Networks Project 3 Experiment Report

A study of P2P transfer rates in relation to thread count and packet buffer size

Stephen Whitcomb and Danny Lee

CSCI 4760a Networks

Michael Cotterell

April 24, 2018

Introduction:

Peer to peer file transfer is an important topic in computer science for several reasons. Increasing the speed of file transfer will expedite any operation that relies on it and you would be hard-pressed to find a program that does not transfer any files. However, improving the speed of file transfer while maintaining file integrity can be a challenge. In this experiment we attempt to speed up file transfer by changing the number of threads the sender is using to send the file to the client as well as the buffer size of the packets we will send. Until now we have implemented our file transfer without threading, which does not take full advantage of our computational resources, since the sender is only sending one part of the file at one time. By adding additional threads to run parallel to each other and send portions of the packet at the same time we can complete file transfer in a fraction of the time one thread could send at. By decreasing the buffer size we can send packets more quickly as well. However, creating more threads to send with or decreasing the buffer size will not decrease transfer time exponentially as you might expect. There are other factors to consider: processing time, for example. Smaller packet sizes mean the sender and the receiver will both have to spend a greater amount of time fragmenting the file, prepending headers, and then sorting and writing them into a single file again. Though if the buffer size is too large you may waste time sending bulky packets and could overwhelm the cache of the receiver. Creating threads will help send more quickly but having too many will strain your computer's processer and may exceed its own capabilities, which will result in a slowdown. So there's a balance to be stuck between buffer size, number of threads, and your computer's processing capabilities. With this experiment we hope to find that balance with Nike's servers.

Methods:

We began conducting the experiment using big.txt and pushing it through various buffer sizes and thread counts. Both the sending client and the receiving client will be set to the same buffer size for all of our tests. Each of our tests is conducted twice and the resulting seconds are the averages of the two results. The receiving client and the server are hosted on a cluster while the sending client is hosted on Nike.

Each client was put through buffer sizes of 2048 bytes, 4096 bytes, and 8192 bytes. For each of these buffer sizes, we experimented with threads ranging from 1 to 10. The receiving client was restarted after every transfer to ensure that any cached files would be removed and that it would have a fresh start.

For the receiving client, arguments were conducted as:

```
$ python3 ftclient.py --receive --server vcf1:47682 -s <custom buffer size>
```

For the sending client, arguments were conducted as:

```
$ python3 ftclient.py --server vcf1:47682 -c <number of threads> --send <client ID> test/big.txt -s 90000
```

The buffer size was custom set to 90000 because the clients will communicate with each other and use the smaller of the two buffer sizes. Therefore, every single sending client would automatically be set to the receiver's buffer size.

Verification of the fie would be randomly taken throughout each transfer and a checksum is used to verify files are transferred properly.

These results can be reproduced using big.txt provided from sp2018's Github. The command line arguments stayed consistent across each test.

Result and Discussion:

The first thing one will notice when looking at our graphs is that the times when working locally, rather than on Nike are faster by an order of magnitude, this is to be expected as Nike is shared and constantly being used by many other students. We decided to do this to test the performance of our program without the interference of others on Nike and because we were curious to see what at what rates an unoccupied machine could transfer at.

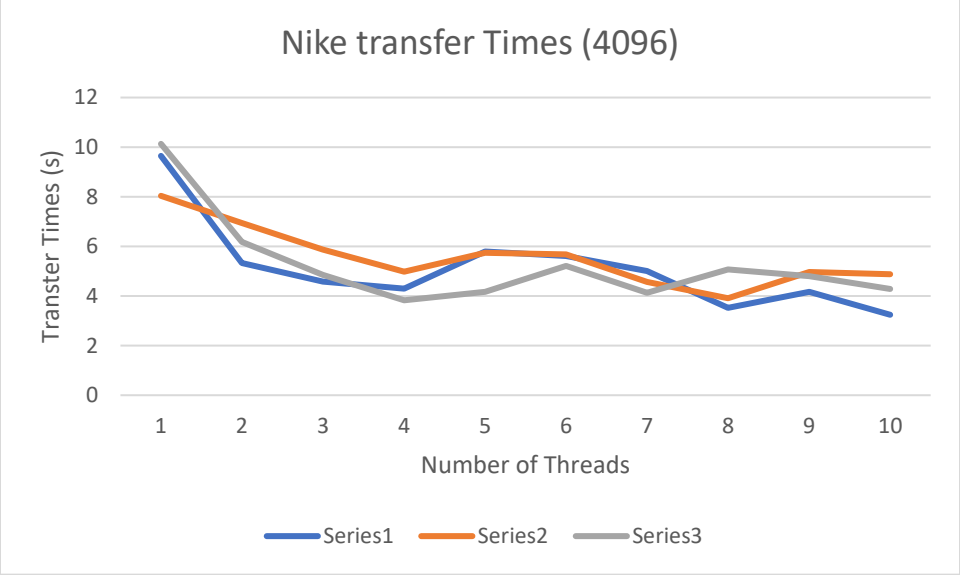
Ignoring this we can see from graphs I and II that the transfer rates decrease as the number of threads increase. However, the speed increase becomes more and more negligible as more threads are added. This is to be expected for two reasons. Firstly, under ideal conditions the time should decrease in an inverse exponential curve. Secondly, we expect less benefit from adding more threads as it adds strain on our processor while providing decreasing benefit.

After testing how the program performed with our default buffer size (4096 bytes) we tested it as well with differing buffer sizes. The two different buffer sizes we tested with were 2048 bytes and 8192 bytes. We initially planned to increment buffer sizes in multiples of two and gather more date, but we found that we were unable to successfully transfer a file with a buffer size much larger than 8192 bytes. As you can see form Graph III, decreasing the buffer size improved our data transfer speed slightly, while increasing it to 8192 bytes drastically improved transfer speed. However, the drawback is that if more than 8 threads were used to transfer the file, the transfer time spikes dramatically and can no longer be considered an efficient use of computing power.

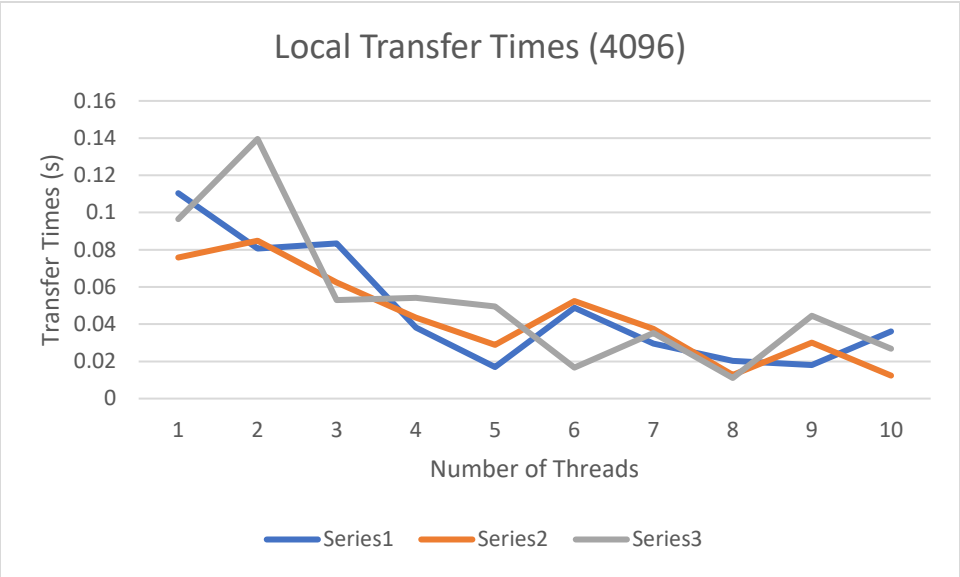
Graphs/ Charts:

(All the following data was gathered while transferring big.txt ~ 40 mb)

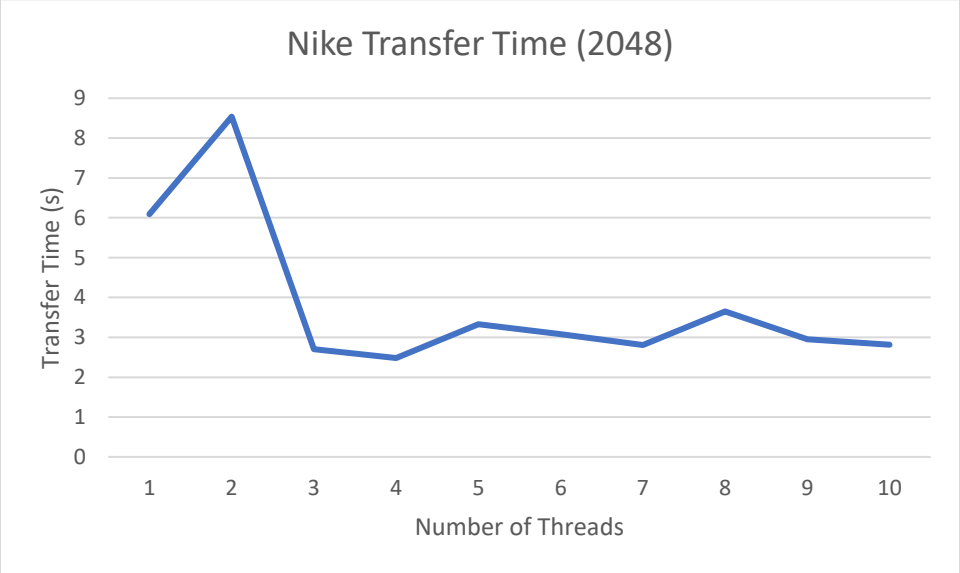
I)



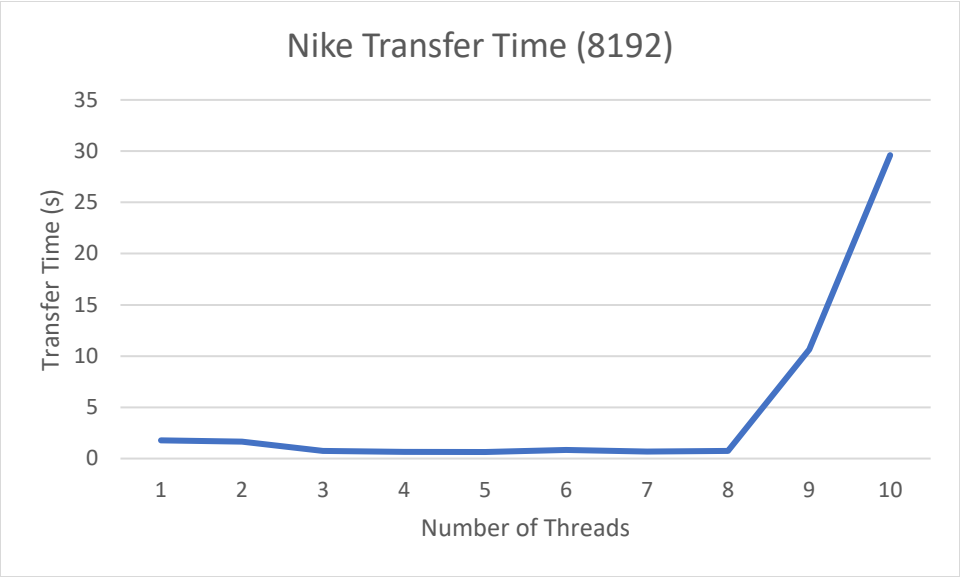
II)


Local Transfer Times (4096)
Transfer Times (s)
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
1
2
3
4
5
6
7
8
9
10
Number of Threads
Series1
Series2
Series3

III)


Nike Transfer Time (2048)
Transfer Time (s)
9
8
7
6
5
4
3
2
1
0
1
2
3
4
5
6
7
8
9
10
Number of Threads

IV)



Conclusion:

We concluded that the optimal number of threads for our program is about 9. Any more than that and the benefit of having the program run in parallel diminished past the point of actually being a benefit. The ideal buffer size for our program is a bit harder to nail down. We were unable to successfully transfer files with a buffer size of more that 8192, and we also saw that increasing the number of threads while attempting to transfer a large packet can have unexpected results. None the less, we can definitively conclude that while using 1-7 threads a large buffer size greatly improves transfer speed. Decreasing buffer size gives more consistent results, but with a far less significant increase in speed.