TCP/UDP BENCHMARK

January 18, 2015

Greg Ward - 5273669
IAD – Assignment 1

REVISION HISTORY

Greg Ward	January 11, 2016	- Created document so that I may record resources here
Greg Ward	January 14, 2016	- Began writing report. Left spaces for where results are important to make speculation
Greg Ward	January 17, 2016	- Updated test results. Changed program to reflect better results. Updated test results
Greg Ward	January 18, 2016	- Completed report. Submitted to drop box

Contents

Abstract

The purpose of this assignment was to create an application/series of applications to benchmark the speed of sending varying quantities of data (blocks of 1,000, 2,000, 5,000 or 10,000 bytes at a time) using either TCP or UDP over IPv4 and Ethernet in the SET labs of Conestoga College. These tests were conducted by sending from one machine in the same row to another machine on the same row (not locally as to actually test speed). Each machine was running the same OS at the time (Windows 7 was used as well as Ubuntu on a virtual machine hosted on the Windows 7 desktop).

The results of the test were under the theoretical speeds. With TCP, we should expect this due to the unseen factors of handshaking/resending missed packets. As for UDP, it was only when we sent large amounts of data that we saw the theoretical meet the actual. This was only the case in Windows as the UDP speed with the Linux VM was not anywhere near expected, but the reliability was much greater than with the Windows machine. Overall, the calculated speed of sending each block size increased as the block size increased, pointing that with higher quantities of data more speed can be devoted to this. No string functions or CPU intensive function calls were used while the sender was operating.

This assignment was completed for Ed Barsalou, professor of the Industrial Application and Design course.

Testing

Testing Method

Testing has been carried out using two separate machines that are next to each other, using the same switch and on the same network. Each machine was running the same OS at the time of each test and no other background applications were running, besides those pre-installed on the student computers at the time.

Using the prescribed metric of block sizes, one computer acted as the sender, sending all blocks indicated when run until either all blocks had been sent or a signal was sent to stop the program. The number of blocks sent would be displayed on the sender, the time taken to complete the send along with any other information (if a partial send was completed or blocks were not sent, this information was recorded).

The receiver would display this information, along with the calculation of estimated speed/time.

Testing Results

The following tables showcase the results of the test run from Windows to Windows and Linux to Linux using both TCP and UDP over IPv4. All tests were completed on a 1gbps network speed. The standard TCP window was left at 64kb and not changed.

In order to account for the true amount of data put over the network for our tests, investigation was done to take a look at what the composition of each frame is, what the amount of data actually is and add it all together. Bytes for the headers (Protocol and Ethernet) were added as well as for the Ethernet protocol. The size of each block was divided and the number of bytes for the fore mentioned headers is multiplied by the number of frames it would take to send each block.

Additional Data Considerations

For TCP/IP, the additional payload per frame is 78 bytes (40 bytes for the protocol header + 18 bytes for the Ethernet header and frame check sequence + 20 bytes for the frame preamble and interframe gap).

For UDP/IP, the additional payload per frame is 66 bytes (the same as above but the protocol header is 28 bytes (20 bytes for IPv4 and 8 for UDP).

Ethernet Frames per block

Assuming the MTU of the Ethernet network is only 1500 bytes, the number of frames are as follows:

TCP:

Block Size 1000:	1000 /1460 = 0.685	Rounded to 1 Frame
Block Size 2000:	2000 / 1460 = 1.370	Rounded to 2 Frames
Block Size 5000:	5000 / 1460 = 3.425	Rounded to 4 Frames
Block Size 10000:	10000 / 1460 = 6.849	Rounded to 7 Frames

UDP:

Block Size 1000:	1000 /1472 = 0.679	Rounded to 1 Frame
Block Size 2000:	2000 / 1472 = 1.359	Rounded to 2 Frames
Block Size 5000:	5000 / 1472 = 3.397	Rounded to 4 Frames
Block Size 10000:	10000 / 1472 = 6.793	Rounded to 7 Frames

Windows 7 to Windows 7 – UDP over IPv4 and Ethernet

Block Size	Blocks Sent	Total Time	Speed	Missing Data	Out of Order Data
1000	500,000	9000ms	59.22 MB/s	8621 blocks	2230 blocks
2000	500,000	12000ms	88.83 MB/s	10,912 blocks	1655 blocks
5000	100,000	5000ms	105.28 MB/S	12,743 blocks	588 blocks
10000	100,000	9000ms	116.24 MB/S	10,350 blocks	566 blocks

Observations

Watching the speeds increase as the block size increased was interesting. Understanding that UDP is not reliable (as proven by the missing data numbers) as it has none of the same handshaking/ resending that TCP does, the speeds are quite good. Doing a bit of research on this, UDP is better with LARGE amounts of data being transferred per frame rather than partial amounts. Seeing it approach the hypothetical maximum confirmed these findings. The higher volume was worked faster.

Ubuntu VM to Ubuntu VM (running on Windows 7, network mode: BRIDGE) – UDP over IPv4 and Ethernet

Block Size	Blocks Sent	Total Time	Speed	Missing Data	Out of Order Data
1000	500,000	22000ms	24.23 MB/s	58 blocks	6 blocks
2000	500,000	43000ms	24.79 MB/s	140 blocks	10 blocks
5000	100,000	16000ms	32.9 MB/s	3 blocks	1 block
10000	100,000	29000ms	36.08 MB/s	48 blocks	9 blocks

Observations

Overall, this test was the most surprising. Understanding that UDP is nowhere near as reliable as TCP as it lacks all the handshaking/requesting resent data, I expected missed blocks/ out of order data comparable to Windows. Yes, data was out of order/missing but it seemed to be much more reliable than the UDP sending in Windows. The time taken to complete each test was well was very surprising. These speeds do not come anywhere close to the theoretical maximum although they do continue the trend of increasing as block size increases.

Windows 7 to Windows 7 – TCP over IPv4 and Ethernet

Block Size	Blocks Sent	Total Time	Speed	Missing Data	Out of Order Data
1000	500,000	21000ms	25.67 MB/s	0	0
2000	500,000	35000ms	30.80 MB/s	0	0

5000	100,000	19000ms	27.96 MB/s	0	0
10000	100,000	29000ms	36.37 MB/s	0	0

Observations

This was a bit surprising but I suspected something like this might occur. This was tested in the lab of 2A214 on row two. Through research and performing a lookup on the computers used for testing, there's no anomalous readings or explanation as to why it is so low. If I had to speculate, things that could impact the speed in this case are the TCP Window size (which I believe is higher than the 64KB standard (my basis for this is based upon the calculation used in this example based on Brad Hedlund's article: *How to Calculate TCP Throughput for long distances*), and the sender waiting for ACK's before sending more through to the receiver, out of order packets (if we judge by how often UDP packets are dropped in our windows UDP test, then this could account for a large amount of our "slow-down".

Ubuntu VM to Ubuntu VM (running on Windows 7, network mode: BRIDGE) – TCP over IPv4 and Ethernet

Block Size	Blocks Sent	Total Time	Speed	Missing Data	Out of Order Data
1000	500,000	18000ms	29.94 MB/s	0	0
2000	500,000	28000ms	38.50 MB/s	0	0
5000	100,000	10000ms	53.12 MB/s	0	0
10000	100,000	16000ms	65.91 MB/s	0	0

Observations

I was surprised to see the TCP/IP speed of the Ubuntu VM be the most reliable based on speed. Considering the results of the UDP test, I was unsure if the speed would be comparable to the TCP test on windows. After researching this, there was only rumors through Stack Overflow articles of this being optimized in the Linux kernel (through the use of: *Why MongoDB is faster on Linux vs windows*). The results are still less than the theoretical but this does not take into account things such as handshaking, the resending of packets or the possible slowdown of the hardware.

Theoretical Speeds

Network	TCP	UDP
100 megabit	11.865 MB/s	11.963 MB/s
1 gigabit	118.660 MB/s	119.635 MB/s

Using Ethernet to transfer this data, the maximum amount of bytes we can transfer at one time is 1538 bytes. An 8 bit preamble is the head of the frame and a 12 bit gap pads the end, well, leaving 1518 bits for the actual Ethernet frame

The structure of this 1518 bytes is 1500 bytes for the maximum amount of data that can be transferred, 14 bytes for an Ethernet header and 4 bytes for a frame check sequence.

Breaking down the 1500 bytes, we know that this must consist of the appropriate protocol headers.

UDP: 1500 bytes MTU - 8 bytes for the UDP header - 20 bytes for the IPV4 headers = 1472 bytes of pure data

TCP: *for this, I am not taking into consideration any TCP options or resends that may occur

1500 bytes MTU - 8 bytes for the UDP header - 20 bytes for the IPV4 headers = 1460 bytes of pure data.

Converting our network speed into actual bytes

100 megabits = .0125 GBs (gigabytes) (or 12,500,000 bytes)

1 gigabit = 0.125 GBs (gigabytes) (or 125,000,000 bytes)

Knowing this, the number of frames per second is as follows:

100 megabit: 12,500,000 / 1538 = 8,127 frames a second (rounded from 8127.438)

1 gigabit: 125,000,000 / 1538 = 81,274 frames a second (rounded from 81274.382)

To determine the maximum amount of data I can send over each network, I multiple the maximum amount of data transmittable excluding headers by the number of frames per second.

For TCP:

Maximum data (1460) bytes * 8127 (frames per second on 100 megabit network) = 11.865 MB/s

Maximum data (1460) bytes * 81,274 (frames per second on 1 gigabit network) = 118.660 MB/s

For UDP:

Maximum data (1472) bytes * 8127 (frames per second on 100 megabit network) = 11.963 MB/s

Maximum data (1472) bytes * 81,274 (frames per second on 1 gigabit network) = 119.635 MB/s

Comparison of Theoretical and Actual Results

All tests were completed on a 1gigabit network using Ethernet. The cables are confirmed CAT6 with a switch acting as the intermediary between two computers in a lab row. At the time of testing network use was low in this lab (I was the only one in there). The Linux tests were completed in Virtual Machines whereas the Windows tests were not.

TCP

The results for TCP were lower than the theoretical. When we look at the amount of data sent over time this is considerable lower, with Windows being much slower than Linux. As stated in the mini observations after the TCP test on the Ubuntu VM's, it appears to be that the Linux Kernel is optimized for TCP communications. This would explain why the data was much faster there than compared to windows. Congestion on each OS uses a different algorithm and one could argue that this algorithm could be more aggressive in Linux, using more of the available bandwidth to complete the task as compared to Windows.

That aside, there is still a big disparity between our hypothetical and the actual results. What we can't see at the application level are the additional handshakes that are completed as well as any resends that the receiving computer would need before completing the transmission on the full block of data. If we were to see these, it is likely that we would get closer to our theoretical maximum amount as this data has not been accounted for. Knowing that there are three handshakes to acknowledge data is sent (or not, which would start the process over) and a fourth to close it, and that the minimum size of an Ethernet frame is 64B (*Minimum and Maximum Ethernet Frame Size*) this could definitely bring us closer to the theoretical.

The blocks were all received in order with none missing, proving to be 100% reliable. TCP being a "connected" protocol enhances this. While UDP can be "connected" this only ensures that the two sockets talking to each other have knowledge of one another. None of the other handshaking occurs. Whereas with TCP this reliability would make sense due to the handshaking that occurs when using TCP and its reliability to ensure that data is always received in the order in which it is sent. That being said, this reliability causes for much more overhead and therefore reduces the amount of data actually being transmitted over time

UDP

UDP had an interesting variance of results. The trend following both tests was that the more data we tried to transmit, the faster UDP was overall. With the significantly less overhead compared to TCP (8 bytes for the header as opposed to the 20 of TCP) and lack of controls around congestion (no handshakes to confirm successful transmission of data AND if it was in order) the speed difference is undeniable.

What was surprising was the vast difference between Linux and Windows. IF we address the amount of out-of-order/dropped blocks, Windows had a much higher drop/ out of order rate than compared to

Linux. Linux proved to be fairly reliable with this protocol overall. Windows would drop much more and place more blocks out of order whereas Linux would place under a full percentage point out of order. When running multiple tests, some cases proved that no blocks in massive quantities were out of order or missing. While researching I could not find an explanation as to why this occurs the way it does.

Windows versus Linux speed wise, Windows proved to be the fastest in sending data overall. We came incredibly close to our theoretical through put with the block size of 10,000 bytes. Accounting for the extra headers, the sent amount was only 3.41MB/s off of our theoretical. While this is still a lot, it was the closest overall and did not seem limited by much of anything. That being said, the reliability of this was 89.65% (blocks missed/blocks received). If we compare to Linux though, our reliability was, at its worse 99.9952% reliable. But, it was significantly slower than all other results. This was 80.16 MB/s slower than the windows comparable at 10,000 bytes per block. It speed was comparable to that of the Windows TCP test at the same size. The reasons for this I cannot explain without performing further network tests. Things that could cause this slow-down could be congestions via the VM for this particular protocol, if there is something that purposely slows this down. Without testing on two native Linux machines I would not be able to explain this in depth. Speculation assumes the VM, the method of networking through the machine to Windows itself.

Conclusion

Given that our foray into networking at this point of the program has not been anything too big, this was a great eye opener. It really exposed the differences in speed and reliability of these two protocols.

Given the amount of research required to truly understand what is going on 'under the covers' is a whole course in itself, and that the time researching this has been nowhere near the years spent that some have/do, my perspective is this:

UDP is fantastic for transmitting a lot of data quickly if you don't need to the same acknowledgments that TCP has built in. As it is presently used for things such as streaming (music, movies, games) it is perfect for these mediums where a small blip can be ignored if it is missed as someone is not likely to notice. Reliability can be built in to ensure a retransmit of missing data, but at that point TCP may be a better option. For anything where I would not depend upon 100% accuracy, UDP would be my go to option.

TCP is just as fantastic, but for different reasons. The dependability of this protocol was proven with these tests. Never once was a block received in the application as out of order or missing. This is as expected, but really proves the reliability of this protocol. For tasks such as sharing files (FTP for example), email, or wanting to ensure that data will be transmitted barring any real network issues (websites…), TCP is the go to for all these reliable items as we would be willing to sacrifice the speed for the reliability in this case.

References

Bibliography

Hedlund, B. (2008, December 19). *How to Calculate TCP throughput for long distance WAN links*. Retrieved from Bradley Hedlund: Stuff and Nonesense: http://bradhedlund.com/2008/12/19/how-to-calculate-tcp-throughput-for-long-distance-links/

Nobel, R. (2011, 06 21). *Actual Throughput on Gigabit Network*. Retrieved from Rickard Nobel: Networking, Windows and VMWare: http://rickardnobel.se/actual-throughput-on-gigabit-ethernet/

poster), v. (. (2014, 06 12). *Why Mongodb performance better on Linux than on Windows?* Retrieved from Stack Overflow: http://stackoverflow.com/questions/24178128/why-mongodb-performance-better-on-linux-than-on-windows

Unknown. (2014, 06 20). *What is the theoretical maximum throughput of a Gigabit Ethernet interface?* Retrieved from NetApp: https://kb.netapp.com/support/index?id=3011938&page=content&locale=en_US

Appendix: Compilation and Usage Instructions

Compilation

Windows

Compile the program in your favourite compiler. The program has been written using Visual Studio 2015 Enterprise. All source files can be located in the 'src' folder, all headers in the 'headers' folder. The source files reference the header folder relatively and can safely be moved together to wherever you want on your machine. The c99 standard must be used to compile the program as well.

Linux

A makefile has been included to compile the program.

All source files can be located in the 'src' folder, all headers in the 'headers' folder. The source files reference the header folder relatively and can safely be moved together to wherever you want on the desktop. The c99 standard must be used to compile the program as well.

Usage

The program must be run from the command line of whichever OS you are using.

The program can be used as a client or a server, depending on the flags used to run it. Either program may be run first. The server will timeout after 5 seconds if nobody initiates a test by attempting to connect or contact the server.

The default is for the program to run as a server (sender) until it is contacted

The following flags may be added to run the program:

-c *IPv4Addr*	-- (indicates client session and IPv4 of server to connect)
-b *Size*	-- (where Size is 1000/2000/5000/10,000 per block (1000 is default))
-u	-- (indicates use of udp, tcp is default)
-l *numBlocks*	-- (where numBlocks indicates the amount of blocks to send (must be a whole number and less than 1 million, 100,000 is default)

Example: *.exe –c 127.0.0.1 –b 2000 –u
This would run the executable as a client, receiving blocks in chunks of 2000 bytes

Evaluation

Document		Self Evaluation	Score
	Completeness	4/ 5	/ 5
	Format	4/ 5	/ 5
	Clarity / Writing	4/ 5	/ 5
	References	4/ 5	/ 5
Document Total			/ 20
Demonstration		Self Evaluation	Score
	Readiness	5/ 5	/ 5
Technical		Self Evaluation	Score
	Results	4/ 5	/ 5
	Theoretical Calculation	5/ 5	/ 5
	UI Quality	(what UI)… 5/ 5	/ 5
	Bug Free Operation	4/ 5	/ 5
	SET Coding Standards	5/ 5	/ 5
Technical Subtotal			/ 25
	Completeness Scale Factor	4/ 5	/ 5
Technical Total (Technical Subtotal * Completeness / 5)			/ 25
Reflection		Self Evaluation	Score
	Self Evaluation Accuracy	5/ 5	/ 5
Assignment Total			/ 55