Access to Big Data in Bioinformatics

Andrew van Rooyen
University of Cape Town

ABSTRACT

TODO

1. INTRODUCTION

Next generation sequencing has resulted in a massive increase in the size of bioinformatics datasets, which can be tens of gigabytes in size [Deorowicz and Grabowski 2011]. Sequencing technologies like SOLiD provide much higher data output at a cheaper cost [Shendure and Ji 2008], 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 [Baker 2010].

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 [Cock et al. 2010], and BAM, the binary, compressed version of the Sequence Alignment Map (SAM) format [SAMTools 2015].

Researchers often require access to the data warehouses to transfer the sequences they need. Luckily, these locations are often connected by massive data pipes like National Research and Education Networks (NREN's). For example, South African universities are connected by the South African National Research Network [SANReN 2015] which runs at 10Gbps. Unfortunately, standard transfer protocols like FTP and SSH were not designed for use on high-throughput networks, and alternate protocols must be used to avoid bottlenecks.

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 [Beloslyudtsev 2014].

There have been some attempts to avoid data transfer altogether. This means processing data remotely, and there has been an explorative push towards cloud solutions from companies such as Amazon and Google [Baker 2010]. 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 [Baker 2010].

There are also security, privacy and ethical concerns with outsourcing processing power to other companies, as sequenced DNA data is often highly sensitive information [Marx 2013].

In this work, we compare the GridFTP, FTP, HPN-SSH and SSH file transfer protocols to determine which is best for transferring bioinformatics data on an educational network. These are the most popular protocols for file transfer in the field, perhaps with the exception of AsperaSoft's *fasp*, which is non-free.

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

2. METHODS

The transfer protocols are tested to establish measures of speed, data overhead and packet size. Transfers are run periodically at various hours of the day.

2.1 Protocols

The OpenSSH [OpenBSD 2015] implementations of FTP (sftp) and SSH (scp) are tested. High Performance SSH (HPN-SSH) is a set of patches to OpenSSH which removes bottlenecks [Rapier and Bennett 2008]. 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 [Allcock et al. 2005] is used. This means that 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 [Globus 2015].

Once the ideal protocol is decided on, it is made available as an endpoint to the microcloud system, so that users can retrieve their results in an optimal way.

2.2 Approach

This paper and associated code is available at https://github.com/wraithy/bigbinf
The OpenSSH-Portable installation used for HPN-SSH was compiled frm 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

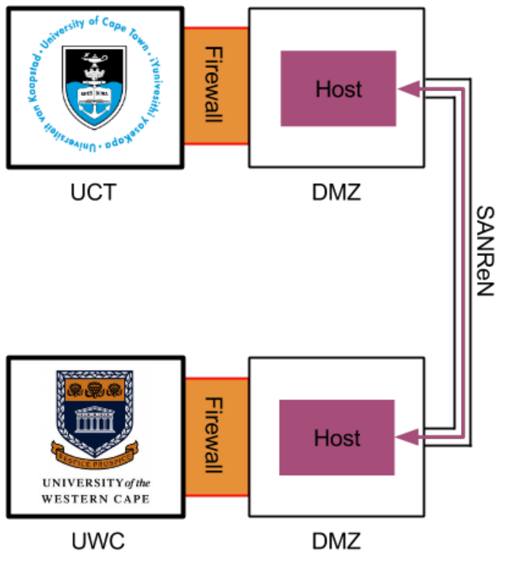


Figure 1: The hosts in their environment.

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

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 1. This means that throttling is avoided, and ensures a minimum speed of 1Gbps.

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 2).

The tests are run with 3 files: A 6 byte file containing the word 'hello', a 350MB video file and a 2.4 GB gzipped sequence alignment file. Even though the protocols are agnostic of file format (they treat everything as binary), the 2.4GB file is a typical dataset that researchers would need.

The 'tcpdump' program (www.tcpdump.org) comes with most Unix systems. As depicted in Figure 2, 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.

This approach allows analysis of transfer overhead (because *all* transferred data is logged, rather than just the file size), speed (data/time), consistency, total time, and total size.

Note that despite the name, tcpdump can also capture UDP packets, which is relevant if GridFTP is run in UDT mode [Gu and Grossman 2007]. This is an alternate protocol built on top of UDP which aims to overcome congestion control bottlenecks found in TCP. Although Bresnahan et. al. found that GridFTP's UDT mode outperformed the TCP mode [Bresnahan et al. 2009], both modes are still tested.

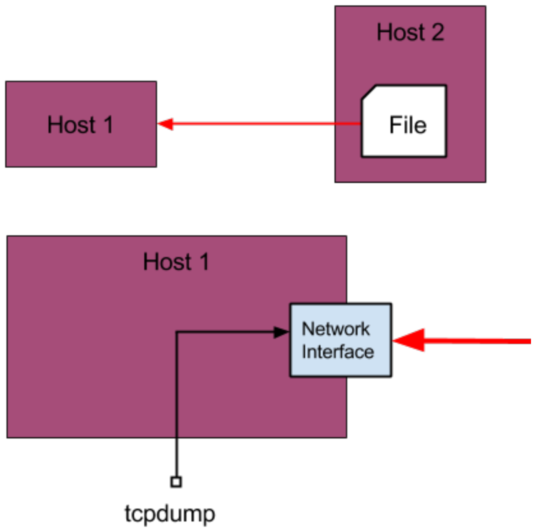


Figure 2: Host 1 copying a file and capturing packets

2.3 Implementation of Testbed

A simple Python script is sufficient to run the transfers, as most of the work is done by a subprocess for each specific protocol. The analysis is also done with Python, as there are a vast number of analytical and visualisation tools available.

All testing is done on an Ubuntu 14.04 system, with the respective transfer programs installed.

The Python script for running the file transfers has been written to accept: 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 3, 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 the 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.

A separate Python file then reads in the log files and parses them. This information is used to calculate metrics and display graphs. Operations can then be run by looking at the time of each packet, and the size of its payload.

This data can then be aggregated over multiple log files, and graphed using the matplotlib Python library.

3. RESULTS

Please note that the following data was not collected from the final tests. It represents a test run between my lab machine at UCT, and my DigitalOcean VM in a datacenter in Amsterdam. The environment was not controlled, and

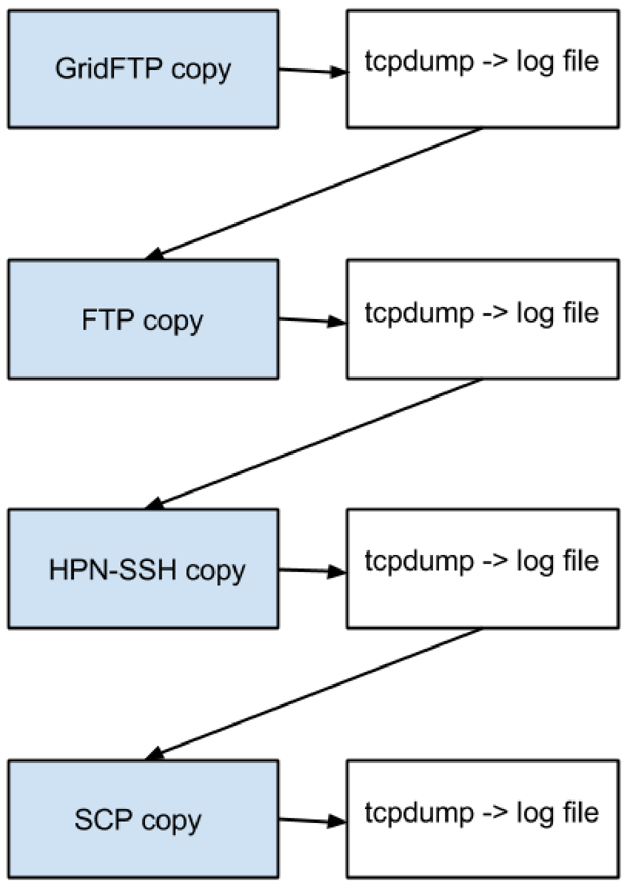


Figure 3: The sequence of subprocesses called by the testbed.

the connection was absolutely *not* stable. In fact, you are able to see this from the graph depicting individual packets. This section will be replaced with real data, and just serves to demonstrate the *kinds* of graphs that can be included.

Figure 4 shows each packet of the transfer as a dot, where the x axis is time and the y axis shows the payload length in bytes.

Figure 5 is what the analysis suite outputs if no arguments for graphs are specified. It is representing the same data, but shows it in a way which is easier to see actual values.

Figure 6 shows the same data as Figure 5, but displayed as a bar graph. This facilitates easier comparison between protocols, and makes it easy to see how much overhead (or compression) was used by comparing the bars to the filesize line.

4. ACKNOWLEDGEMENTS

TODO

5. REFERENCES

[Allcock et al. 2005] William Allcock, John Bresnahan, Rajkumar Kettimuthu, Michael Link, Catalin Dumitrescu, Ioan Raicu, and Ian Foster. 2005. The Globus striped GridFTP framework and server. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*. IEEE Computer Society, 54.

[Baker 2010] Monya Baker. 2010. Next-generation sequencing: adjusting to data overload. *nature methods* 7, 7 (2010), 495–499.

[Beloslyudtsev 2014] Dima Beloslyudtsev. 2014. Aspera transfer guide. (2014).

[Bresnahan et al. 2009] John Bresnahan, Michael Link, Rajkumar Kettimuthu, and Ian Foster. 2009. Udt as an alternative transport protocol for gridftp. In *International Workshop on Protocols for Future, Large-Scale and Diverse Network Transports (PFLDNeT)*. Citeseer, 21–22.

[Cock et al. 2010] Peter JA Cock, Christopher J Fields, Naohisa Goto, Michael L Heuer, and Peter M Rice. 2010. The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic acids research* 38, 6 (2010), 1767–1771.

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

[Globus 2015] Globus. 2015. The South African National Research Network. http://toolkit.globus.org/toolkit/data/gridftp/quickstart.html. (2015). Accessed: 2015-09-06.

[Gu and Grossman 2007] Yunhong Gu and Robert L Grossman. 2007. UDT: UDP-based data transfer for high-speed wide area networks. *Computer Networks* 51, 7 (2007), 1777–1799.

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

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

[Rapier and Bennett 2008] Chris Rapier and Benjamin Bennett. 2008. High speed bulk data transfer using the SSH protocol. In *Proceedings of the 15th ACM Mardi Gras conference: From lightweight mash-ups to lambda grids: Understanding the spectrum of distributed computing requirements, applications, tools, infrastructures, interoperability, and the incremental adoption of key capabilities*. ACM, 11.

[SAMTools 2015] SAMTools. 2015. Sequence Alignment/Map Format Specification. https://samtools.github.io/hts-specs/SAMv1.pdf. (2015). Accessed: 2015-04-27.

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

[Shendure and Ji 2008] Jay Shendure and Hanlee Ji. 2008. Next-generation DNA sequencing. *Nature biotechnology* 26, 10 (2008), 1135–1145.

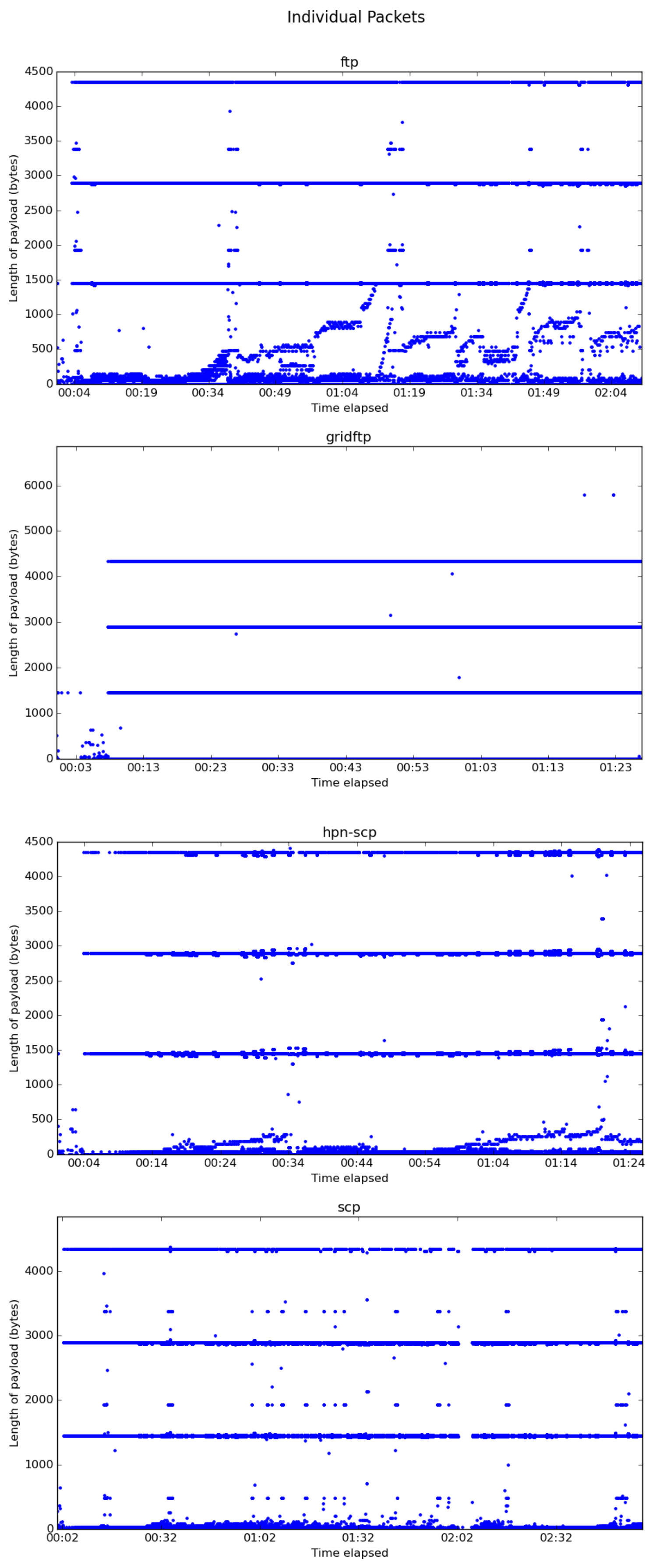
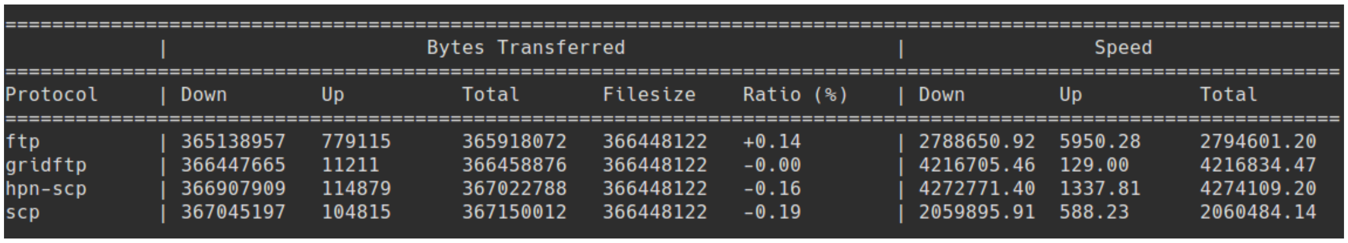


Figure 4: A graphical representation of each transfer



```
==========================================================================================================================
              |                    Bytes Transferred                            |                Speed
==========================================================================================================================
Protocol      | Down         Up           Total        Filesize     Ratio (%)   | Down         Up           Total
==========================================================================================================================
ftp           | 365138957    779115       365918072    366448122    +0.14       | 2788650.92   5950.28      2794601.20
gridftp       | 366447665    11211        366458876    366448122    -0.00       | 4216705.46   129.00       4216834.47
hpn-scp       | 366907909    114879       367022788    366448122    -0.16       | 4272771.40   1337.81      4274109.20
scp           | 367045197    104815       367150012    366448122    -0.19       | 2059895.91   588.23       2060484.14
```

Figure 5: The raw data, aggregated per protocol

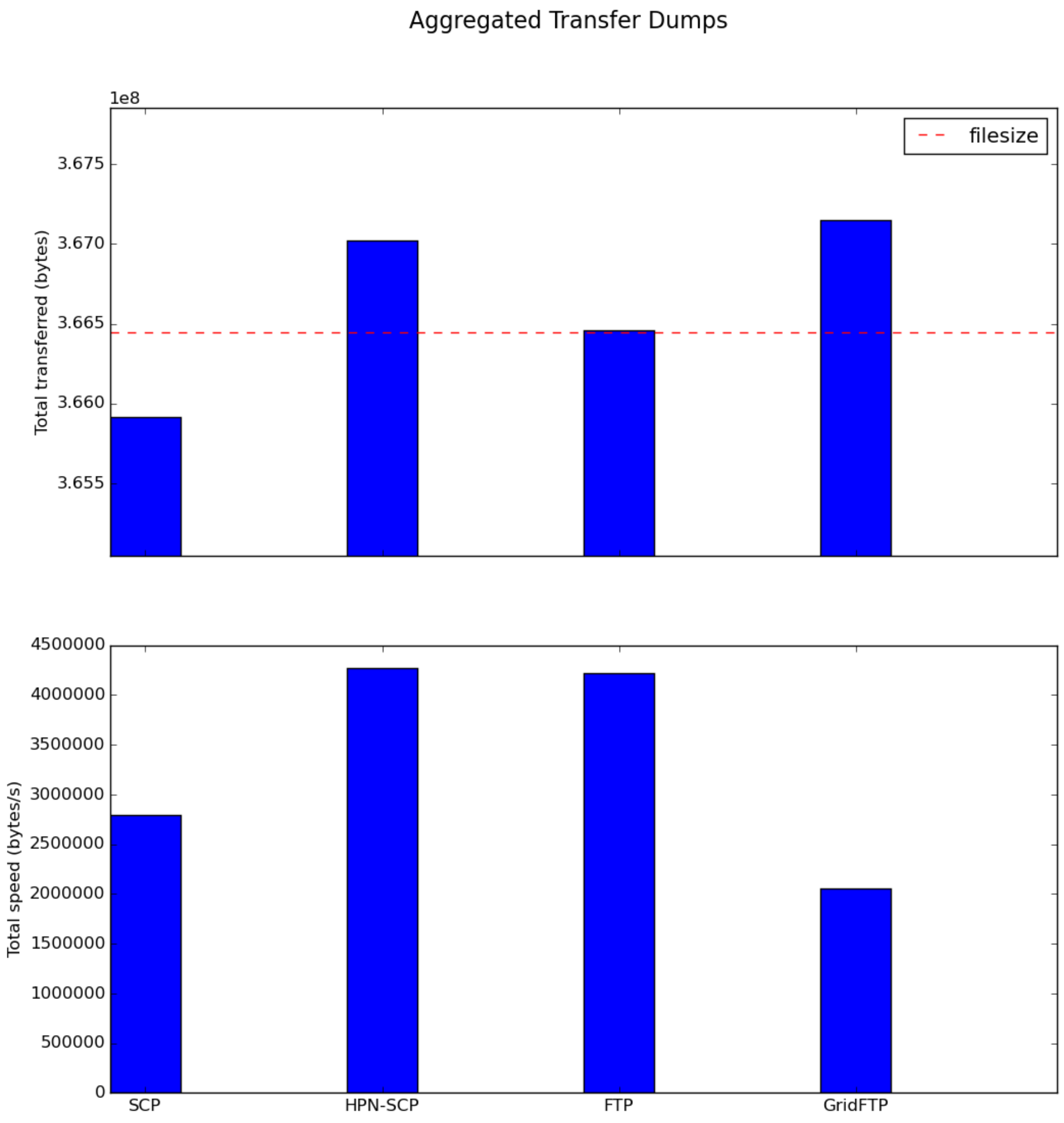


Figure 6: Aggregated data per protocol, with a line representing filesize