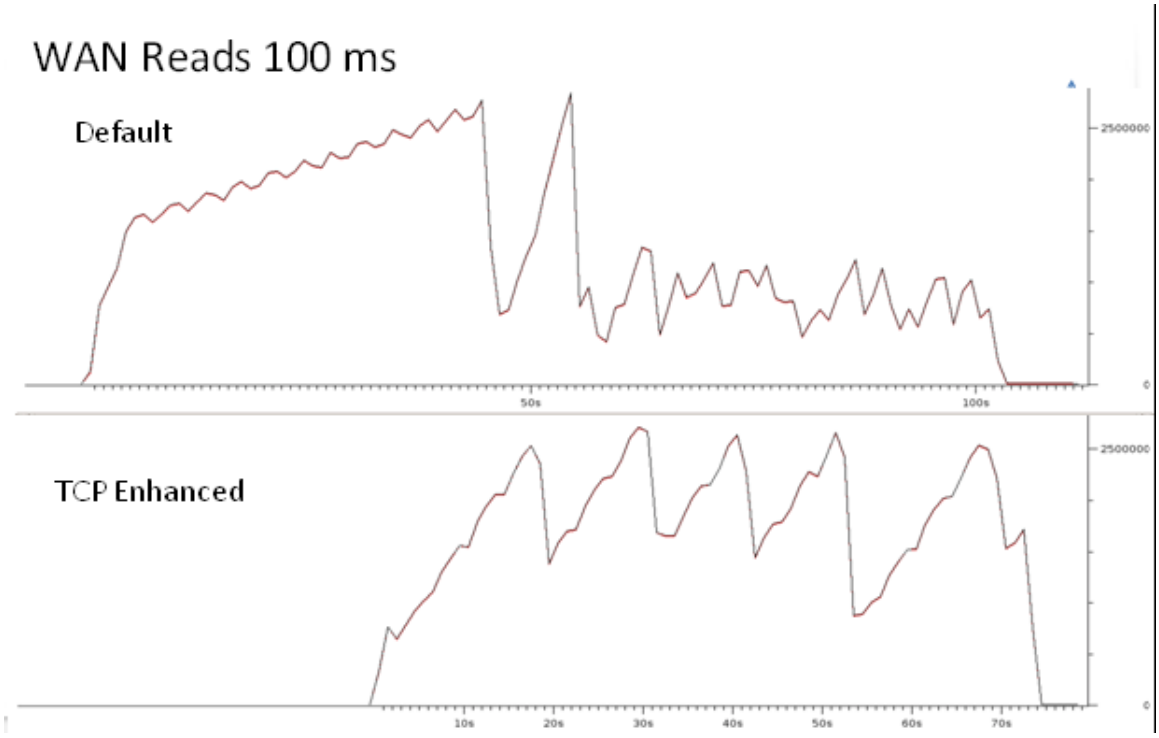


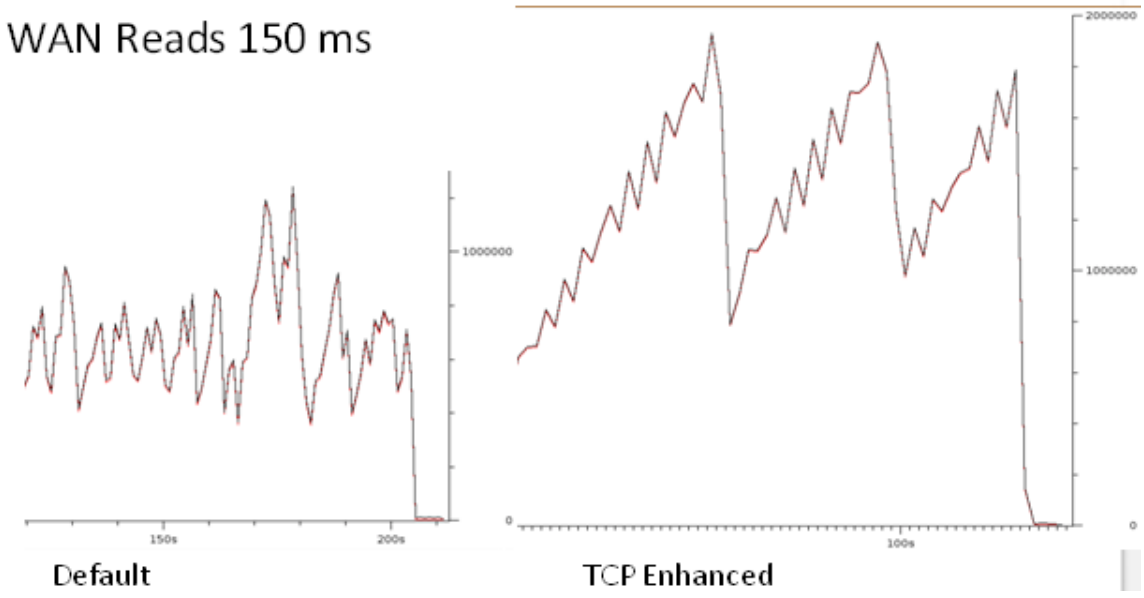
Default TCP Enhanced

WAN Reads 50 ms



Default TCP Enhanced





Even though the performance is improved while using enhanced TCP setting, the T3 link bandwidth (around 45 Mbps) is not being fully utilized.

If we analyze the tcpdump captures, we can notice that the receiver's window size never goes higher than 65k even if we use enhanced TCP settings, unlimited bandwidth and no latency (0 ms).

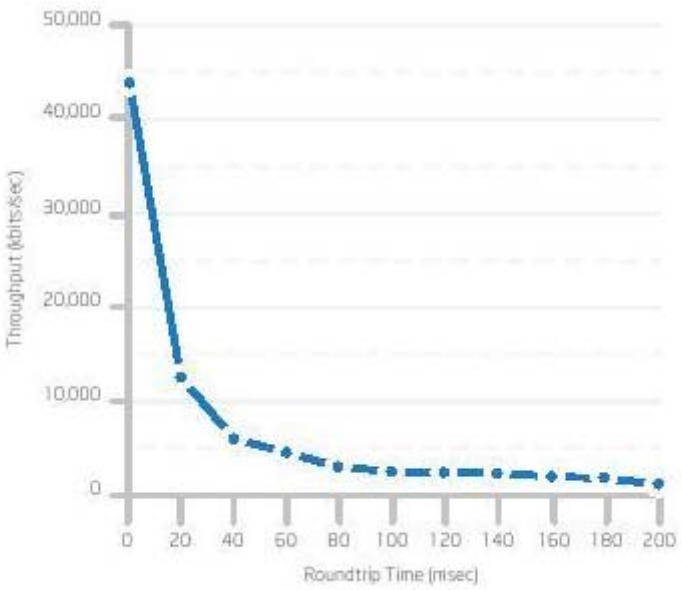
This might be happening because the YottaYotta firmware does not have RFC 1323 turned on. In traditional TCP, the window size cannot be larger than 65,535 bytes (because the unsigned integer field that holds it on the TCP header is only 16 bits wide). Normally, applications for data replication need a larger window than 64KB. This is achieved by turning on TCP extensions specified in RFC1323, `TCP Extensions for High Performance', published in 1992 and now supported by most operating systems. The TCP window scaling option allows one to use TCP window sizes of up to 1,073,741,823 bytes.

However, RFC 1323 is turned on in the YottaYotta firmware. The issue is that if Ethereal or Wireshark do not witness the connection negotiation, they will not display the right receiver's window size, because it is during the connection negotiation that the hosts advertise whether RFC1323 is turned on or off and if Ethereal doesn't witness that, it will calculate the receiver's window size assuming that RFC1323 is off.

After capturing the WAN traffic making sure that the connection negotiation was included, the receiver's window size showed on Ethereal started to make sense. It normally starts with the same value as the buffer size and it doesn't seem to change much over time (it is always around the buffer size value). Therefore, the receiver's window size doesn't seem to be limiting the WAN bandwidth utilization.

However, the algorithms that prevent a sending TCP peer from overwhelming the network known as slow start and congestion avoidance increase the sender's window (the number of segments that the sender can send) when initially sending data on the connection and when recovering from a lost segment seem to be limiting the WAN bandwidth utilization.

These algorithms work well for small bandwidth links with low latency and smaller receiver's window sizes. However, when you have a TCP connection with a large receive window size and a large BDP (bandwidth-delay product), such as replicating data between two servers located across a high-speed WAN link (e.g. T3) with a 100ms round-trip time, these algorithms do not increase the sender's window size fast enough to fully utilize the bandwidth of the connection as we can see in the picture below from an Intel's white paper "Optimizing WAN Performance for the Global Enterprise":



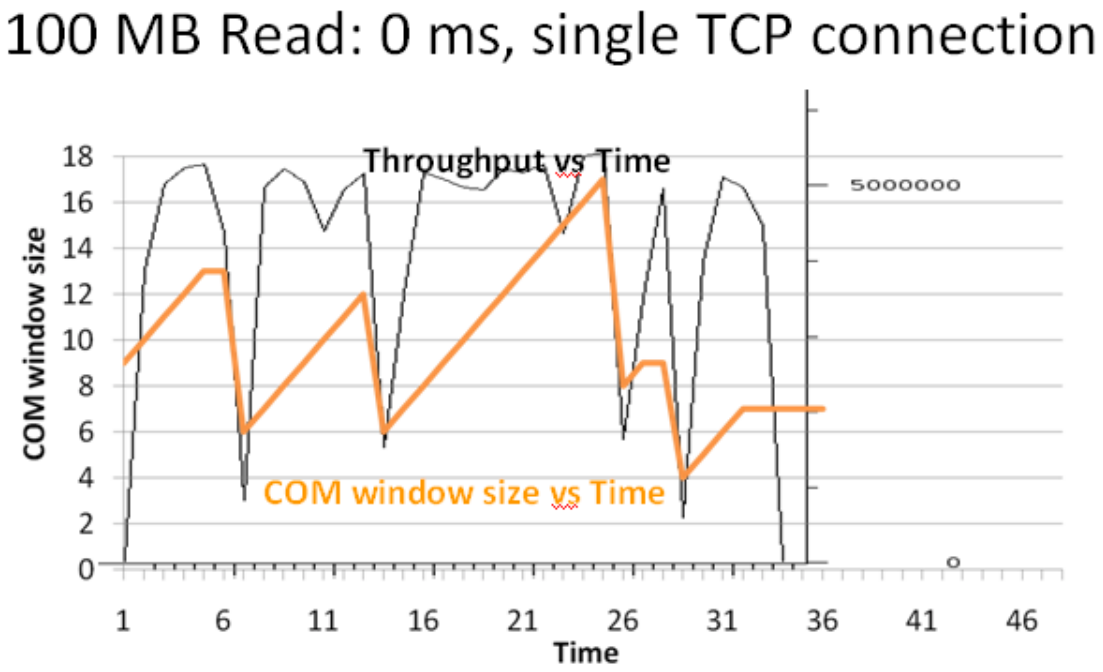
Impact of latency on T3 throughput

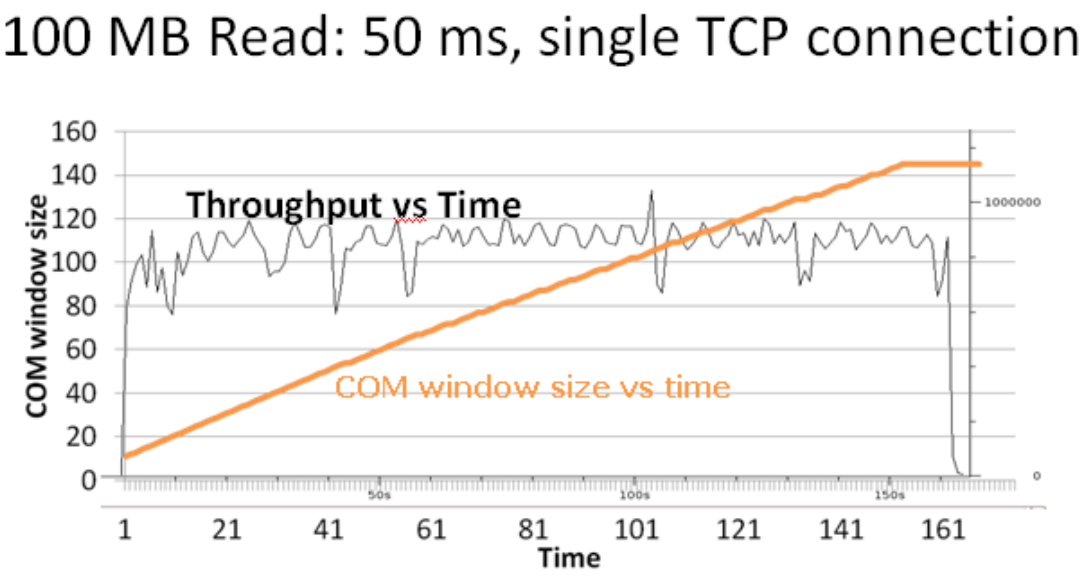
To better utilize the bandwidth of TCP connections in these situations, the Next Generation TCP/IP stack includes Compound TCP (CTCP). CTCP more aggressively increases the sender's window size for connections with large receiver's window sizes and BDPs. CTCP attempts to maximize throughput on these types of connections by monitoring delay variations and losses. In addition, CTCP ensures that its behavior does not negatively impact other TCP connections.

As mentioned before, even though the WAN throughput performance is positively influenced while using enhanced TCP setting, we can observe that the TCP window is still periodically collapsing (impact becomes increasingly severe at greater latencies) and T3 link bandwidth is not being well utilized. To maximize WAN bandwidth utilization, we optimized the COM bandwidth management settings.

COM bandwidth management is a YottaYotta dynamic tuning mechanism that supports throttling the number of outstanding messages for busy links and expanding the number of messages sent when response times for message acknowledgements are met. Provided messages are acknowledged within an acceptable time limit, the number of messages sent can be increased until link saturation is achieved (similarly to the TCP flow control mechanism).

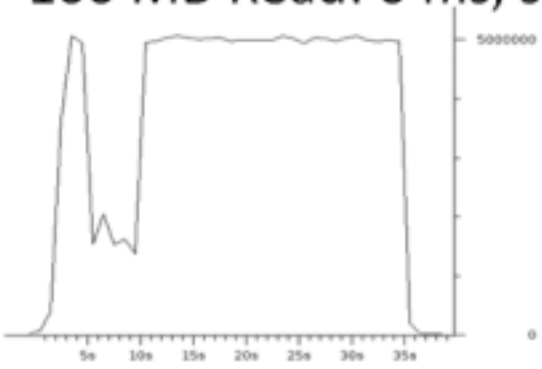
Below are shown the COM window size over the time graph overlapped by WAN traffic throughput over time for default settings:



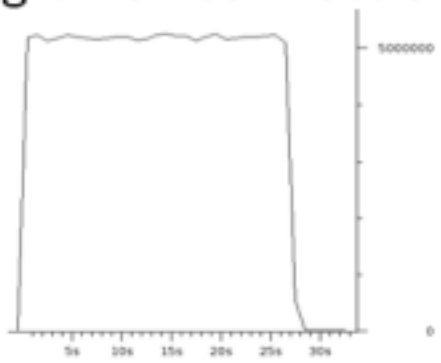


Fixing a static **optimal** COM window size (62 messages for 50 ms of latency and 8 messages for 0 ms) can limit the amount of data transmitted by the sender providing a stable WAN throughput and avoiding the TCP window to collapse. In addition, it will avoid the sender to be increasing its sending rate until it exceeds the capacity of the network and packet loss occurs as shown by the graphs (WAN throughput over time) below:

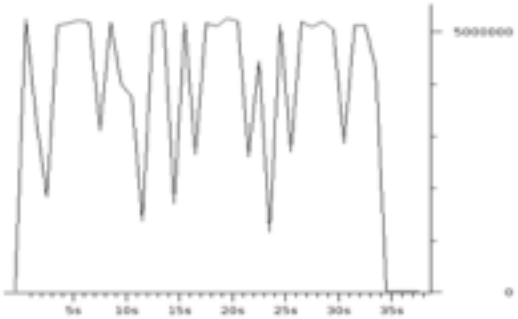
100 MB Read: 0 ms, single TCP connection



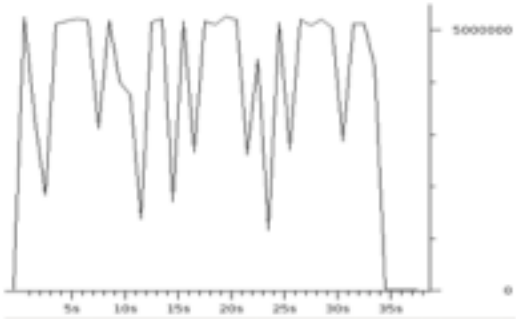
COM: 6 messages



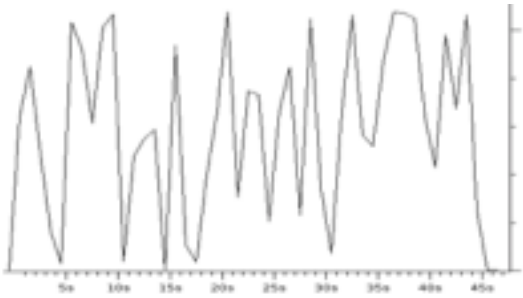
COM: 8 messages



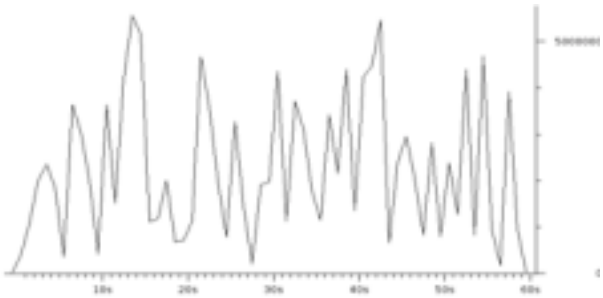
COM: 10 messages



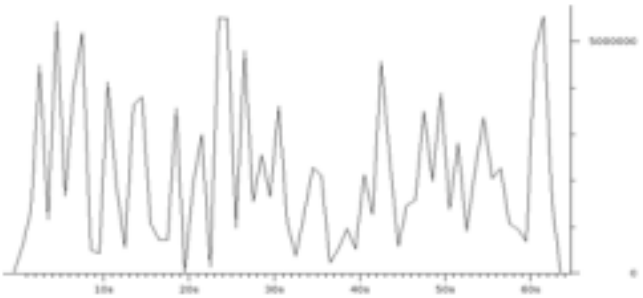
COM: 12 messages



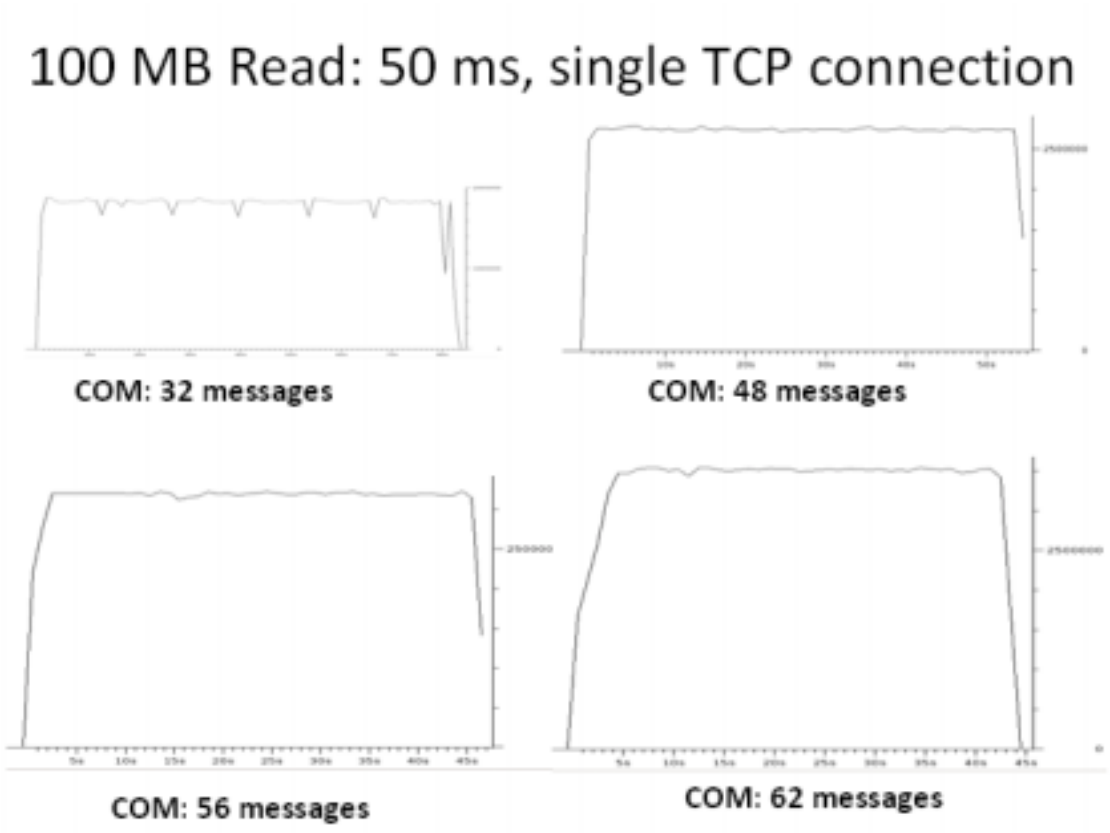
COM: 14 messages

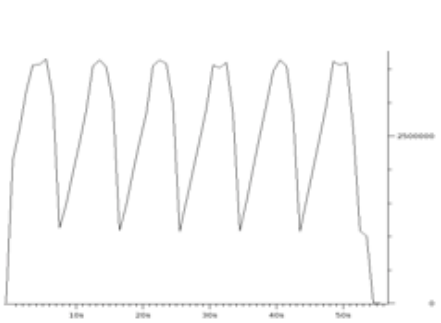


COM: 60 messages

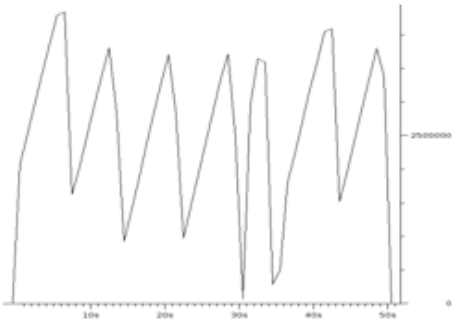


COM: 1000 messages

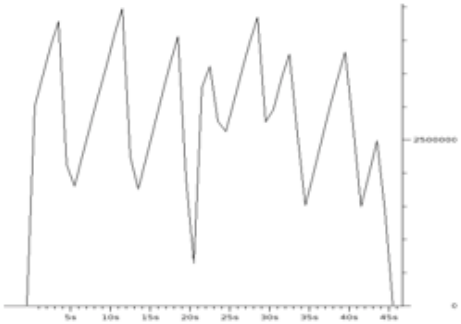

100 MB Read: 50 ms, single TCP connection
COM: 32 messages
COM: 48 messages
COM: 56 messages
COM: 62 messages



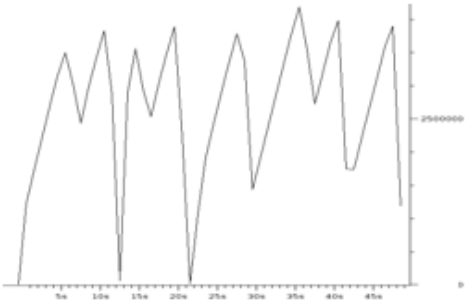
COM: 64 messages



COM: 110 messages

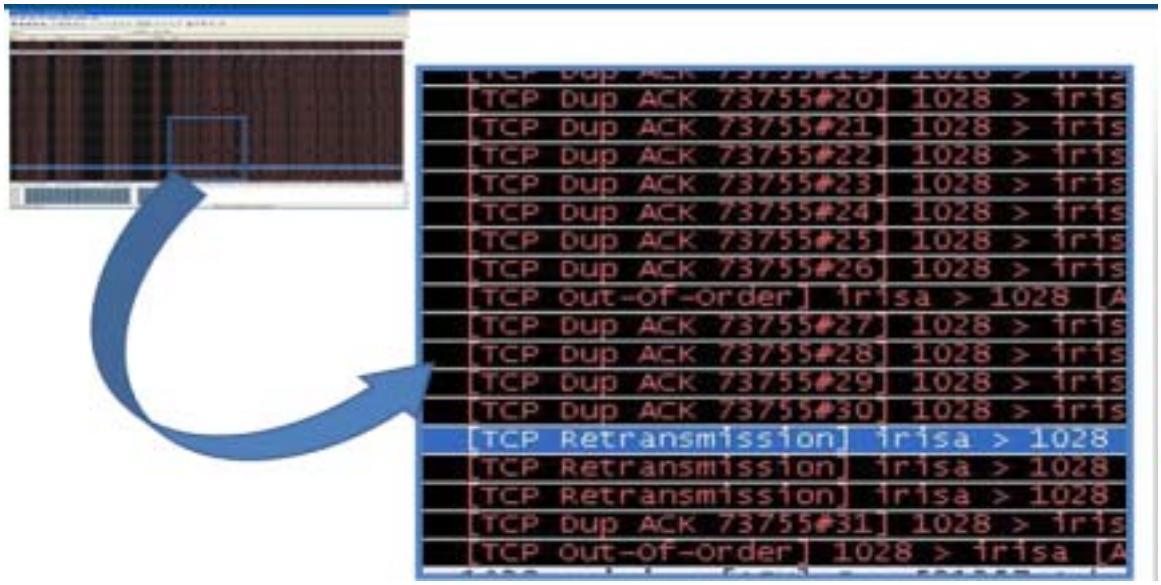


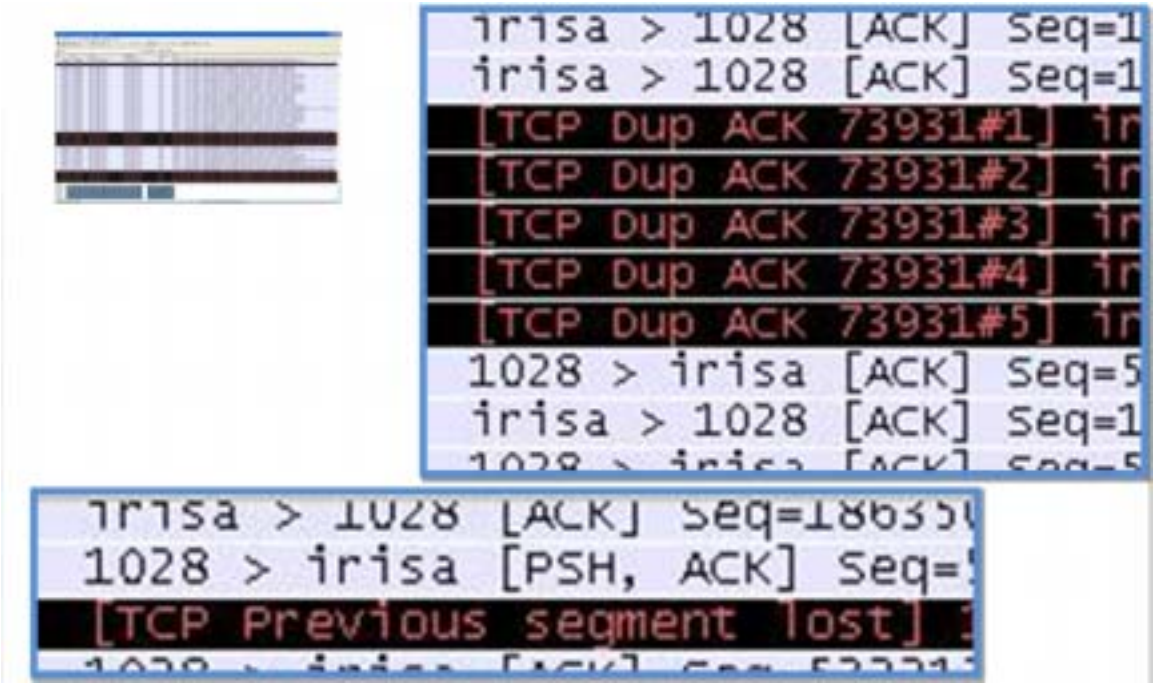
COM: 120 messages



COM: 130 messages

If we analyze the tcpdump captures for **optimal** fixed COM window size (62 messages for 50 ms of latency and 8 messages for 0 ms), we can notice that we still have indication of TCP window being collapsing because of the presence of TCP duplicate acknowledge and TCP retransmission packets as shown below:





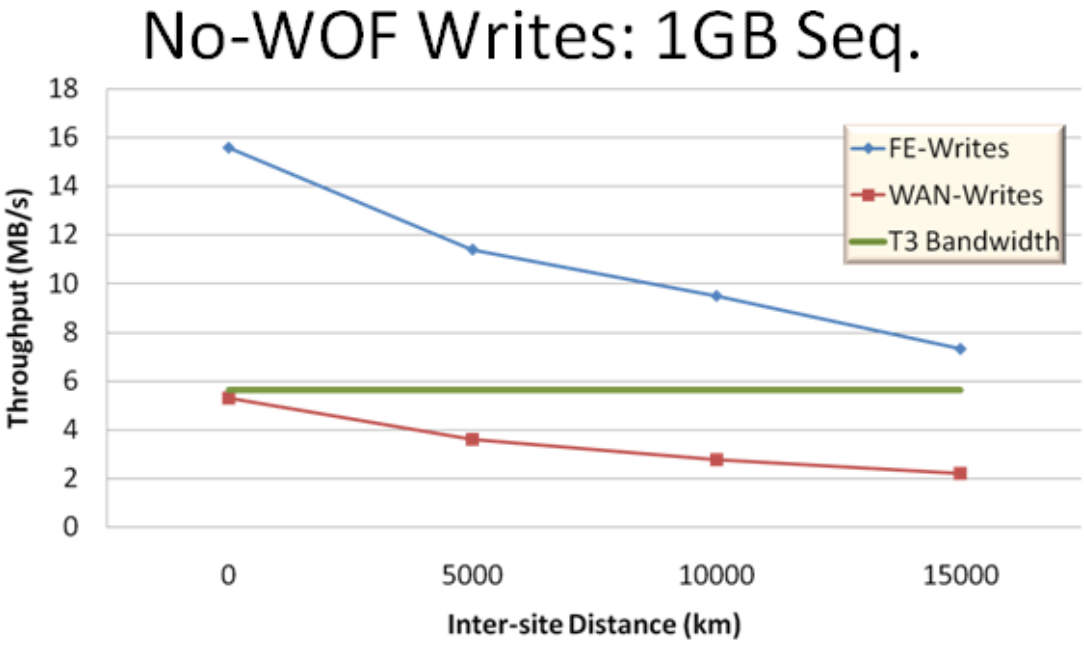
However, those packets are either: indicators of the receiver's window size being opened after previously being shrunk. This is a normal TCP behavior, resending a just-sent ACK with a different window size to indicate that the receiver's window size has changed (this will explain TCP Dup ACK). Or the sender is transmitting a packet and keeping a timer from when the packet was sent. Even if the receiver gets this packet and acknowledges it, this acknowledged packet sent by the receiver might take a while to get back to the sender, depending on link congestion, latency and other factors. Therefore, if this acknowledged packet comes after the sender's timer expires, the sender will retransmit (TCP Retransmission) this packet and the receiver will re-acknowledge it (TCP Dup ACK).

So, it make sense to see a bunch of TCP Dup ACK and TCP Retransmission together for the second case mentioned above since this period will represent the peaks on the throughput over time graphs where the WAN link is more congested and the acknowledge packet sent by the receiver might take longer to get back to the sender forcing the sender to decrease the amount of data sent per second and causing the dips on the throughput over time graphs.

In addition, if you take a look on the tcpdump captures for the cases where the throughput over time graphs are almost a square (stable WAN throughput), you will notice that there isn't TCP Retransmission and TCP Dup ACK packets because we are limiting the amount of data sent to the COMTCP port through COM window size avoiding the link to get congested.

4.2- Large Sequential Write Testing

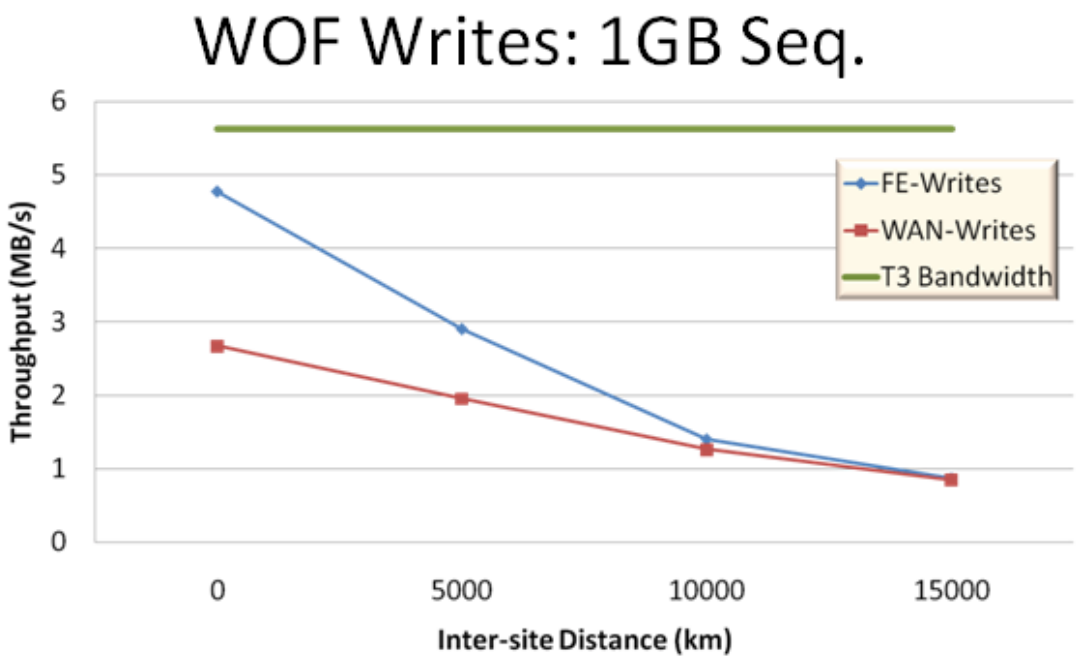
The first large sequential write testing were performed using the default configuration (no-WOF) on the YottaYotta network storage nodes and following the procedure described previously on the Methodology chapter.



In the above graph, the RTT latency is varying among 0, 50, 100 and 150 ms which correspond to the following distance between sites respectively 0, 5000, 10000 and 15000 km. The green line represents to bandwidth of the T3 link used (around 45 Mbps which is equivalent to 5.625 MB/s) and FE-Writes means front-end writes.

As mentioned before, in opposite to the reading operation performance, the writhing performance does not depend on the network storage nodes' cache state. The process of writing on asynchronous cache mode (default mode) start with the workload server writing data to the network storage nodes' cache of a site. Then, acknowledgements are returned immediately upon data receipt by the GSX 3000s of this site and the cached data is written to the disks (local and remote storage systems) later. The "dirty" data is kept in cache and periodically (each 60 seconds by default) flushed to the disks (local and remote storage systems.

The throughput performance for writing operations will depend on whether the Write Order Fidelity (WOF) functionality is in use or not because of the way WOF algorithm (this algorithm was explained earlier in the Methodology chapter) works to ensure that the order of writes to a particular storage volume are preserved all the way to the physical medium.



The above results are for the same testing but using WOF. It is clear that performance is degraded while using WOF because data is transmitted over the WAN to the other site at the same that front-end operations still are being performed. This happens to maintain same order of writes on both sites.

During no-WOF write operations, data is stored in cache during front-end operations and sent over WAN to the other site after front-end operations are done. This is the reasons because no-WOF writing present better performance.

To improve performance while using WOF, the same strategy of fixing the COM window size used during the read testing was implemented and the same kind of results and improvement obtained during the read testing were obtained during the write testing with WOF. Those results are shown in the APPENDIX A.

5- Conclusion

For the large sequential read testing on a DR1 (Distributed RAID1), testing achieves local (as a single site setup, i.e. non-distributed storage system) read performance for most data access:

- Cold cache block-level reads around 45 MB/s.
- Warm cache block-level reads approximately 77MB/s

However, read performance across the WAN is sub-optimal. At 50 ms of inter-site RTT latency and the default 8-page pre-fetch setting, YottaYotta network storager achieves only about 10% utilization of the available T3 bandwidth and even with no distance (RTT equal 0ms), netstorage achieves only approximately 50% of available bandwidth.

Our analysis suggests that this issue may be related to buffer management within TCP setting on netstorager's Gigabit Ethernet interfaces which are responsible for inter-site communication and COM bandwidth management for both reading operations.

We partially solve this performance issues by adjusting some non-exposed TCP COM settings (COM window size) and adjusting the number of TCP connections and buffer size. We partially solved this issue because part of it is related to the algorithms that prevent a sending TCP peer, **using standard TCP stack**, from overwhelming the network known as slow start and congestion avoidance.

Increasing pre-fetch improves WAN read performance, but the improvement is non-linear. An 8-fold increase in pre-fetch only improves WAN performance by a factor of 2.2. In addition, volume striping also improves performance due to the fact that smaller stripes of the entire chunk allows data to be read off the volumes in parallel, giving this type of arrangement higher bandwidth.

For large sequential write testing, front-end write performance is significantly better than WAN write throughput. Nonetheless, front-end write performance is still limited by WAN write performance.

WOF front-end write performance is below T3 bandwidth even at 0 ms of RTT latency. No-WOF WAN-write throughput at 50 ms latency is only around 60% of full T3 bandwidth and WOF WAN-write throughput at 50 ms latency is only approximately 35% of full T3 bandwidth.

As for the read testing, WAN write performance for both with and without WOF are also sub-optimal and this issue is solved in the same way as for the read testing by adjusting some non-exposed COM window size, the number of TCP connections and buffer size.

6- Bibliography

1. YottaYotta Inc., “Configuration & Operation Guide”, GSX 3000 Series, Release 3.10 Rev. F, 2007.

2. T. Verral, "Optimizing WAN Performance for the Global Enterprise", Intel Corporation, 2006.

3. G. Oikawa, G. Czarniewski, A. Feldman, "USPTO Performance Investigation", YottaYotta Inc., 2007.

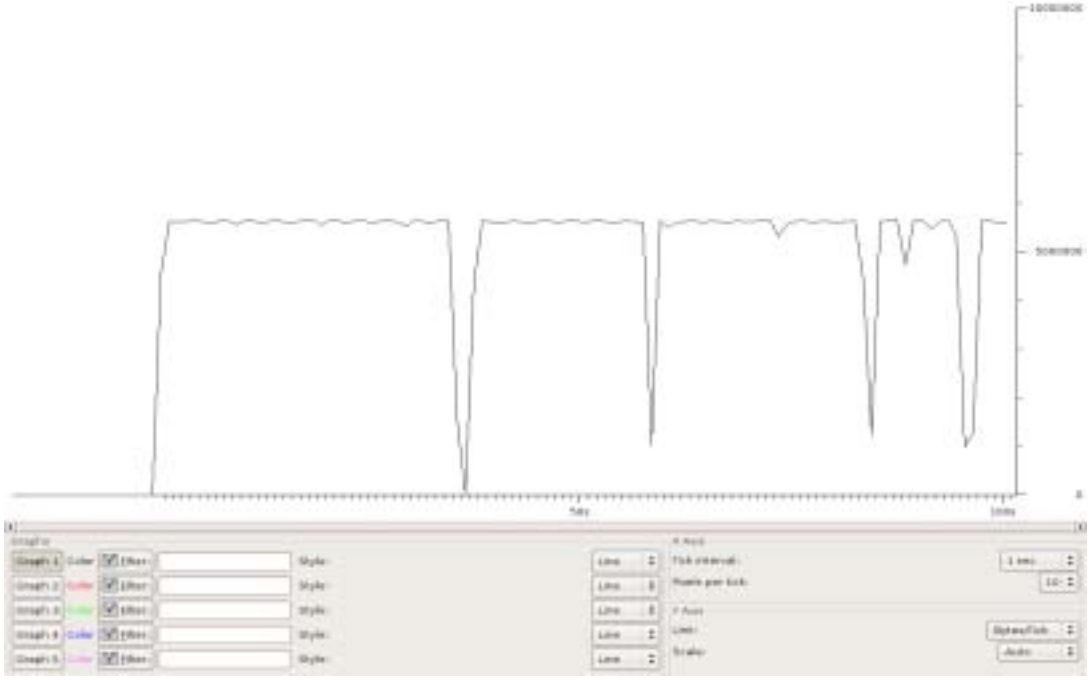
4. YottaYotta Inc., “Field Note COM Bandwidth Tuning”, GSX 3000 NetStorager System, Release 3.1rc50, 2008.

5. YottaYotta Inc., “Field Note COMTCP Tuning”, GSX 3000 Series, Release 3.10 Rev. F, 2007.

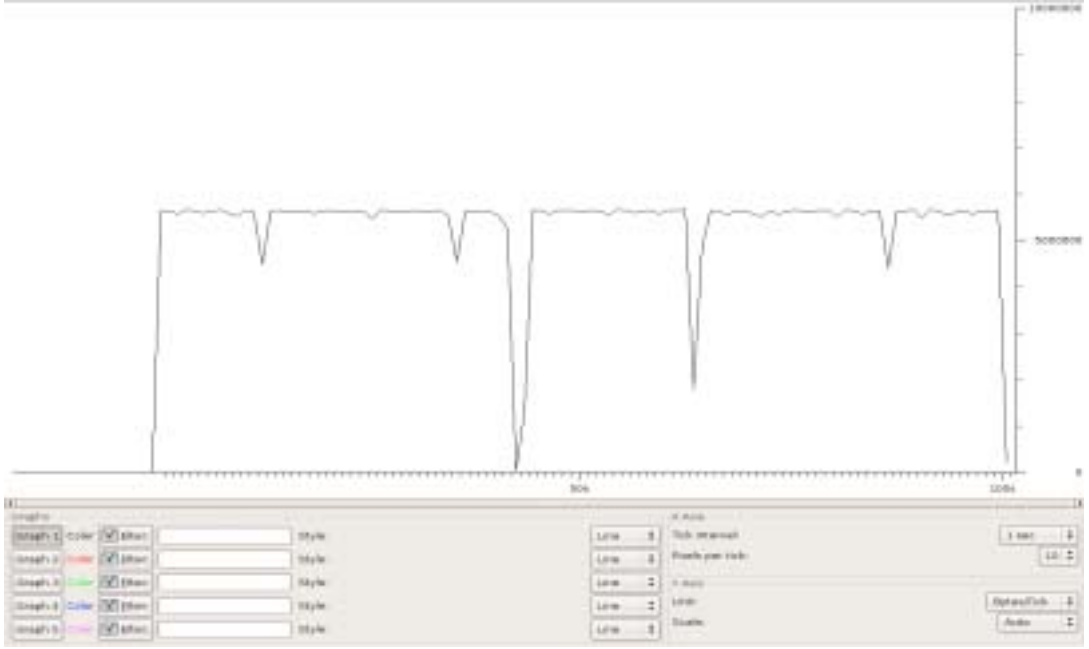
6. J. Matthews, “Computer Networking: Internet Protocols in Action”, Wiley, 2005.

APPENDIX A: Large sequential writing results with WOF and fixed COM window size

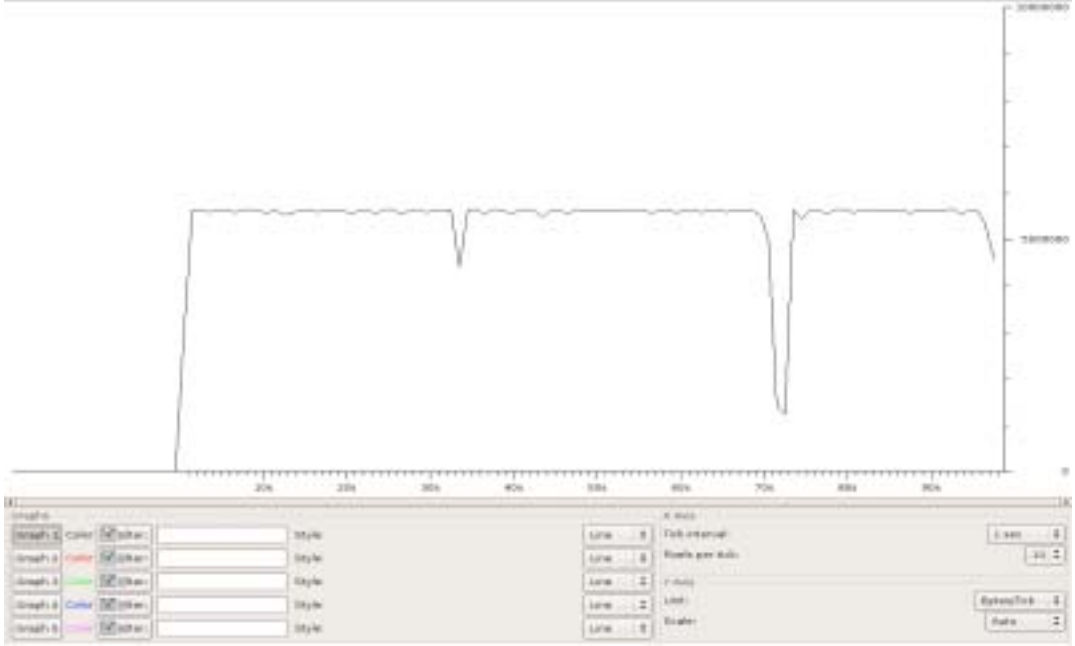
- Fixed COM window size equal to 12 messages and no latency (0 ms):



- Fixed COM window size equal to 16 messages and no latency (0 ms):



- Fixed COM window size equal to 17 messages and no latency (0 ms):



- Fixed COM window size equal to 22 messages and no latency (0 ms):