Determining the optimal protocols for transfer of large data files

Andrew van Rooyen
University of Cape Town
Andrew.vanRooyen@alumni.uct.ac.za

ABSTRACT

The transfer of large data files remains a necessity for scientific research, including Bioinformatics, despite alternate solutions like cloud computing. When transferring large files, there are many transfer protocols to choose from. In this paper, a lightweight testbed is developed and used to test the SCP, FTP, HPN-SCP and GridFTP protocols for transfer of large files between the Universities of Cape Town and the Western Cape. A comparison of GridFTP's 'UDT' mode (built on UDP) and its default mode is also done using the same testbed.

We investigate the effect of a congested network on transfers. Speed and data efficiency are compared on a stable network to determine if any one protocol is significantly better than the rest.

We find that for network speeds of around 200Mbit/s, the protocols achieve very similar speeds. The TCP based protocols achieved very similar efficiency, while GridFTP in UDT mode had much higher overhead.

1. INTRODUCTION

The size of data sets is increasing rapidly - this is the era of 'Big Data' [13]. One example is in Bioinformatics, where next generation sequencing has resulted in a massive increase in the size of raw data, which can be tens of gigabytes in size [7]. Genome sequencing technologies like SOLiD provide much higher data output at a cheaper cost [22], which creates challenges in data storage, transfer and access. In fact, the cost of storing a byte has been higher than sequencing a base pair since before 2010 [3].

Generally, sequence data is stored in a data warehouse. Storing this information for long periods of time requires the data to be structured efficiently in order to save space and to allow it to be transferred efficiently. There are a plethora of sequence file formats whose efficiency depends on the kind of data stored. Two of the most popular are FASTQ, which stores aggregated reads along with the quality of each DNA base pair [6], and BAM, the binary, compressed version of the Sequence Alignment Map (SAM) format [20].

As researchers require access to data warehouses to transfer the sequences they need for local analysis, these locations are often connected by massive data pipes like National Research and Education Networks (NREN's) [21] [23]. For example, South African universities are connected by the South African National Research Network which runs at 10Gbps. Unfortunately, standard transfer protocols like FTP [24] and SSH [24] were not designed for use on high-throughput networks, and alternate protocols must be used to avoid bottlenecks. However, it is not always clear which is the most suitable protocol for transfer of a particular file type between two end points.

Some proprietary transfer protocols are widely used in practice - for example, the *fasp* protocol by the US based company AsperaSoft. Based on UDP, this protocol eliminates the latency issues seen with TCP, and provides transfer bandwidth of up to 10 gigabits per second [8].

There have been some attempts to avoid data transfer altogether, which means processing data remotely. There has been an explorative push towards cloud solutions from companies such as Amazon and Google [3]. Unfortunately, even though cloud data centres have plenty of cheap storage, the transfer bottleneck remains as researchers must still upload their raw data to the cloud data centres every time they run a new experiment. Some researchers have even resorted to mailing hard drives [3]. There are also security, privacy and ethical concerns with outsourcing processing power to other companies, as sequenced DNA data is often highly sensitive information [14].

Although it would be ideal for researchers if this reliance on data transfer was removed, the current solutions do not provide good enough answers at every scale. Therefore, transferring big data is still necessary. In situations where data *must* be transferred, it stands to reason that the transfer should happen in the most efficient way.

In this work, we construct a testbed which can be deployed on a network to test various transfer protocols. The testbed is outlined, and then used to test SCP, FTP, GridFTP [2] and HPN-SCP [18].

In order to test the protocols in a relevant environment, tests are run between the University of Cape Town (UCT) and the University of the Western Cape (UWC) which are connected via SANReN.

2. BACKGROUND

The chosen protocols are the most popular protocols for file transfer in the field, perhaps with the exception of As-

This paper and associated code is available at https://github.com/wraithy/bigbinf
The OpenSSH-Portable installation used for HPN-SSH was compiled from https://github.com/rapier1/openssh-portable
The large hg38.fa.align.gz dataset used is available at http://hgdownload.cse.ucsc.edu/goldenPath/hg38/bigZips/
For more information, please see pubs.cs.uct.ac.za

peraSoft's *fasp*, which is non-free.

SCP (secure copy) and FTP (file transfer protocol) come with most Unix-based systems and the binaries (*scp* and *sftp*) are widely available as part of the OpenSSH [16] project. FTP is a generic protocol outline that may have many different implementations, but these have historically been insecure [4]. OpenSSH implements a *sftp-server* program which is built on top of its own ssh-transport protocol [24]. This allows for secure communications using current cryptographic techniques. This protocol makes use of a control channel (to send administrative messages between client and server) and a data channel (to actually transport the data). This approach of using multiple connections enables some additional features, for example, pausing and resuming transfers.

SCP is built directly on top of the SSH (secure shell) stack. In fact, when the *scp* binary is invoked, the *ssh* program is run with specific arguments. Because this abstraction removes the need to remember a list of command flags, SCP is arguably the most convenient transfer program to use. This simplicity also has drawbacks however, as SCP lacks the ability to pause and resume transfers.

One improvement that OpenSSH has made is in compression. By default, transfers via SCP and FTP will be compressed using methods from the *zlib* library [10], but the compression algorithm can be varied, as it is negotiated during the initial key exchange [15].

High Performance SSH (HPN-SSH) is a set of patches to OpenSSH which removes bottlenecks. It has not been implemented into OpenSSH for various reasons (there tends to be a certain amount of politics around large open source projects), but is still widely used in large data centers at big companies like Google, as well as NASA and the military [19].

GridFTP is an example of 'grid computing', which is defined by Foster and Kesselman as the emerging computational and networking infrastructure that is designed to provide pervasive, uniform and reliable access to data, computational, and human resources distributed over wide area environments [9]. GridFTP uses FTP as a base to provide a security infrastructure, resource management services, job management and information services [1]. It also provides features like data 'striping', which allows a transfer to be split across multiple hosts, potentially combining their bandwidths.

GridFTP also provides two modes of general operation. The default mode operates on top of the TCP stack, but there is a newer 'UDT' mode which is based on UDP. This mode aims to overcome congestion control bottlenecks found in TCP.

3. DESIGN AND IMPLEMENTATION

The simplest way to compare transfer protocols is to actually copy files. In this way, low-level technical details are ignored while practical data is captured. If more information is needed to explain any results, these details can be examined afterwards. With these requirements in mind, a testbed is created to measure speed, data efficiency and packet size of specific transfer protocols. Transfers are run and information about them is captured and saved. These logs are then analysed and displayed visually. The testbed does not output the 'best transfer protocol', but rather presents the information relevant to the specific environment so that the user can make an informed decision.

3.1 Testbed Design

The testbed relies heavily on the transfer protocol binaries themselves, and these in turn will vary. While we test a connection between two universities, another team might want to examine a message passing system within a localised computing cluster. Therefore, the testbed is designed to be simple, modular and extensible. A simple Python system is sufficient, because almost all of the work transferring and logging the data is done by external programs. The testbed simply acts as a mediator between them.

The protocols to be tested are specified in a config file, and have no limits imposed. As long as the correct binaries are installed on the machine, a user could test any transfer program as long as they know the command line arguments.

The testbed itself is designed to have minimal dependencies, while the optional Jupyter notebook [17] (an interactive python-based web page that can be used for analysis) makes heavy use of Python data science packages.

The data dump format is a simple json template, and can be parsed by the separate analysis code.

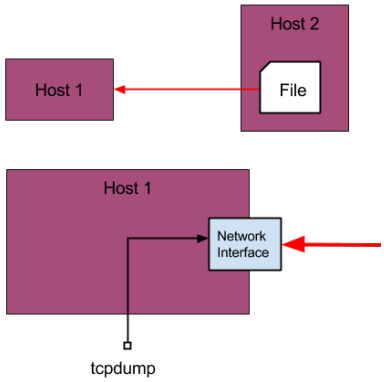


Figure 1: Host 1 copying a file and capturing packets.

The logging mechanism makes use of the 'tcpdump' program [12], which comes with most Unix-like systems. As depicted in Figure 1, it watches a network interface (e.g. eth0) and logs information about packets which pass through. The program is run while each transfer is in progress, and the output is filtered to include only packets sent between Host 1 and Host 2. This output is used as the raw data for analysis.

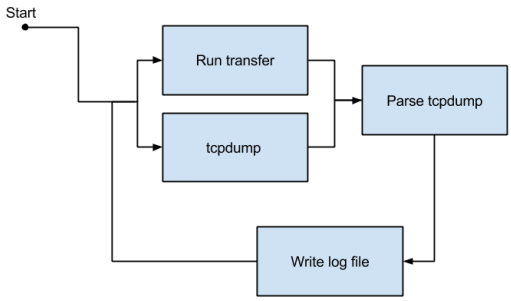


Figure 2: The sequence of actions performed by the testbed. This is repeated once for each protocol.

The Python program for running the file transfers accepts: the name of the network interface, the remote hostname

(Host 2), the path of the file on Host 2, and a local path to copy the file to. It then resolves the IPs of each host, and for each protocol, runs a transfer in isolation. As shown in Figure 2, it spawns a tcpdump subprocess which runs for precisely as long as the copy runs. The tcpdump program is started with filters, so that only traffic between the two hosts is captured. It then saves the output in a file. This allows for a controlled environment, because tcpdump only captures while the copy is running, no other packets are included in the logs. Also, the copies are run programmatically and consecutively. Successive copies are not started until both the tcpdump and protocol processes have been closed, and the log file has been written. This means that they are all run in an identical (within reason) environment, but at the same time do not interfere with each other.

This test process is run multiple times for statistical reasons, generating multiple log files.

Using the Jupyter notebook, this information can be aggregated and used to calculate metrics and display graphs. These include plots which are computed by looking at the time of each packet, and the size of its payload.

3.2 Evaluation

The testbed is simultaneously tested between various locations. During early stages, tests are run between lab computers at UCT. These are naturally on the same LAN. This means that there is no interference or issues with stability, but the network is capped at 100Mbit/s. Because all the protocols could easily reach this speed, the results from different tests don't reveal much.

The testbed is then deployed between a home network in Cape Town, and a Virtual Private Server in Amsterdam. The speed is again capped, but this enabled testing under unstable conditions, and the testbed had to be made more robust.

The testbed is finally deployed between two hosts to test that the binaries for each transfer protocol are set up correctly (both on the client and server). Once ready, a full set of tests was run between them.

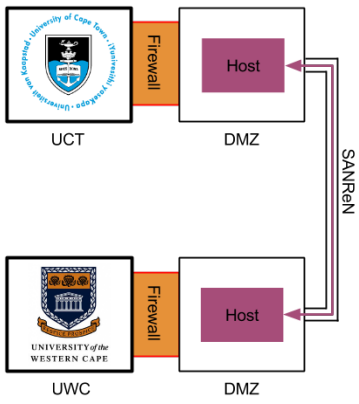


Figure 3: The hosts in their environment.

The hosts are virtual machines running at the South African National Bioinformatics Institute (UWC) and the Science DMZ (UCT). Both locations are close to the SANReN link, and outside institutional firewalls as depicted in Figure 3. This means that throttling is avoided, and ensures a speed of up to 1Gbps.

Four different protocols were chosen to be tested. Two of the most widely used are FTP and SCP, and each of these has a more recent successor (GridFTP and HPN-SCP respectively).

Transfers using each of the four protocols are run while the network traffic is logged.

The testing environment is kept as stable as possible during tests, and multiple tests are run at different times of the day.

For each transfer, a copy is initiated from Host 1. A file from Host 2 is transferred to Host 1 using the particular protocol (see Figure 1).

The tests are run with 6 files: one small 5MiB file and five files between 0.5GiB and 2.4GiB in 0.5GiB increments. These were all generated by reading chunks of a 2.4 GB gzipped sequence alignment file. Even though the protocols are agnostic to file format (they treat everything as binary), the sequence alignment file is a typical dataset that researchers transfer.

3.3 Protocols

The configuration of each transfer protocol is extremely important, as the performance can rely heavily on specific settings. There are also specific concerns for each protocol. For example, HPN-SSH isn't a standalone system, but rather a set of patches to OpenSSH. Therefore, the *scp* binary from a patched, portable version of OpenSSH is tested for HPN. This is installed alongside the original so that both binaries are available.

The 'lite' version of GridFTP is used. In this version, authentication is done via ssh as opposed a previously-configured certificate authority. This makes no difference to the file transfer itself, but it prevents unnecessary configuration of the testbed which can be quite complex in the case of 'full' GridFTP [11].

Although Bresnahan et al. found that GridFTP's UDT mode outperformed the TCP mode [5], both modes are still tested to see if their conclusions hold for this specific NREN as well. Note that despite its name, the tcpdump program will capture UDP packets.

We do not use the data striping features in GridFTP, because the number of available nodes can vary greatly between configurations, thus making comparisons with single-stream transfers unrealistic.

3.4 Data collection

Results were collected by running the testbed at 13h00 and 3h00 each day, for a period of two weeks. For some of the tests, all the data collected (including information about each packet) was stored to disk. Because this is very space intensive (a single test run could generate multiple gigabytes of dump files), most of the tests only stored aggregated data.

Algorithm 1 Script to run one round of tests.

```
SIZES = 5M 512M 1G 1_5G 2G 2_4G
for ROUND in {0..4} do
  for S in SIZES do
    Run transfer for S using testbed
  end for
end for
```

The metrics were chosen for practicality. For example,

behind-the-scenes information like window sizes was not collected, because the biggest concern for users is speed and size.

The tests were scheduled using the 'cron' daemon. Each job would call a script as in Algorithm 1. This provides plenty of data for various file sizes to be interpreted by the Jupyter notebook.

4. RESULTS

Even though the network can theoretically support up to 1Gbit/s, the speeds achieved throughout these tests were around 200Mbit/s. Although this is not as high as hoped, it is still a massive speedup over conventional South African networks, and these speeds are still typical to the use case in the field. Unfortunately, it is difficult to determine the exact location of this bottleneck because access was only available at the end points of the link, while the link itself is owned and managed by other third parties.

4.1 Network stability

Although each transfer in one test runs in the same environment, the time that tests are run can influence the results if the network is busier at certain times of the day.

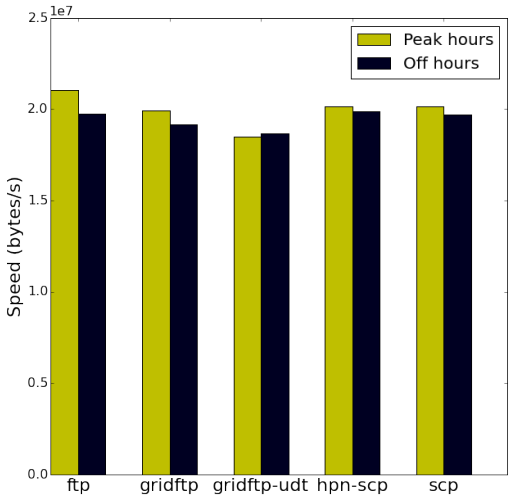


Figure 4: Aggregated comparison of speed of data transfer during peak and off hours, with five transfer protocols.

Protocol	p-value
ftp	0.4
scp	0.5
hpn-scp	0.7
gridftp	0.4
gridftp-udt	0.9

Table 1: p-values for a T-Test between the two sets, with null hypothesis that the speeds are the same.

Figure 4 shows a summary of the speed at various times in the day. 'Peak hours' were considered to be between 11h00 and 15h00, and all tests were either in this time frame (peak) or not (off hours). Interestingly, TCP based protocols tend to be slightly faster than GridFTP-UDT during peak hours, while the opposite is true during off hours. However, even though some might appear faster, a t-test between each set of subgroups reveals that the differences are not statistically significant. The p-values from this test are listed in Table 1, and are clearly quite large (meaning that we cannot reject the null hypothesis).

While the stability of the network at each time affects the speed of the transfers, it is sometimes useful to look other, more subtle effects as well. Figure 5 represents one complete transfer for each transfer protocol, for one file size (five rotations of Figure 2). This test is shown as a time series, where each point on the plot represents a packet. Both incoming and outgoing packets are plotted, and the height of each packet represents the size of its payload. In this case, there was a confirmed instability at the specific date and time that Figure 5b was captured. Note that transfers in Figure 5a take about 45 seconds, while in Figure 5b they take over 1.5 minutes.

Comparison of Figure 5a and Figure 5b confirms that transfers on a busy network do take longer. Secondly, the comparison shows how each transfer protocol handles the congestion. The fingerprints for the TCP based protocols are very similar, but close inspection shows that the unstable transfers are more clustered. The stable transfers are sparsely populated, but somewhat consistent, while the unstable transfers have distinct 'bands' with densely packed columns separated by thinner gaps with no packets. A likely explanation for this is that a router on one of the hops (most likely the first hop) was congested and had to buffer packets until space was made. This is a typical side effect of congestion on packet switching networks.

The UDP-based UDT mode of GridFTP changes in a more obvious way. Note that the axis limit doubles from the stable to unstable plot. This mode tries to match it's packet sizes to the network MTU (Maximum Transmission Unit) for efficiency. The MTU for the ethernet interface on the test machine was 1500, and Figure 5a is an example of the transfer operating in ideal conditions, consistently achieving this size. However, it is clear from Figure 5b that the first half of the transfer was congested, and a bigger payload is forced into each packet. This is only one of the factors that influence transfer speed, but it clearly does have a negative impact.

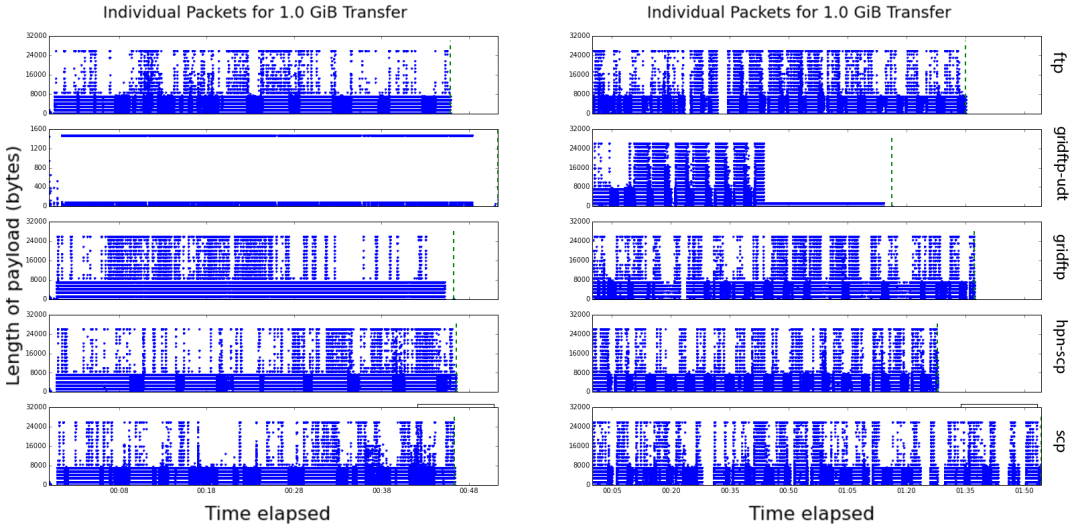
Because the differences in Figure 4 are not statistically significant, all tests are used for the speed and efficiency analysis.

4.2 Speed and Data Efficiency

Figure 6 is a simple plot of the average time taken to transfer the largest file. This by no means allows one to see the full picture, but it is useful if you want to find out which protocol will move your data in the shortest time. The SCP and HPN-SCP are the fastest, and both GridFTP modes finish roughly 5 seconds later. FTP is the slowest, taking 8.7 more seconds, which is 7.9% more time than SCP.

At this stage is is useful to note that, while GridFTP has improved upon FTP, the HPN patches to SCP have not sped up the transfer on this specific network configuration. Also, the improvement of GridFTP only holds true for this specific file size. A complete comparison for all file sizes appears in Figure 8a.

Figure 7 is a comparison between all five protocols measuring the total speed, and total bytes. This is aggregated



(a) Stable network

(b) Unstable network

Figure 5: A graphical fingerprint of each transfer run under two network conditions. Each point on the graph represents a packet.

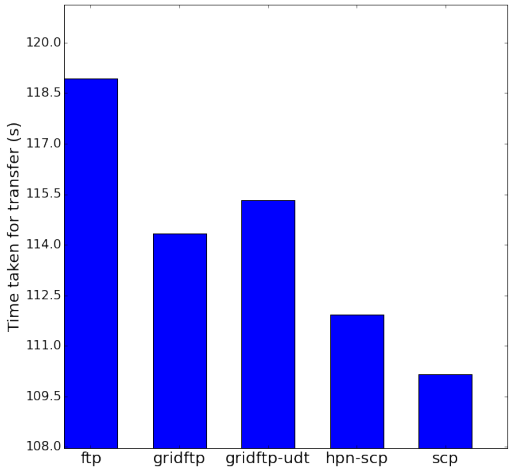


Figure 6: Total average time to transfer a 2.4GiB file. Smaller values are better.

across all tests for each protocol. Note that the speed calculations only include downstream data, while the 'transferred data' measures include data that is sent from the client back to the server via control channels etc. The dashed red line in each plot represents the size of the transferred file, and can be used for quick reference to see if a particular transfer protocol sends more or less data than the filesize on average. If the point lies to the right of the red line, the transfer has an overhead on average. If the point is on the left, the protocol has managed to send compressed data in an efficient way.

The corresponding numerical data (including upstream packets) can be seen in Table 2.

This same data can be represented as a function of filesize. In this case, Figure 8a compares the speed of each protocol for all tested file sizes. Surprisingly, the only protocols that are good consistently are SCP and HPN-SCP. GridFTP is consistently average, while FTP GridFTP-UDT change drastically with file size.

Note that the differences in speed are quite small at this scale (the graph is scaled to 10^6).

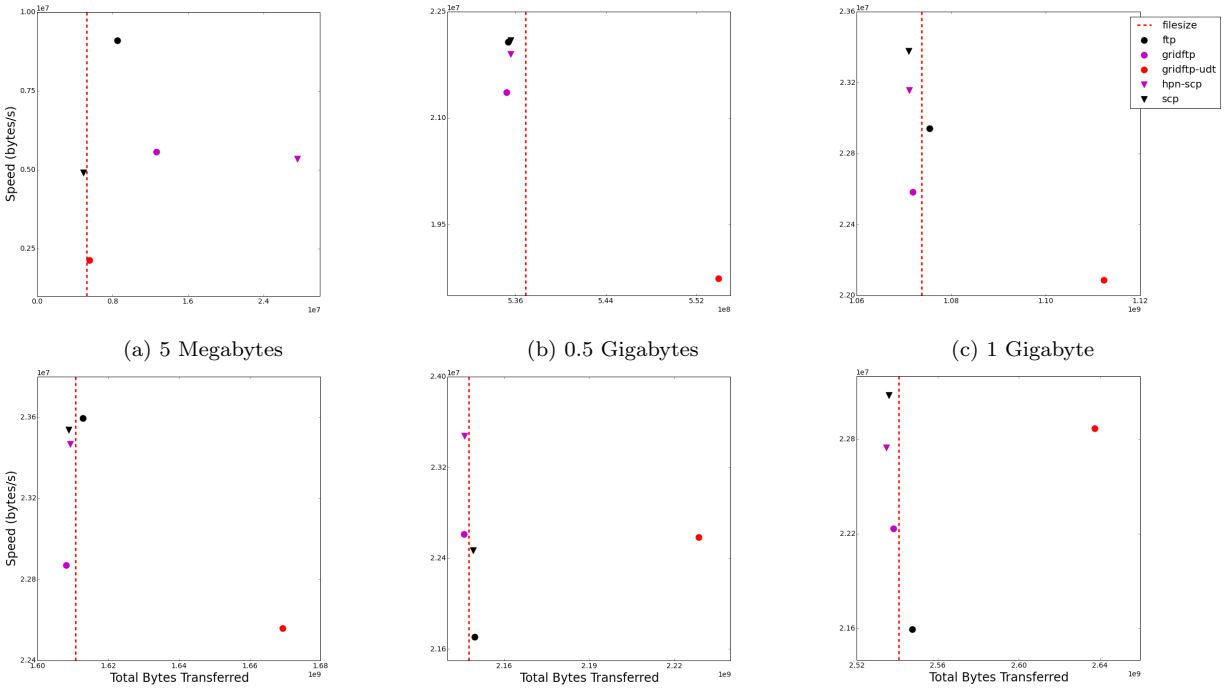
Figure 8b shows the data transferred as a ratio of the file's size. A ratio of 100% means that the total number of bytes transferred (both up and downstream) is exactly the file size. Ratios larger and smaller than this are more - and less data efficient, respectively. Here, one can see that GridFTP, HPN-SCP and SCP all achieve some level of optimization, as they are between 99.5% and 100%. FTP is close to, but just above this line between 100% and 100.5%. GridFTP-UDT is an outlier in this case, and clearly sends much more data than the TCP-based protocols.

One might expect that UDP transfers would use less data than TCP because UDP packets are not acknowledged by the host (requiring downstream data as well as upstream for each packet). This is not the case, because TCP acknowledgement packets are very small and tend not to make much of a difference. In fact, when the downstream data is removed completely, the resulting plots are almost identical to these that include the data. The overhead can instead be explained by UDP's lack of redundancy, and it is rare (but not impossible) for data to be sent more than once.

In a practical sense, all 5 protocols, save GridFTP-UDT have roughly the same efficiency, where both compression and overhead are less than 0.5%. GridFTP-UDT uses slightly more data (less than 5% overhead) which is not significant in most circumstances.

5. CONCLUSIONS

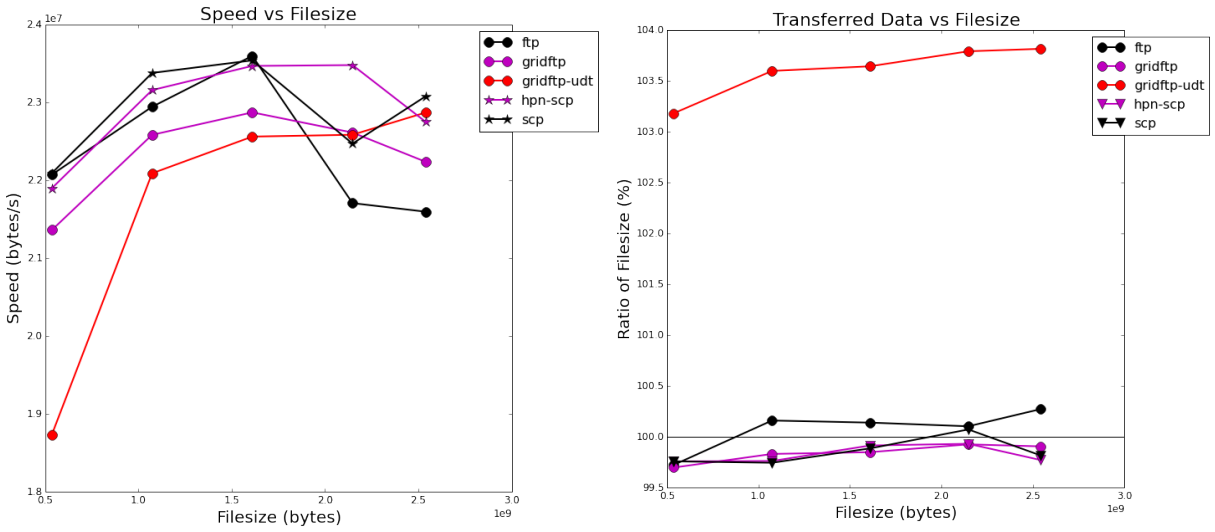
While transferring large datasets is not ideal, alternatives are not mature enough to remove the need completely. Cloud solutions are viable in some circumstances, but their shortcomings result in many situations where manually copy-



(a) 5 Megabytes (b) 0.5 Gigabytes (c) 1 Gigabyte

(d) 1.5 Gigabytes (e) 2 Gigabytes (f) 2.4 Gigabytes

Figure 7: Aggregated data per protocol. The dashed-red line represents the file size. Points to the left and right of the line are compressed/have overhead.



(a) Speed (b) Data Efficiency

Figure 8: Metrics calculated per file size

File Size (bytes)	Protocol	Bytes Down	Bytes Up	Bytes Total	Ratio (%)	Time (s)	Speed (bytes/s)
5242880	ftp	8413007.7	110014.3	8523022.0	162.6	1.4	9092596.7
	gridftp	7412012.0	5270788.4	12682800.4	241.9	1.3	5560789.2
	gridftp-udt	5543205.4	6239.1	5549444.5	105.8	2.7	2121498.5
	hpn-scp	5095229.0	22532226.6	27627455.6	527.0	1.0	5339580.5
	scp	4808391.8	104417.7	4912809.5	93.7	1.0	4907455.3
536870912	ftp	534084205.9	1298533.8	535382739.7	99.7	24.4	22069687.9
	gridftp	535240945.3	4425.0	535245370.3	99.7	25.3	21358608.5
	gridftp-udt	553768959.7	173186.2	553942145.9	103.2	30.2	18732210.9
	hpn-scp	535401726.9	164294.4	535566021.3	99.8	24.7	21895264.8
	scp	535381827.3	185875.7	535567702.9	99.8	24.4	22088599.5
1073741824	ftp	1072881895.1	2558993.1	1075440888.1	100.2	47.1	22940550.1
	gridftp	1071918333.8	4422.0	1071922755.8	99.8	47.6	22580395.8
	gridftp-udt	1112012162.9	340049.3	1112352212.1	103.6	50.4	22086429.9
	hpn-scp	1070826111.6	341411.1	1071167522.6	99.8	46.3	23154917.1
	scp	1070596084.3	382239.0	1070978323.3	99.7	46.1	23375141.7
1610612736	ftp	1609012156.9	3810770.9	1612822927.7	100.1	68.3	23594127.5
	gridftp	1608155406.3	4423.1	1608159829.5	99.8	70.5	22868925.7
	gridftp-udt	1668742869.2	507308.4	1669250177.6	103.6	74.0	22557996.6
	hpn-scp	1608701999.0	513687.6	1609215686.6	99.9	68.7	23465480.1
	scp	1608169009.5	581313.7	1608750323.2	99.9	68.4	23536710.8
2147479552	ftp	2144495341.4	5166344.3	2149661685.7	100.1	100.0	21704026.4
	gridftp	2145845215.4	4422.5	2145849637.9	99.9	95.0	22610027.5
	gridftp-udt	2228159986.6	676775.8	2228836762.4	103.8	98.7	22581830.1
	hpn-scp	2145264352.9	687628.8	2145951981.7	99.9	91.5	23476005.1
	scp	2148226021.1	753291.0	2148979312.1	100.1	96.2	22467830.4
2540610608	ftp	2541336324.7	6128837.9	2547465162.6	100.3	118.9	21593507.4
	gridftp	2538172141.5	4412.9	2538176554.4	99.9	114.3	22233098.8
	gridftp-udt	2636682223.1	799058.2	2637481281.3	103.8	115.3	22868390.6
	hpn-scp	2533961524.4	806759.9	2534768284.3	99.8	111.9	22745305.3
	scp	2535048746.9	908753.4	2535957500.3	99.8	110.2	23077735.8

Table 2: Numerical data aggregated data per protocol

ing large files is unavoidable.

In these cases, it is only logical to make sure that the transfer is done in the fastest and most efficient way possible. An easily-deployable testbed has been created to test a specific connection and provide data which can be used to determine the most optimized solution.

This testbed was deployed on a LAN network, a conventional home internet connection, and a high-speed research and education network. The results from the SANReN connection between the Universities of Cape Town and the Western Cape were compiled and analysed. The traditional FTP and SCP were tested, as well as their newer counterparts, GridFTP and HPN-SCP.

Tests were run during lunch hours, and early hours of the morning. The data dumps from these tests were analysed using the testbed's supplementary Jupyter notebook. It was found that there was no significant difference between the speeds during peak and off hours. However, there were interesting artefacts in the few cases where the network was congested.

It was found that the TCP-based protocols achieved very similar speeds and data efficiency. GridFTP's 'UDT' mode is built on UDP, and had a much higher overhead than the TCP protocols. However, it was faster than its default mode for larger file sizes on this network.

Finally, we found no significant improvements for the newer protocols. SCP and FTP both remained fast and efficient on the connection, where GridFTP and HPN-SCP were either mediocre, or changed with file size.

Of course, out of these four protocols, GridFTP remains the best choice for a fully configured data centre, as it can make use of striping and more complex authentication. However, if a researcher needs to copy a file occasionally, SCP is still a solid choice, with the option of gaining some efficiency by applying the HPN patches with little effort.

In the future, it will be worthwhile to run the same tests on a network achieving over 1Gbit/s to see if the same results are yielded. This can be easily done by deploying the testbed described in this paper.

6. ACKNOWLEDGEMENTS

This project would not have been possible without access to virtual machines at both UCT and UWC. Mr. Peter van Heusden (South African Bioinformatics Institute) was readily available during the project period, and kindly set up a VM in the SANBI DMZ. He also granted access to the Cisco switch which allowed rules for each protocol to be configured. Mr. Heine de Jager provided access to the UCT Science DMZ, and was very accommodating under strict security protocols. He was also extremely available, and helped fix

multiple issues with the GridFTP setup and firewall rules.

7. REFERENCES

[1] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, S. Tuecke, and I. Foster. Secure, efficient data transport and replica management for high-performance data-intensive computing. In *Mass Storage Systems and Technologies, 2001. MSS'01. Eighteenth IEEE Symposium*, pages 13–13. IEEE, 2001.

[2] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster. The Globus striped GridFTP framework and server. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 54. IEEE Computer Society, 2005.

[3] M. Baker. Next-generation sequencing: adjusting to data overload. *Nature Methods*, 7(7):495–499, 2010.

[4] D. Bonachea and S. McPeak. *SafeTP: Transparently securing FTP network services*. Computer Science Division, University of California, 2001.

[5] J. Bresnahan, M. Link, R. Kettimuthu, and I. Foster. UDT as an alternative transport protocol for GridFTP. In *International Workshop on Protocols for Future, Large-Scale and Diverse Network Transports (PFLDNeT)*, pages 21–22. Citeseer, 2009.

[6] P. J. Cock, C. J. Fields, N. Goto, M. L. Heuer, and P. M. Rice. The sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic Acids Research*, 38(6):1767–1771, 2010.

[7] S. Deorowicz and S. Grabowski. Compression of DNA sequence reads in FASTQ format. *Bioinformatics*, 27(6):860–862, 2011.

[8] X. Fan and M. Munson. Petabytes in motion: Ultra high speed transport of media files a theoretical study and its engineering practice of Aspera fasp^{TM} over 10gbps wans with leading storage systems. In *SMPTE Conferences*, volume 2010, pages 2–13. Society of Motion Picture and Television Engineers, 2010.

[9] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[10] J.-l. Gailly and M. Adler. Zlib compression library. 2004.

[11] Globus. GridFTP lite. http://toolkit.globus.org/toolkit/data/gridftp/quickstart.html, 2015. Accessed: 2015-09-06.

[12] V. Jacobson, C. Leres, and S. McCanne. Tcpdump. http://www.tcpdump.org/, 2015. Accessed: 2015-10-18.

[13] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers. Big data: The next frontier for innovation, competition, and productivity. 2011.

[14] V. Marx. Biology: The big challenges of big data. *Nature*, 498(7453):255–260, 2013.

[15] D. Miller. Security measures in OpenSSH, 2007.

[16] OpenBSD. OpenSSH. http://www.openssh.com/, 2015. Accessed: 2015-09-11.

[17] M. Ragan-Kelley, F. Perez, B. Granger, T. Kluyver, P. Ivanov, J. Frederic, and M. Bussonier. The Jupyter/IPython architecture: a unified view of computational research, from interactive exploration to communication and publication. *AGU Fall Meeting Abstracts*, page D7, Dec. 2014.

[18] C. Rapier and B. Bennett. High speed bulk data transfer using the ssh protocol. In *Proceedings of the 15th ACM Mardi Gras conference*, page 11. ACM, 2008.

[19] C. Rapier (http://stackoverflow.com/users/3191124/chris rapier). Why when i transfer a file through sftp, it takes longer than ftp? Stack Overflow. http://stackoverflow.com/questions/8849240/why-when-i-transfer-a-file-through-sftp-it-takes-longer-than-ftp (Accessed: 2015-10-28).

[20] SAMTools. Sequence alignment/map format specification. https://samtools.github.io/hts-specs/SAMv1.pdf, 2015. Accessed: 2015-04-27.

[21] SANReN. The South African National Research Network. http://www.sanren.ac.za/, 2015. Accessed: 2015-09-01.

[22] J. Shendure and H. Ji. Next-generation DNA sequencing. *Nature biotechnology*, 26(10):1135–1145, 2008.

[23] B. Van Rooyen. Unlocking the potential of research networking: ICT and HCD. *CSIR Science Scope*, 5(2):30–31, 2011.

[24] G. Venkatachalam. The OpenSSH protocol under the hood. *Linux J*, 156:6, 2007.