An Alternative Prototype for Improving Windows' Method of Data Transmission for Copying and Moving Files

Jason P. Sermeno [1], Kaiser Lou A. Sermeno [2]

Abstract: Empirical studies comparing the speed or rate of data transmission of various file transfer software versus the Windows default file transfer utility have rarely been attempted. This study describes an implementation of the file transfer utility (prototype) integrating the concepts of multi-threading, dynamic buffer partitioning, and Tel's knapsack algorithm and conducting an experiment against the Windows 7 default file transfer utility (commercial) with regards to their transmission rate. The study presents the motivation, experimental approach, issues in attempting this type of empirical study, and a summary of the experimental results and insights gained.

Keywords: Data Transfer, Data Transmission, Multi-Threading, Knapsack Algorithm, Buffer Management, Speed Test

1. Introduction

Data transmission, digital transmission, or file transfers are nearly universal concerns among computer users. Home users download software updates and upload backup images (or delta images), share multimedia content among peers, and enterprise users often distribute software packages to cluster or client machines. Operating systems such as Windows, Linux, and/or Macintosh are already equipped with these features that allow users to manage files of different sizes. This utility is commonly used and accessed by users in their daily lives, but when software evolves, its resource files, dynamic link libraries, systems, binary files, or the ones that have been generated by the application, grows along with it. When these files are needed to be relocated, the need for an optimized data transfer process is critical for it to obtain functionality throughout the system.

Previous versions of Windows operating systems, such as Windows 98, possess several disadvantages compared to their latest version [1][2]. It is slow, and you can't pause and resume a transfer process [3]. But basically, these utilities are better with small-sized files because the system implements a very small buffer to read and write a stream of data [4]. However, this type of

[1] College of Computer Studies, University of Antique, Sibalom, Antique, Philippines
Email: jason.sermeno@antiquespride.edu.ph
[2] Pascual M. Osuyos Memorial High School, Tobias Fornier, Antique, Philippines
Email: klasermeno@gmail.com

Received [February 20, 2022]; Revised [April 10, 2022]; Accepted [May 12, 2022]

Published by InnoCon Publishing
ISSN 2704-4440

implementation may slow down the transmission process and eventually wear the hard disk down due to an enormous number of movements of the hard disk head when moving or copying large-size files [5][6].

Algorithmic designs for data transmission have influenced a lot of developers to redesign the model. Most of the concepts focus on memory or buffer management, scheduling algorithms, and multiprocessing.

This study was motivated by the challenge of designing a prototype that integrates the concepts of multi-threading, dynamic buffer partitioning, and the knapsack algorithm. It also aims to find out if there were significant differences between both utilities in terms of speed and rate of success in resuming a halt-state process during transmission. For these reasons, the default Windows 7 file transfer utility was selected to be tested against the prototype for comparison.

To perform this study, a prototype was constructed, and similar case scenarios were conducted on both subjects on a single operating system platform. The hypothesis was that both utilities have no significant differences in their mean speed of transferring a file with regards to file size, the volume of files being transmitted, and the physical orientation of the two devices during transmission. The results of this study supported this hypothesis.

The rest of this paper is organized as follows: Section 2 briefly presents the contributing approaches to the design of several file transfer utilities and some existing file transfer utilities similar to the prototype being built; Section 3 describes the design of the prototype and the experiments done; Section 4 details the obtained results and the discussion of observations and insights; and Section 5 presents the conclusions and future directions of the study.

2. Review of Related Literature

The opportunity of obtaining a high transfer rate between endpoints in data transmission depends on how the system manages to schedule the read-write process, maintain the available buffer, and optimize the isolation of files. We could establish multiple executions by buffering the partitioned files in variable-sized partitioned blocks and writing them simultaneously on disks. However, there are considerable bodies of work that have explored better ways to accomplish high-speed file transfers.

2.1 Contributing Approaches

While it might be true that low CPU utilization can actually indicate a problem in the input/output (I/O) subsystem, making the CPU spend more time waiting than running. A slow disk system can drag down the performance of all programs when virtual memory paging is involved. Thus, optimal disk performance is critical to the system's throughput.

2.1.1 Buffer Management Schemes

Memory access times may range from 1 to 20 clock cycles depending on whether the access hits the CPU caches, whereas the latency of a disk access ranges in the tens to hundreds of milliseconds [7]. Processes here must wait for I/O to complete before proceeding. To avoid deadlock, certain pages must remain in main memory during this process.

No buffer, single buffer, double buffer, and circular buffer are some of the well-known methods by which the operating system manages to cope with processes from its local or external devices. These basic methods of managing the buffer have influenced and motivated a large body of research in the area of storage, aiming to increase the transfer speed, availability, and accessibility of data. One thread of research led to the development of the Unified Buffer Management (UBM) scheme, which attempts

to upgrade the performance of the Least Recently Used (LRU) block replacement scheme by making use of reference regularities such as sequential and looping references [8]. Another similar implementation, called Software Cache Unification (SCU), allows applications to make better use of distributed caches by combining the speed of local caches with the large aggregate capacity of a shared cache [9]. But these papers were focused on improvements to database systems, network-, and microchip-related studies.

Buffering is a technique that smooths out peaks in I/O demand. However, no amount of buffering will allow an I/O device to keep pace with a process indefinitely when the average demand of the process is greater than the I/O device can handle. Even with multiple buffers, all of the buffers will eventually fill up, and the process will have to wait after processing each chunk of data. However, the efficiency and effective method for managing the buffer may depend on how it is partitioned to maximize the use of available work space for applications.

2.1.2 Buffer and File Partitioning Strategies

Partitioning buffers and/or files have proven to have a great impact on the performance of certain applications. It is a powerful method of managing records in a relational database scheme. One particular partitioning strategy, called optimal splitting, illustrates the division of relations in a database into several partitions [10]. But this strategy has nothing to do with the block size, except that it is fixed. This algorithm is beneficial when there is a heavy data skew, but in other situations, small block sizes make it expensive [11][12].

There are several partitioning strategies that exist. It may either be classified as no splitting, partial, one-pass, or multi-pass partitioning. These traditional strategies, however, are employed in various situations depending on the size of the operands that are related to the work space area. With the traditional splitting strategy, there is a fixed size of memory for each group. When a new tuple arrives, it is moved to its block buffer, and when the buffer is full, it is written to disk [10][11][13][14].

However, in a multiprogramming environment, when there is a variety of I/O activity and a variety of process activity to service, file and buffer partitioning is one tool that can increase the efficiency of the operating system and the performance of individual processes. This method can be further improved by improving the schedule of the read-write or I/O process of a certain disk.

2.1.3 Disk Scheduling Algorithms

The details of the disk I/O operation depend on the computer system, the operating system, and the nature of the I/O channel and disk controller hardware. Most traditional disk scheduling algorithms, such as First-Come-First-Serve (FCFS), Shortest-Service-Time-First (SSTF), SCAN, C-SCAN, LOOK, and CLOOK, are designed to reduce disk-seek time and increase its throughput. The FCFS algorithm performs operations in the order in which the task arrives; however, the performance of this algorithm is poor. SSTF reduces the total seek time compared to FCFS. The disadvantage of this scheme is starvation, where the read/write head stays in one area of the disk if it is very busy [15]. Some studies show that these algorithms do not consider the real-time constraints of I/O tasks and, therefore, are not suitable to be applied directly to a real-time system [16][17][18][19]. In the SCAN algorithm, the disk arm starts at one end of the disk and moves toward the spindle, servicing requests as it reaches each cylinder until it gets to the spindle. From this end, the direction of head movement is reversed, and servicing continues [17][12]. C-SCAN scheduling is a variant of SCAN designed to provide a uniform wait time. Like SCAN, C-SCAN moves the head from one end of the disk to the spindle, servicing the requests along the way. When the head reaches the spindle, it immediately returns to the beginning of the disk without servicing any requests on the return trip [20][12][21]. The LOOK algorithm is also a

variant of SCAN. In this algorithm, the disk head does not move inward or outward when there is no request in that direction. It performs better than SCAN when the load is low, but it is equivalent to SCAN when the load is high. A variant of LOOK scheduling is C-LOOK. C-LOOK moves the head in one direction from its current position. After serving all the requests in the current direction, the disk head starts to serve the first request on the other end without serving the requests in its return trip. It provides a more uniform wait time for the requests [22].

Other scheduling algorithms had been devised and analyzed by several researchers in some areas of optimization. For example, a novel representation scheme for rotational position optimization that reduces the manufacturing cost per drive was designed by Burkhard and Palmer in 2002 [23]. In 2007, Bachmat [24] considered the problem of estimating the average tour length of the asymmetric TSP arising from the disk scheduling problem with a linear seek function and a probability distribution on the location of I/O requests. In 2008, Povzner's team (2008) showed that by reserving disk resources in terms of utilization, it is possible to create a disk scheduler that supports the reservation of nearly 100% of the disk resources, provides arbitrarily hard or soft guarantees depending upon application needs, and yields efficiency as good as or better than best-effort disk schedulers tuned for performance [14]. Dimitrijev presented a semi-preemptible I/O, which divides disk I/O requests into small temporal units of disk commands to improve the pre-emptibility of disk access. The evaluation of this prototype system showed that semi-pre-emptible I/O substantially improved the pre-emptibility of disk access with little loss in disk throughput and that pre-emptive disk scheduling could improve the response time for high-priority interactive requests [25][26].

Another disk scheduling method proposed by Bonyadi [27] was based on a genetic algorithm that considers making a span and the number of missed tasks simultaneously. In his method, a new coding scheme is presented that employs crossover, mutation, and a penalty function for fitness. Its parameters, such as the number of chromosomes in the initial population, mutations, and crossover probabilities, had been adjusted by applying it to some sample problems. The algorithm had been tested on several problems, and its results were compared with traditional methods. Experimental results showed that the proposed method worked very well and excelled in most related works in terms of miss ratio and average seeks.

The choice of scheduling algorithm depends on expected performance and implementation complexity [28][29]. The I/O architecture is designed to provide a systematic means of controlling information and to provide the operating system with the information it needs. The I/O function is broken into a number of layers, with the lower layer dealing with details closer to the physical functions to be performed and the higher layers dealing with I/O in a logical and generic fashion. A key aspect of I/O is the use of buffers rather than application processes. Buffering smooths out the difference between the internal speed of the computer system and the speed of the I/O device, while disk scheduling is used to improve the I/O performance of the disk.

2.2 Existing File Transfer Utilities (Prior Arts)

The biggest factor affecting the speed of any file transfer utility is what you're copying from or to, such as hard drives, SSDs, USB sticks, networks, *etc*. Most previous versions of Windows had never been as efficient as they could be when dealing with these operations. There have been several file transfer utilities that have been developed and are available on the Internet [30]. FastCopy [31][32] is one particular tool developed by Shirouzu Hiroaki, that claims to offer fast transfer of files between hard drives and copying of files within the same drive. The read-write processing depends on the locations of the source and destination directories. When both the source and destination directories are located on different hard drives, reading and writing are processed respectively in parallel by separate threads.

But when they are located on the same hard drives, reading is processed until the big buffer fills and immediately starts the writing process in bulk. The utility doesn't hog resources because Multicore-aware File Cache (MFC) was not employed in the design; rather, it uses Win32 API and C Runtime resources only.

TeraCopy [33][34] is another utility initially designed by Code Sector in 2007 and released in early November 2011. The application was designed to move or copy files faster and has more functions than the native tool. It uses dynamically adjusted buffers to reduce seek times and asynchronously processes the read-write operation between two physical hard drives. The utility supports pause and resume features, but it can only resume the process if the utility is instructed to pause for a certain period of time.

Another useful file copy utility is the Unstoppable Copier [35][36], designed and developed by Roadkil in 2007. The utility was designed to copy and recover data from a physically damaged medium, such as CDs or DVDs. Other utilities have similar functionalities but use a different approach. Their throughput rate depends on the number of available features that are used to manage and tune up the transfer processes.

3. Methodology

This study had undergone prototyping and experimental methods. The prototype had to be reviewed, redesigned, and implemented before it underwent an experimental test with our existing Windows 7 default file transfer utility. The following subsections describe the concept and design of the prototype and how the experiment was conducted.

3.1 The Prototype

Initially, the prototype was derived from an initial concept illustrated in a block diagram, as shown in Figure 1.

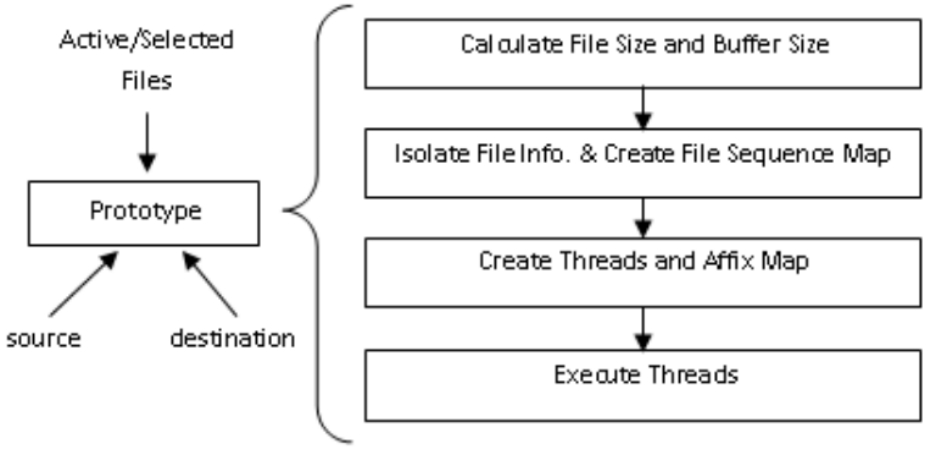

Figure 1. Initial concept of the prototype under study

Reviews of existing concepts in multi-threading, buffer management, isolation, and scheduling algorithms had allowed us to come up with a more detailed and perspective view of how the system should be designed. Figure 2 shows a more detailed conceptual design of the prototype.

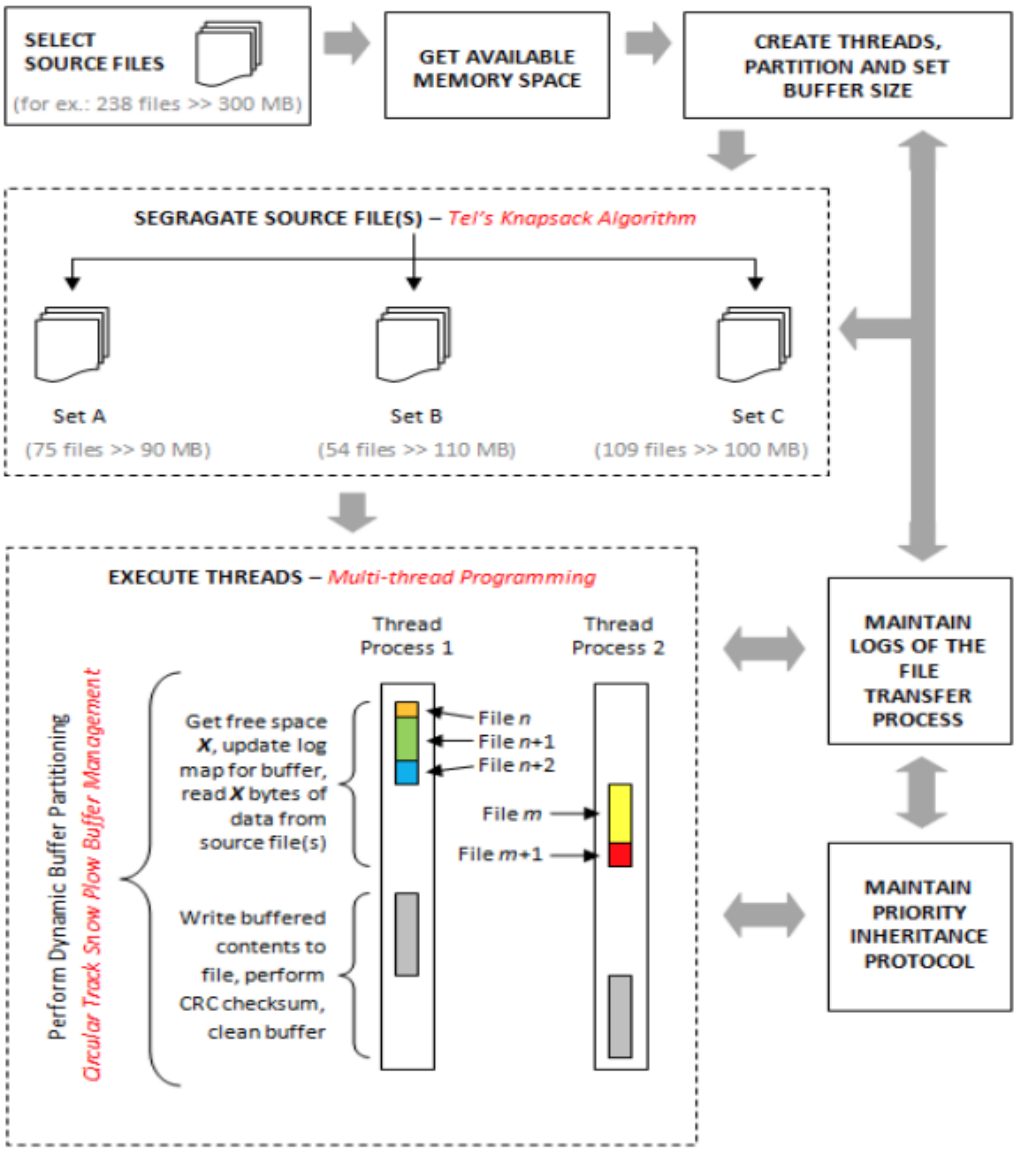

Figure 2. Conceptual Framework of the Prototype

The idea here is to determine and manage the available space left in the memory in order to calculate and allocate the right amount of space for our partitioned source files to be buffered, which will be taken care of by the system. In this architecture, an estimate of 1/2 of the total available space will be used to allocate the partitioned source files, leaving the other half free for other applications to use it for other purposes (in this case, they won't suffer the performance of other applications during the transfer process). With the determined buffer size, the system partitions it into several blocks where the isolated source files are to be allocated. This method benefits from small to large file transfers and offers multiple files to be fetched and written to the destination in a synchronized manner. The source files had to be organized in an optimized manner to allow each thread of execution to independently maintain and manage the buffer in an optimized way. Protocols for setting up the priority flags for each file had to be considered in order to deal with the problematic scenarios in scheduling when unrecognizable chunks are being processed in threads. This will allow the utility to stop a thread or skip over bad chunks of files without interfering with the rest of the threads in Figure 2. The initial concept of the prototype under study details the process. During transmission, the system monitors the progress, such as the

details of the partitioned blocks (*e.g.*, number of blocks created, block size, *etc*.), the order in which the files are segregated and grouped together in threads, and a list of chunk information (*e.g.*, memory and disk addresses, pointers, *etc*.) in order to keep track of the files being transmitted and support the pause and resume feature of the prototype.

Since there are tradeoffs between speed and the number of functionalities, the prototype is designed with a small number of features to increase speed. Figure 3 shows the top-level architectural design and describes the basic internal components of the prototype. In this figure, the main application consists of only a small number of features to increase speed.

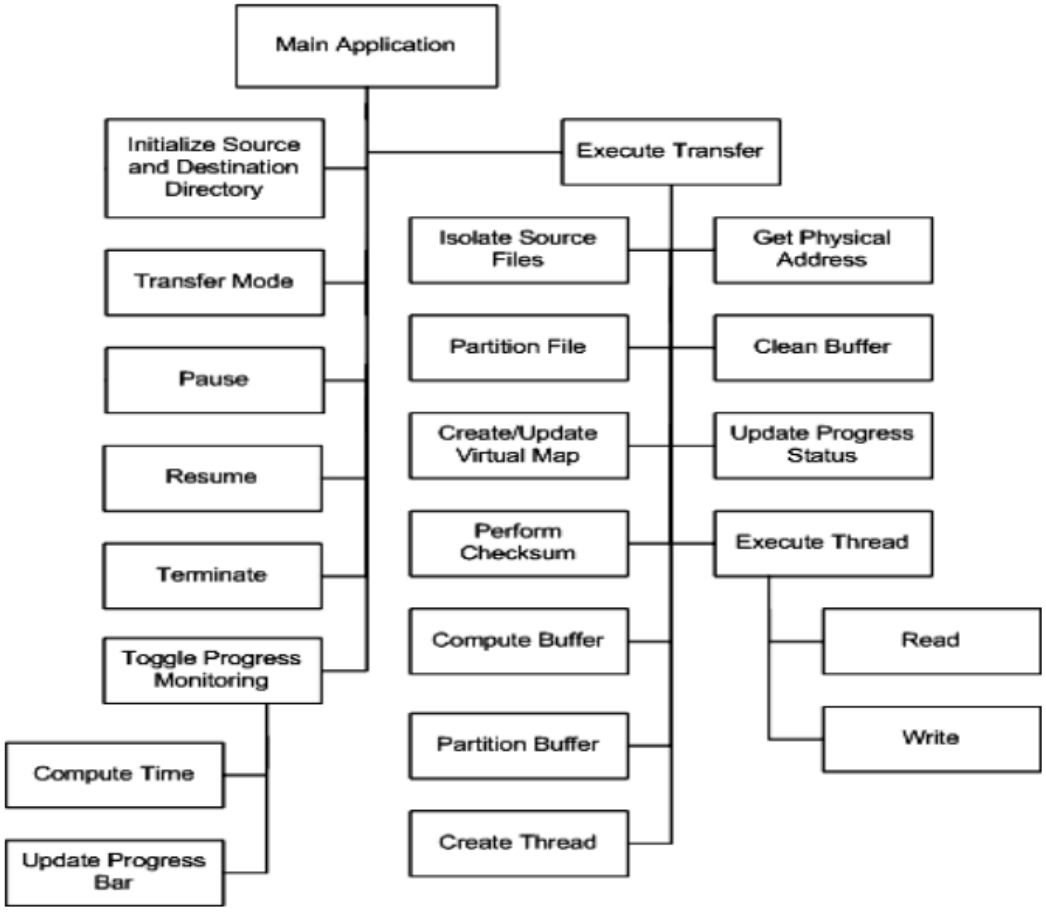

Figure 3. Prototype's top-level architectural design

Figure 4 shows a sequence diagram illustrating the general view of how the prototype operates during program execution. It basically shows how the prototype responds to user actions and/or events triggered by the application and other corresponding classes or objects.

Figure 5 shows another perspective view of the prototype, illustrating the transmission process in detail. In this design, the XCopy is housed in threads. Before the threads execute, they are set with priority information that will be used in managing the priority inheritance protocol. All read and write operations are done by the XCopy objects, and the thread objects run them simultaneously in the background. During the read-write process, the application receives the status from the XCopy objects about the current file, the progress of the transmission process, and several Boolean flags representing indicators. When the transmission process is put to a halt, the prototype temporarily freezes and marks

the section of progress where it was paused, thus giving us options on whether we would like to continue, cancel, or save the ongoing session.

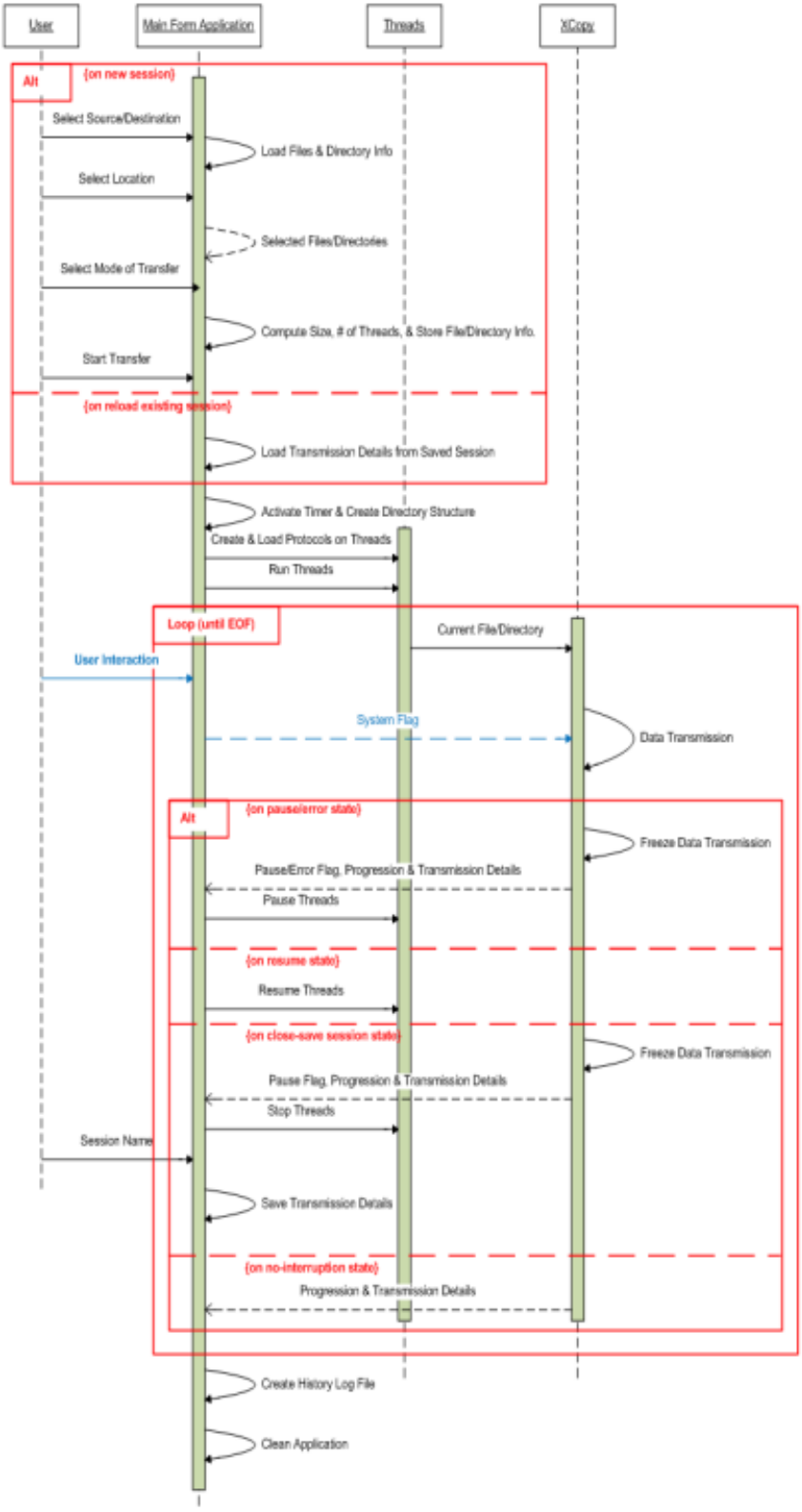

Figure 4. A sequence diagram showing the general view of the prototype

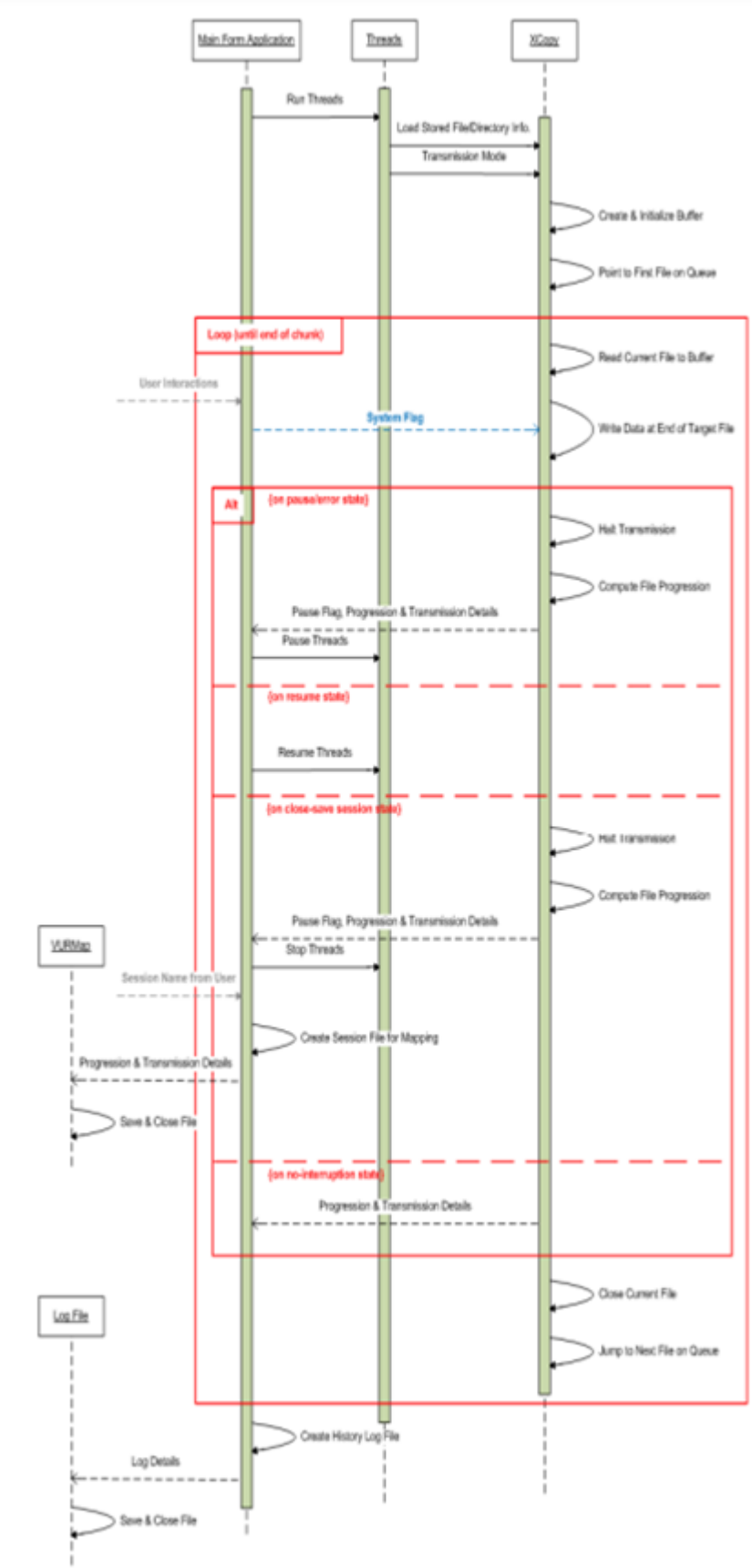

Figure 5. Detailed view of the prototype's transmission process

The implementation of the design led us to the result of our final prototype. The prototype had undergone a series of function and use case tests before being compared to the commercial utility. Figure 6 illustrates our sample output of the prototype during the transmission process.

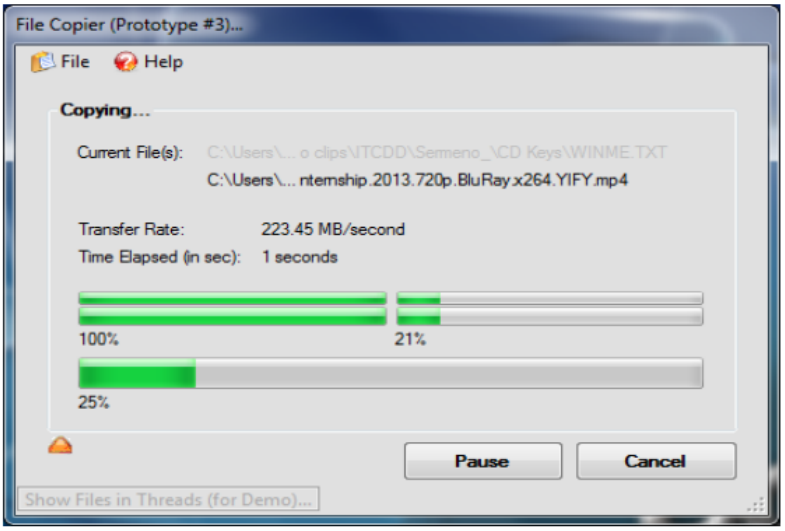

Figure 6. Graphical User Interface (GUI) of the prototype

3.2 The Experiment

The experiment was divided into four modes of simulation. The first three sets of tests were conducted to determine the average speed of both utilities with respect to the mode of transfer parameters. The last set of tests was conducted to determine the mean rate of success using the pause-resume feature between two utilities.

The file size-based test was the first test conducted wherein the utilities were tested with 9 sets of files based on the incrementing file size from 25MB to 1.5GB of data on a partition-to-partition basis. The second test was the volume-based test, where the utilities were tested with 9 sets of 600MB files based on the incrementing number of assorted files from 1 to 1500 on a partition-to-partition basis. The third test was the orientation-based test, wherein both utilities were tested by sending a 600MB file in five different orientations of data transmission, such as partition-to-ext. partition or vice versa. Again, the default file transfer utility was first tested and was followed by the integrated file transfer utility.

The last test was the pause-resume test, in which both utilities were tested based on their capability of pausing and resuming a file transfer process. The results were gathered and plotted in a single-group experimental design to determine the significance difference of the mean speed of the two utilities by applying the t-test.

The type and size of the files were carefully selected as samples to be used in the simulation process. To produce fair results, the current settings for each type of test were calibrated and tested three times. The results were then determined by getting their average rounded to the nearest hundredth decimal place to be significant.

4. Results and Discussion

The results of the experiment were carefully repeated and recorded. Both utilities were assessed by third-party software that measures the duration of the file copy process running in parallel on both utilities. At the end of every experimental test performed, the results were recorded in tabular form and

plotted graphically to show the differences between the utilities. The following Tables 1, 2, 3, and 4 and Figures 7, 8, 9, 10, 11, and 12 show the results and discussions of the post-activities.

Table 1. Result of the file size-based Test

File Transfer Parameter: @ 1 File	**Prototype**		**Windows 7 File Transfer Utility**	
	Time elapsed (in sec.)	Speed (in MB/s)	Time elapsed (in sec.)	Speed (in MB/s)
25 MB	2.25	11.11	2.25	11.11
50 MB	4.00	12.50	3.75	13.33
100 MB	7.75	12.90	7.75	12.90
300 MB	21.50	13.95	24.75	12.12
600 MB	43.75	13.71	45.00	13.33
900 MB	67.25	13.38	74.75	12.04
1100 MB	77.75	14.15	85.50	12.87
1300 MB	91.00	14.29	98.50	13.20
1500 MB	107.00	14.02	121.25	12.37
Average Speed	**13.33 MB/s**		**12.59 MB/s**	

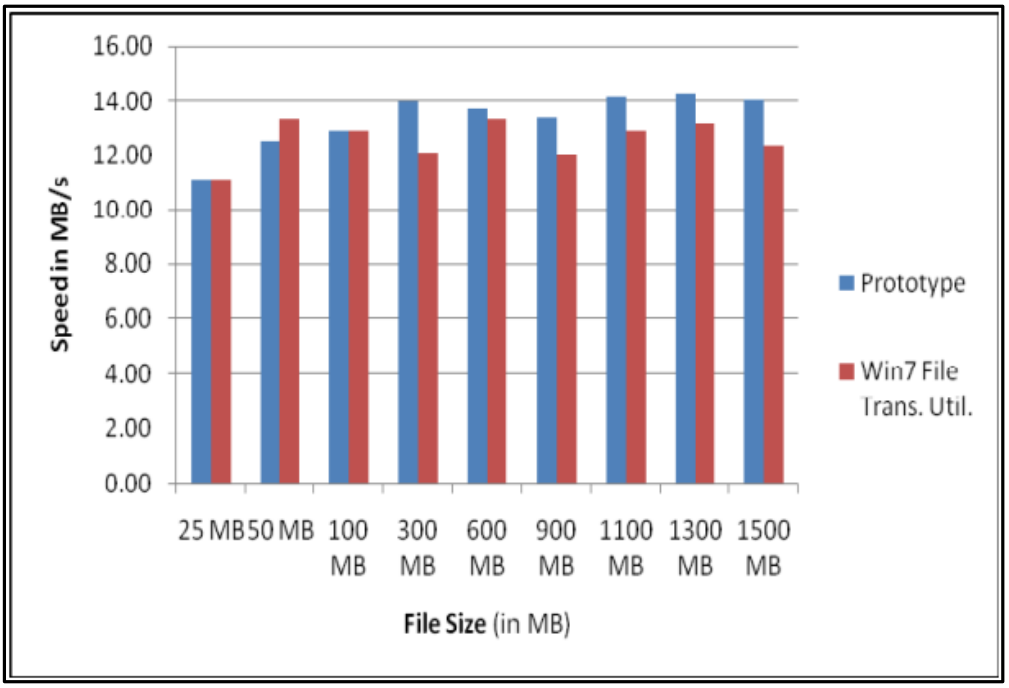

Figure 7. Graphical presentation of result of the filesize-based test

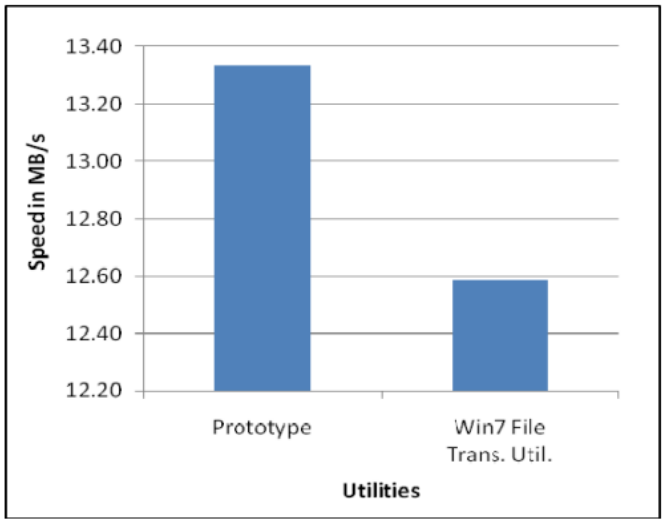

Figure 8. Summary of the file size-based test

In this test, the prototype performed better than Windows 7 on most levels of file sizes. It shows that the prototype performed better with files larger than 100MB than Windows, as shown in Table 1 and Figures 7 and 8. In addition, the mean speed of the prototype is 5.95% faster than that of the commercial utility.

Table 2. Result of the volume-based test

File Transfer Parameter: Number of Files @ 600 MB File	**Prototype**		**Windows 7 File Transfer Utility**	
	Time elapsed (in sec.)	Speed (in MB/s)	Time elapsed (in sec.)	Speed (in MB/s)
1 File	43.50	11.49	43.75	11.43
50 Assorted Files	44.00	11.36	44.00	11.36
100 Assorted Files	44.00	11.36	44.25	11.30
300 Assorted Files	44.25	11.30	44.25	11.30
600 Assorted Files	44.25	11.30	44.50	11.24
900 Assorted Files	44.75	11.17	45.75	10.93
1100 Assorted Files	46.25	10.81	47.25	10.58
1300 Assorted Files	46.25	10.81	47.75	10.47
1500 Assorted Files	46.75	10.70	49.25	10.15
Average Speed	**11.15 MB/s**		**10.97 MB/s**	

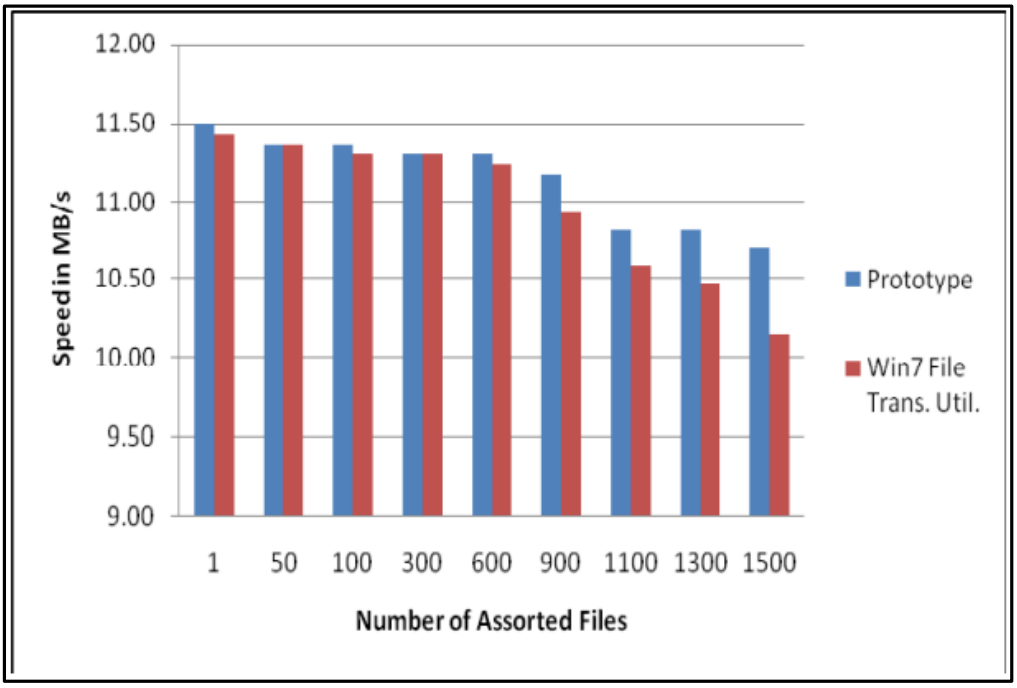

Figure 9. Graphical presentation of result of the volume-based test

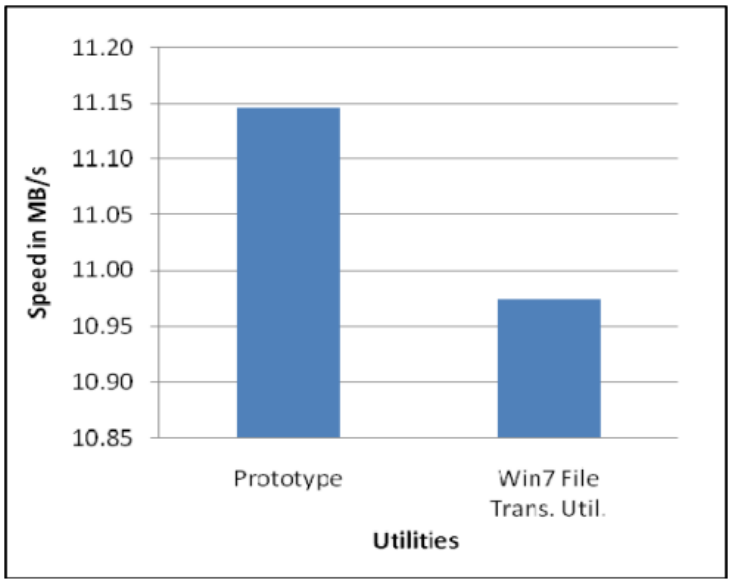

Figure 10. Summary of the volume-based test

In this test, the prototype also performed slightly better than Windows 7 on all levels of the number of assorted files, as shown in Table 2 and Figures 9 and 11. It shows that the mean speed of the prototype is 1.57% faster than that of the commercial one.

Table 3. Result of the orientation-based test

File Transfer Parameter: Storage Medium @ 600 MB File	**Integrated File Transfer Utility (IFTU)**		**Windows 7 File Transfer Utility (W7FTU)**	
	Time elapsed (in sec.)	Speed (in MB/s)	Time elapsed (in sec.)	Speed (in MB/s)
HDD-to-HDD	43.75	13.71	44.25	13.56
Ext. Drive-to-Ext. Drive	47.25	12.70	47.75	12.57
HDD-to-Ext. Drive	47.75	12.57	47.25	12.70
Ext. Drive-to-HDD	48.00	12.50	47.50	12.63
Over Local Area Network	51.75	11.59	49.50	12.12
Average Speed	**12.61 MB/s**		**12.72 MB/s**	

In this third set of tests, Windows managed to do better than the prototype on almost all levels of file transfer with regards to the accessibility of the file from different locations on certain mediums, as shown in Table 3 and Figures 11 and 12. It is intriguing that Windows performs better on local area network file transfers than the prototype. The mean speed of the prototype is 0.79% slower than the commercial utility.

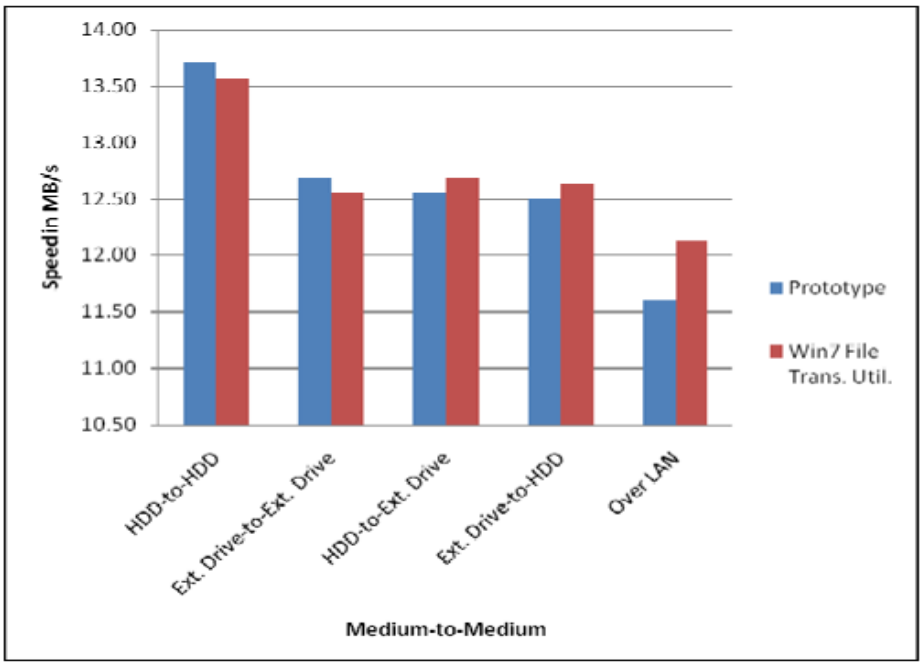

Figure 11. Graphical presentation of results of the orientation-based test

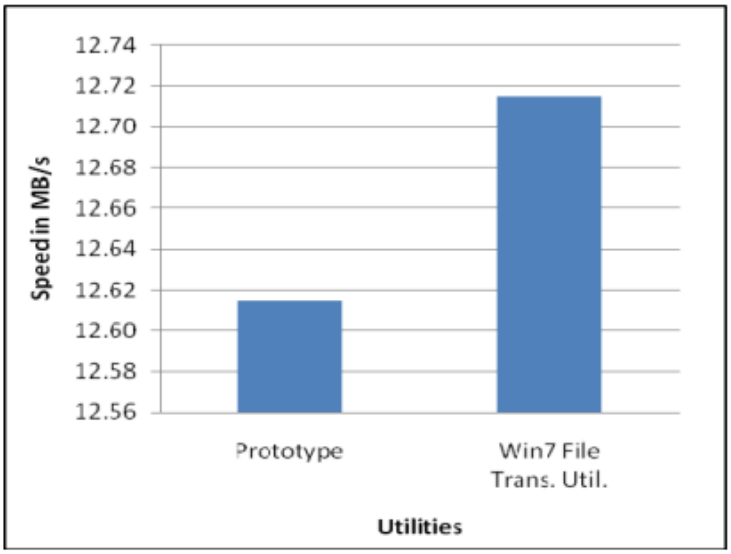

Figure 12. Summary of the orientation-based test

Table 4. Result of the pause-resume-based test

Synchronized Pause-Resume Test with...		Success Flags @ Trial #1	Trial #2	Trial #3	Rate of Success
Filesize-Based Test @	25 MB	Yes	Yes	Yes	100%
	50 MB	Yes	Yes	Yes	100%
	100 MB	Yes	Yes	Yes	100%
	300 MB	Yes	Yes	Yes	100%
	600 MB	Yes	Yes	Yes	100%
	900 MB	Yes	Yes	Yes	100%
	1100 MB	Yes	Yes	Yes	100%
	1300 MB	Yes	Yes	Yes	100%
	1500 MB	Yes	Yes	Yes	100%
				Average Rate of Success >>	100%
Quantity-Based Test @	1 File	Yes	Yes	Yes	100%
	50 Assorted Files	Yes	Yes	Yes	100%
	100 Assorted Files	Yes	Yes	Yes	100%
	300 Assorted Files	Yes	Yes	Yes	100%
	600 Assorted Files	Yes	Yes	Yes	100%
	900 Assorted Files	Yes	Yes	Yes	100%
	1100 Assorted Files	Yes	Yes	Yes	100%
	1300 Assorted Files	Yes	Yes	Yes	100%
	1500 Assorted Files	Yes	Yes	Yes	100%
				Average Rate of Success >>	100%
Physical Orientation-Based Test @	HDD-to-HDD	Yes	Yes	Yes	100%
	Ext. Drive-to-Ext. Drive	Yes	Yes	Yes	100%
	HDD-to-Ext. Drive	Yes	Yes	Yes	100%
	Ext. Drive-to-HDD	Yes	Yes	Yes	100%
	Over LAN	No	No	Yes	33%
				Average Rate of Success >>	87%
				Overall Average Rate of Success >>	96%

In this type of test, Windows wasn't qualified to perform the test since it doesn't have the ability to pause and resume a transfer process on all levels of transfer parameters, as shown in Table 4. The prototype, having been tested, turned out to be pretty successful on the first two sets of tests. The result shows that the prototype has managed to carry out an overall average rate of 96% successful operations on all aspects of the file transfer parameters.

To answer the significant differences between both utilities, we finally consolidated the data and applied statistical treatment to each respective parameter. Tables 5, 6, and 7 show the results and interpretation of this research using a t-test on paired samples.

Table 5. A t-test result of the file size-based test

	Prototype	**Windows 7**
Mean	13.33444444	12.58555556
Variance	1.050427778	0.550477778
Observations	9	9
Pearson Correlation	0.520104298	
Hypothesized Mean Difference	0	
df	8	
T Stat	2.496433279	
P(*T*<=*t*) two-tail	0.037147927	
t Critical two-tail	2.306004133	

Interpretation (Table 5): The computed t-value obtained is 2.496433279 with 8 degrees of freedom (df). This value is significant at the 5% level of confidence due to a greater than 2.306004133 tabular value of .05% level at df 8 (*i.e.*, *t*.05(8) = 2.306004133). This means that the mean speed of the prototype and the Windows 7 default file transfer utility really differ from each other because the prototype is faster than Windows with regards to file size.

Table 6. A t-test result of the volume-based test

	Prototype	**Windows 7**
Mean	11.14559875	10.9734993
Variance	0.086471149	0.21532379
Observations	9	9
Pearson Correlation	0.983119988	
Hypothesized Mean Difference	0	

df	8	
T Stat	2.820986664	
P(*T*<=*t*) two-tail	0.022459728	
t Critical two-tail	2.306004133	

Interpretation (Table 6): The computed t-value obtained is 2.820986664 with 8 degrees of freedom (df). This value is significant at the 5% level of confidence due to being less than 2.306004133, a tabular value, of .05% level at df 8 (t.05(8) = 2.306004133). This means that the mean speed of the prototype and the Windows 7 default file transfer utility really differ from each other because the prototype is much faster than the commercial utility with regards to the number of files being transferred at once.

Table 7. A t-test result of the orientation-based test

	Prototype	**Windows 7**
Mean	12.61446927	12.71519417
Variance	0.568273192	0.273766114
Observations	5	5
Pearson Correlation	0.971170143	
Hypothesized Mean Difference	0	
df	4	
T Stat	– 0.817391889	
P(*T*<=*t*) two-tail	0.459594194	
t Critical two-tail	2.776445105	

Interpretation (Table 7): The computed *t*-value obtained is 0.817391889 with 4 degrees of freedom (df). This value is not significant at the 5% level of confidence due to being less than 2.776445105, a tabular value of .05% at df 4 (*t*.05(4) = 2.776445105). This means that the mean speed of the prototype is slower than the Windows 7 default file transfer utility in terms of the physical orientation of the data transfer.

5. Conclusion and Future Directions

Empirical evidence was presented that supports our concept of integrating multi-threading, dynamic buffer partitioning, and Tel's knapsack algorithm in constructing a file transfer prototype as an alternative and efficient way of copying or moving files on the Windows platform. The result of the experiment led us to the conclusion that there is a significant difference in the transmission rate between the prototype and the Windows 7 default file transfer utility with regards to the size of files and the

number of files being transmitted, but both are not significantly different in terms of the physical orientation of the transmission process.

Although this study was tested and compared against only one subject, measuring only a small problem domain, we look forward to using the insights and experience gained to plan more elaborate studies comparing the prototype with other related third-party software.

References

[1] Vogons, "*[Resolved] Windows 98 lag when copy files*," www.vogons.org/viewtopic.php?t=53578 (Accessed July 20, 2021).

[2] M. Muchmore, "*Windows, macOS, Chrome OS, or Linux: Which Operating System Is Best?*," www.pcmag.com/picks/windows-vs-macos-vs-chrome-os-vs-ubuntu-linux-which-operating-system-reigns (Accessed July 20, 2021).

[3] Microsoft Community, "*Sharing files between Windows 98 SE and Windows Vista Home Premium*," www.answers.microsoft.com/en-us/windows/forum/all/sharing-files-between-windows-98-se-and-windows/ad0b389c-2134-40d1-ae6a-aaed882cc3b4 (Accessed July 20, 2021).

[4] Tom's Hardware, "*Can't transfer files to an external hard drive*," www.forums.tomshardware.com/threads/cant-transfer-files-to-an-external-hard-drive.3399989/ (January 13, 2022).

[5] AOMEI Backupper, "*Solved: External Hard Drive Freezes When Copying Files*," www.ubackup.com/synchronization/external-hard-drive-freezes-when-copying-files-8523.html (Accessed July 20, 2021).

[6] Microsoft Community, "*Windows 7 large file transfer external hard drive freeze*," www.answers.microsoft.com/en-us/windows/forum/all/windows-7-large-file-transfer-external-hard-drive/3d38ba39-a6f8-4e7a-a109-22693078d90e (Accessed July 20, 2021).

[7] J. L. Hennesy and D. A. Patterson, "*Computer Architecture: A Quantitative Approach*," 3rd edition, Morgan Kaufmann: Burlington, Massachusetts, USA, 2002.

[8] J. M. Kim, J. Choi, J. Kim, S. Noh, S. Min, Y. Cho, and C. S. Kim, "*A Low-Overhead High-Performance Unified Buffer Management Scheme that Exploits Sequential and Looping References*," in Proceedings of the 4th USENIX Symposium on Operating System Design and Implementation, vol. 4, October 2000, pp. 119-134, Corpus ID: 84765.

[9] S. B. Wickizer, M. F. Kaashoek, R. Morris, and N. Zeldovich, "*A Software Approach to Unifying Multicore Caches*," Massachusetts Institute of Technology, www.people.csail.mit.edu/nickolai/papers/boyd-wickizer-scu-tr.pdf (January 13, 2022).

[10] M. Kitsuregawa, H. Tanaka, and T. Motooka, "*Application of Hash to Data Base Machine and its Architecture*," New Generation Computing, vol. 1, March 1983, pp. 63-74, doi: 10.1007/BF03037022.

[11] L. D. Shapiro, "*Join Processing in Database Systems with Large Main Memories*," ACM Transactions on Database Systems, vol. 11, no. 3, August 1986, pp. 239-264, doi: 10.1145/6314.6315.

[12] E. G. Coffman Jr. and M. Hofri, "*On the Expected Performance of Scanning Disks*," SIAM Journal on Computing, vol. 11, no. 1, 1982, pp. 60-70, doi: 10.1137/0211005.

[13] K. Bratbergsengen, R. Larsen, O. Risnes, and T. Aandalen, "*A Neighbor Connected Network for Performing Relational Algebra Operations*," in Proceedings of the Fifth Workshop on Computer Architecture for Non-Numeric Processing, Association for Computing Machinery, New York, NY, USA, March 1980, pp. 96-105, doi: 10.1145/800083.802697.

[14] A. Povzner, T. Kaldewey, S. Brandt, R. Golding, T. Wong, and C. Maltzahn, "*Efficient guaranteed disk request scheduling with fahrrad*", ACM SIGOPS Operating Systems Review, vol. 42, no. 4, May 2008, pp. 13-25, doi: 10.1145/1357010.1352595.

[15] M. Seltzer, P. Chen, and J. Ousterhout, "*Disk scheduling revisited*," USENIX Tech Conference, 1990, pp. 313-324, Corpus ID: 62186399.

[16] T. S. Chen, J. A. Stankovic, J. F. Kurose, and D. Towsley, "*Performance evaluation of two new disk scheduling algorithms for real-time systems*," Journal of Real-Time System, vol. 3, no. 3, September 1991, pp. 307-336, doi: 10.1007/BF00364960.

[17] T. S. Chen, W. P. Yang, and R. C. T. Lee, "*Amortized analysis of some disk scheduling algorithms: SSTF, SCAN and N-StepSCAN*," BIT Numerical Mathematics, vol. 32, December 1992, pp. 546-558, doi: 10.1007/BF01994839.

[18] M. Hofri, "*Disk scheduling: FCFS vs. SSTF revisited*," Communications of the ACM, vol. 23, no. 11, November 1980, pp. 645-653, doi: 10.1145/359024.359034.

[19] B. L. Worthington, G. R. Ganger, and Y. N. Patt, "*Scheduling Algorithms for Modern Disk Drives*," in the Proceedings of the ACM Sigmetrics Conference, May 1994, pp. 241-251, doi: 10.1145/183018.183045.

[20] E. G. Coffman Jr. and C. J. M. Turnbull, "*A note on the relative performance of two disk scanning policies*," Information Processing Letters, vol. 2, no. 1, March 1973, pp. 15-17, doi: 10.1016/0020-0190(73)90019-7.

[21] Z. Dimitrijevi, R. Rangaswami, and E. Y. Chang, "*Systems Support for Preemptive Disk Scheduling*," IEEE Transactions on Computers, vol. 54, no. 10, October 2005, pp. 1314-1326, doi: 10.1109/TC.2005.170.

[22] J. M. Sohn and G. Y. Kim, "*Earliest-Deadline-First Scheduling on Non-Preemptive Real-Time Threads for Continuous Media Server*," in Proceedings of International Conference on High-Performance Computing and Networking, LNCS, vol. 1225, 1997, pp. 950-956, doi: 10.1007/BFb0031666.

[23] W. Burkhard and J. D. Palmer, "*Rotational Position Optimization (RPO) Disk Scheduling*", Technical Report, University of California at San Diego, USA, July 2001.

[24] E. Bachmat, "*Average Case Analysis of disk scheduling, increasing subsequences and space-time Geometry*," Algorithmica, vol. 49, no. 3, September 2007, pp. 212-231, doi: 10.1007/s00453-007-9017-6.

[25] J. R. Iyengar, P. D. Amer, and R. Stewart, "*Concurrent Multipath Transfer Using SCTP Multihoming Over Independent End-to-End Paths*," IEEE/ACM Transactions on Networking, vol. 14, no. 5, October 2006, pp. 951-964, doi: 10.1109/TNET.2006.882843.

[26] R. M. Selvi, R. Rajaram, "*A Genetic Based Approach for Multiobjective Optimization of Disk Scheduling to Reduce Completion Time and Missed Task*," International Journal of Information Technology Convergence and Services, Vol. 1, no. 4, August 2011, pp. 69-79, doi: 10.5121/ijitcs.2011.1407.

[27] M. R. Bonyadi, H. Rahmani, and M. E. Moghaddam, "*A Genetic-Based Disk Scheduling Method to Decrease Makespan and Missed Tasks*," Information Systems, vol. 35, no. 7, November 2010, pp. 791-803, doi: 10.1016/j.is.2010.04.002.

[28] A. Thomasian and L. Chang, "*Some new disk scheduling policies and their performance*," in Proceedings of ACM SIGMETRICS international conference on Measurement and modeling of computer systems, vol. 30, no. 1, June 2002, pp. 266-267, doi: 10.1145/511334.511373.

[29] S. Vazhkudai, "*Enabling the Co-Allocation of Grid Data Transfers*," in Proceedings of Fourth International Workshop on Grid Computing, Phoenix, AZ, USA, November 17, 2003, pp. 44-51, doi: 10.1109/GRID.2003.1261697.

[22] Madhuparna, "*12 Best Free File Copy Software for Windows PC to Copy Faster*", The Geek Page, www.thegeekpage.com/12-best-free-file-copy-software-for-windows-10-to-copy-faster/ (July 20, 2021)

[30] Wikipedia, "*FastCopy*," www.en.wikipedia.org/wiki/FastCopy (Accessed July 20, 2021).

[31] FastCopy, "*FastCopy*," www.fastcopy.jp/ (Accessed July 20, 2021).

[32] Wikipedia, "*TeraCopy*," www.en.wikipedia.org/wiki/TeraCopy (Accessed July 20, 2021).

[33] Code Sector, "*TeraCopy for Windows/Mac*," www.codesector.com/teracopy (Accessed July 20, 2021).

[34] Softpedia, "*Unstoppable Copier*," www.softpedia.com/get/System/Back-Up-and-Recovery/Unstoppable-Copier.shtml (Accessed July 20, 2021).

[35] Roadkil, "*Roadkil's Unstoppable Copier*," www.roadkil.net/program.php?ProgramID=29 (Accessed July 20, 2021).