Comparative Analysis of Big Data Transfer Protocols in an International High-Speed Network

Se-young Yu, Nevil Brownlee, and Aniket Mahanti
University of Auckland, New Zealand
{se-young.yu,n.brownlee,a.mahanti}@auckland.ac.nz

***Abstract*—Large-scale scientific installations generate voluminous amounts of data (or big data) every day. These data often need to be transferred using high-speed links (typically with 10 Gb/s or more link capacity) to researchers located around the globe for storage and analysis. Efficiently transferring big data across countries or continents requires specialized big data transfer protocols. Several big data transfer protocols have been proposed in the literature, however, a comparative analysis of these protocols over a long distance international network is lacking in the literature. We present a comparative performance and fairness study of three open-source big data transfer protocols, namely, GridFTP, FDT, and UDT, using a 10 Gb/s high-speed link between New Zealand and Sweden. We find that there is limited performance difference between GridFTP and FDT. GridFTP is stable in terms of handling file system and TCP socket buffer. UDT has an implementation issue that limits its performance. FDT has issues with small buffer size limiting its performance, however, this problem is overcome by using multiple flows. Our work indicates that faster file systems and larger TCP socket buffers in both the operating system and application are useful in improving data transfer rates.**

I. Introduction

Large scientific installations such as the Large Hadron Collider (LHC) and the Square Kilometer Array (SKA) produce huge amounts of raw data every day. These data (often referred to as 'big data') are processed locally and distributed to researchers all over the world for data preservation, storage, and analysis. For example, the LHC transfers about 30 PB of data every day through the LHC Computing Grid due to storage limitations[1]. These research sites are often equipped with high-speed Internet connectivity (10 Gb/s or more) to facilitate the transfer of big data.

Standard data transfer protocols (e.g., TCP) do not adequately utilize the large transfer capacity of such networks. Traditional TCP's maximum achievable throughput is relatively low compared to the available capacity, even without any exogenous packet losses, since TCP takes substantial time after a packet loss to restore its congestion window size [1]. Another approach is to use UDP as a transport layer protocol, and develop congestion and reliability control algorithms at the application layer. Compared to TCP with its slow-start and congestion window algorithm, the UDP-based protocol's congestion control algorithm could be more flexible in terms of aggressiveness.

[1]http://wlcg.web.cern.ch/

These approaches can be further improved by using specialized big data transfer protocols such as FDT [2] and GridFTP [3]. They optimize transport protocol to accommodate large data size with parallel streams using multiple sockets. They strip data across sockets, and use multiple threads to transfer data between the local file systems and the network sockets. These enhancements improve the utilization of high-speed links while transferring large datasets.

In this paper, we present a comparative analysis of three big data transfer protocols, namely, *GridFTP*, *FDT* and *UDT*. We measured their performance and fairness in a long distance intercontinental 10 Gb/s link between Australasia and Europe. From our analysis we find that GridFTP showed the best performance because of its implementation stability and ability to set large TCP socket buffer size. FDT was not able to finish data transfer with large TCP socket buffer, however, this problem is overcome by using multiple flows. We also find that to fully utilize a long distance link, it is essential to keep the congestion window size as large as possible. Faster file systems and larger TCP socket buffers in both the operating system and application are useful in improving performance.

To the best of our knowledge, this is the first comprehensive analysis of three major open-source big data transfer protocols in an international link. Our trace-driven analysis provides detailed insights into the functioning of these protocols. These insights can be used by researchers and developers to tune or improve performance and fairness of the protocols. Our results can be used by scientists to choose the most suitable transfer protocol based on their logistical needs and network connectivity.

The rest of the paper is organized as follows. Section II provides an overview of the big data transfer protocols and TCP congestion algorithms. Section III discusses our experimental setup and trace collection methodology. Comparative analysis of the big data transfer protocols based on transfer of single flow and multiple flows are presented in Section IV and Section V, respectively. Section VI discusses implications of our results. Section VII discusses related work. Section VIII concludes the paper.

II. Background

There are two major categories of big data transfer protocols: *TCP-based* and *UDP-based*. TCP-based protocols [2], [3] depend on the existing TCP congestion avoidance algorithm,

978-1-4673-8590-9/15/$31.00 ©2015 IEEE

TABLE I
OVERVIEW OF BIG DATA TRANSFER PROTOCOLS

Protocol	Version used	Transport protocol	Congestion control	Multiple transfer flows
GridFTP	6.0	TCP	TCP CUBIC	Supported
FDT	0.19.2	TCP	TCP CUBIC	Supported
UDT	4.7	UDP	D-AIMD	Not Supported

and focuses on optimizing it. UDP-based protocols [4], [5] use their own congestion control algorithm.

Table I summarizes the key features of the three protocols evaluated in this paper. *GridFTP* [3] is a free software implementation, which provides extensions to FTP. It provides enhancements such as parallel data transfer, data striping, and TCP socket buffer optimization. *FDT* [2] is a platform-independent software implementation that uses TCP to provide parallel data transfer, socket buffer optimization, and multiple I/O threads. *UDT* [4] is an UDP-based connection-oriented data transfer protocol with its own congestion control algorithm, called Decreasing Increases AIMD (DAIMD).

We used *CUBIC TCP* [6] for GridFTP and FDT. CUBIC TCP uses a cubic function for increasing its congestion window instead of a square function. CUBIC's congestion window size t seconds after a congestion event is $W = C(t-K)^3 + W_{max}$, where $K = \sqrt{\frac{W_{max}*\beta}{C}}$ and β is its 'decreasing factor.' CUBIC's increasing function is not dependent on Round Trip Time (RTT) to have better fairness among flows. The W is calculated and applied when current window size is smaller than $W_{max}(1-\beta) + 3\frac{\beta}{2-\beta}\frac{t}{RTT}$. CUBIC has been further optimized with *Hystart*, replacing the traditional slow-start with a new algorithm to reduce slow-start overhead [7].

UDT [4] aims to provide TCP-friendly congestion control on top of UDP, using decreasing AIMD and rate control to achieve high throughput over long fat networks. Its sending rate increasing parameter is related to the current sending rate. As the current sending rate nears the maximum available capacity (measured by packet pair estimation), the sending rate increases more slowly to probe the optimal throughput. UDT decreases its sending rate by $1/8$ when consecutive packets are lost, so as to tolerate exogenous packet losses.

III. METHODOLOGY

Our experimental setup consisted of two machines located in Queenstown, New Zealand and Stockholm, Sweden. Figure 1 shows the network path between Queenstown and Stockholm. `Traceroute` revealed that there were 13 hops in the network path, and the average RTT for the path was 323 *ms* during the experiment, without any experimental traffic injected.

The Queenstown side of network was operated by REANNZ[2], and NORDUnet[3] operated the Stockholm side. There are Internet2[4] and GEANT[5] operated links in between them.

[2]http://www.reannz.co.nz/
[3]https://www.nordu.net/
[4]http://www.internet2.edu/
[5]http://www.geant.net/

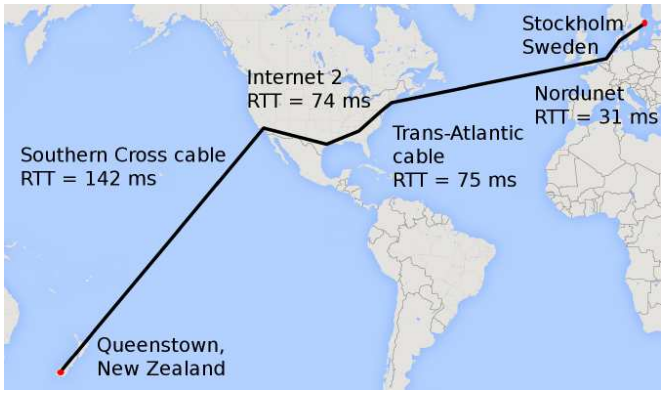

Fig. 1. Path between the experiment machines

The transmission capacity of the link between the two machines was at least 10 Gb/s. We conducted our data transfers during off-peak hours, so as to minimize the impact of our experiments on other network users. We also did not notice any major route changes during the experiments.

There are many scientists who need to send large volumes of scientific data (e.g., astronomical or biological data) to colleagues in Europe from Australasia. The large RTT is thus not extreme (and rather common) when seen from the perspective of residents of Australasia. In addition, the RTT may be longer if any part of the link involves a satellite link, which some current research sites may have.

The two machines were set up with 10 Gb/s Ethernet interfaces and the Linux operating system. Table II shows the details of sender and receiver used in the testbed. We installed GridFTP, FDT, and UDT on these machines. We chose GridFTP, UDT, and FDT because they are well-known high-speed data transfer protocols using TCP or UDP.

We surveyed the open-source protocols that large-scale distributed research projects are using for high-speed transfers, and chose the above-mentioned protocols to provide a meaningful contribution to the high performance computing community. We do not consider SCP and FTP because they do not focus on high-speed big data transfers (link speed $\geq$ 10 Gb/s). Other high-speed transfer protocols, such as Tsunami and HPN-SSH, are not considered because of their poor performance and stability problems during prior experiments conducted in our local 10 Gb/s testbed [8].

We performed two types of data transfers between the machines using GridFTP, FDT, and UDT. For *disk-to-disk transfers*, the sender transferred a zero-filled synthetic 30 GB data file ten times. The receiver wrote the transferred data to its file system. For *memory-to-memory transfers*, the sender sent streamed data from `/dev/zero` (to avoid disk read overhead) ten times to the receiver's `/dev/null` (to avoid disk write overhead). When analyzing performance of multiple flows, we used one through eight transfer flows for GridFTP and FDT.

In our study, we define *big data* as data large enough that transferring it over a network keeps link utilization high for significant amount of time so as to affect other network flows. The actual volume of data differs based on RTT and the link

TABLE II
HARDWARE USED IN THE EXPERIMENTS

Location	Model	CPU	Memory	NIC	OS
Queenstown, New Zealand	Gigabytes X79-UP4	Intel Core i7 4820K 3.70GHz	32 GB	Intel 82599EB 10-Gigabit SFI/SFP+	Ubuntu 12.04
Stockholm, Sweden	Dell PowerEdge R310	Intel Celeron G1101 2.27GHz	2 GB	Intel 82599EB 10-Gigabit SFI/SFP+	CentOS 5.10

TABLE III
GOODPUT MEASURED FOR PROTOCOLS USING A SINGLE FLOW

Protocol	Transfer method	Goodput (Gb/s)
GridFTP	Disk-to-Disk	0.49
	Memory-to-Memory	3.13
FDT	Disk-to-Disk	0.37
	Memory-to-Memory	1.00
UDT	Disk-to-Disk	0.20
	Memory-to-Memory	0.20

capacity. In our case, we use a 10 Gb/s network with 320 *ms* RTT. A data file larger than 8 TB would take more than 100 seconds to transfer, which is equivalent to 312.5 *ms* RTT and we regard this as a big data transfer.

The Queenstown machine used an `EXT4` filesystem, and the Stockholm machine used an `EXT3` filesystem to perform file reads and writes. We chose the *Queenstown machine as our receiver* because it had a faster file system for writing files to disk. The *Stockholm machine was the sender*. Host TCP tuning for each machine was done as described in ESnet Linux Tuning Guide[6]. We used CUBIC TCP for our congestion control algorithm. We did not increase the TCP sender's buffer size for FDT because it was implemented in Java. We found that changing the Java Virtual Machine's maximum memory size, and that of FDT's TCP socket buffer made FDT unstable. For GridFTP, we set our TCP sender's buffer size to an arbitrary 180 MB.

For all our experimental file transfers, we recorded packet traces using `tcpdump`. We developed custom analysis programs to investigate performance and fairness related to the three big data protocols on the international network. GridFTP and FDT optimize their high-speed transfers using parameter tuning and other features such as multiple I/O threads and data pipelining. However, we found that there was no significant improvement in either goodput and throughput using such other features, while tuning TCP provided significant improvement. Therefore, we further analyzed each protocols on how they tune their TCP.

To measure fairness of the big data transfer protocols, RTT changes in the link during the data transfer can be observed. Through the observation, we can compare potential impact on the network while using the protocols and how much it will cost to transfer data using the protocols in terms of RTT increases experienced by other flows in the same network.

IV. SINGLE DATA FLOW ANALYSIS

We performed several intercontinental experiments between the sender and receiver hosts. We first report on results when transferring data through a single flow using the three big data transfer protocols.

A. Performance

We measured the single-flow performance of each protocol. GridFTP was most efficient at reading and writing via the `EXT4` file system, compared to the other protocols. Although these protocols have different measured goodput values, their transfer rates are all less than 0.5 Gb/s.

After removing the file system I/O performance bottleneck, GridFTP goodput improves 600% and FDT goodput improves 270%. UDT shows no significant difference when the file system I/O limitation is removed. We found that UDT reported many lost packets within a few seconds after the transfer started. The packet loss is observed regardless of the amount of available bandwidth and causes UDT's congestion control algorithm to malfunction and to dampen its packet sending rate.

Compared to our previous study in a smaller in-lab testbed [8], all the protocols achieved significantly lower throughput. This decreased performance was due to large RTT and small TCP socket buffer sizes. Compared to our national testbed [9] (RTT was 10 *ms*), the international testbed has an RTT 30 times longer. This large RTT causes CUBIC TCP to take longer to increase its congestion window size, as well as for UDT to increase its sending rate after packet losses. The TCP send buffer size also limited the performance of each protocol.

We also measured how much data is in transit across the network for both GridFTP and FDT. UDT was not considered in this analysis because of its poor performance. Figure 2 shows the amount of data in transit ('on the wire') during the transfer. Even though GridFTP and FDT were using a single TCP socket to deliver data via the transport layer, their behaviours differ. FDT is not able to carry more than 50 MB of unacknowledged data at any time, while GridFTP is able to carry more than 170 MB.

Note that we had to use a smaller sender TCP socket buffer size for FDT compared to GridFTP because of the FDT performance problem. We tried to allocate the same buffer size as GridFTP, but found that FDT would not work properly with a large buffer size. FDT was not able to write the transferred data to the output socket. We also tried to set the TCP socket

[6]http://fasterdata.es.net/host-tuning/linux/

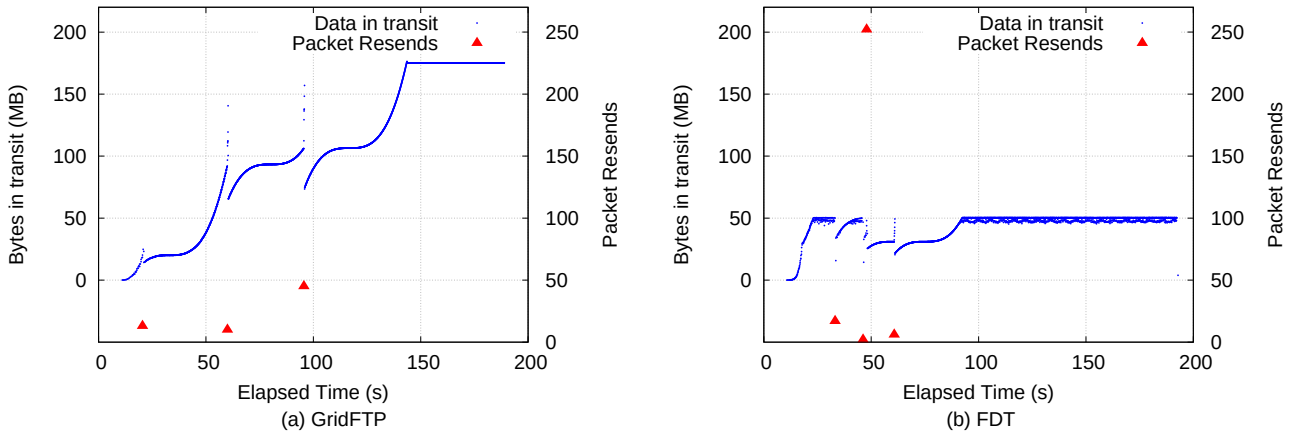

Fig. 2. Amount of data in transit (left axis) and packet resends (right axis) for GridFTP and FDT

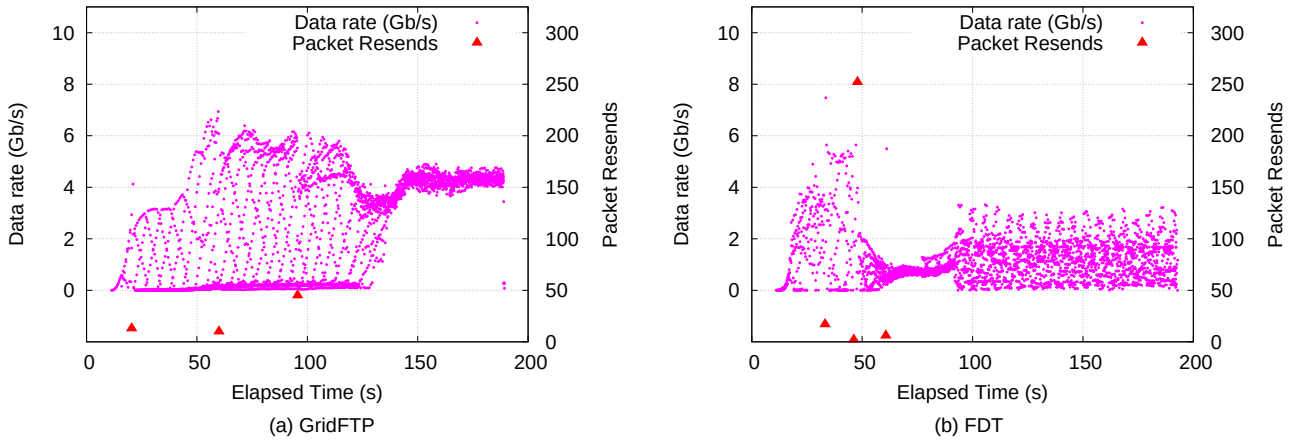

Fig. 3. Data rate (left axis) and packet resends (right axis) for GridFTP and FDT

buffer size in FDT, but found that FDT was not able to flush some streams well and paused for long periods.

Figure 3 shows the average data rate of flows per 5 *ms* interval. We observe that GridFTP's TCP send buffer size is high enough to settle at a rate of about 4.2 Gb/s. FDT's smaller buffer size causes the data rate to vary around lower rates. These fluctuations in rates most likely arise from the TCP socket's waiting time. FDT's 50 MB TCP socket buffer causes it to wait on further acknowledgement packets until its unacknowledged data is reduced to less than 50 MB. Performance of FDT is degraded compared to that of GridFTP because of the TCP socket wait time. Note that it does not necessarily mean that FDT is less efficient than GridFTP. This is because we were not able to make a fair comparison pertaining to implementation problem (as described above).

B. Impact on RTT

In our experiments, we transferred large data files through the globally-routed path between Queenstown and Stockholm. We were concerned about the *fairness* of such transfers, and decided to use increases in the path's RTT as an indication of any path congestion that they caused.

Figure 4 shows a microscopic view of RTTs observed during the flow, measured from arrival times of packet pairs, i.e., data packets and their corresponding acknowledgement packets.

There are several peaks shown in RTT during the flow, especially when there are packet losses for FDT. However, the relationship between packet resends and RTT peaks is weak. The interval between packet loss events are closer for FDT and rapid increase in congestion window appears several times between 20 *s* and 70 *s*, with multiple packet losses. Bursts of packets sent within a short time frame from FDT causes peaks in RTT, while the bursts are more spread out for GridFTP, and has less impact on the RTT increases.

For FDT, there is a series of packet losses between 40 *s* and 60 *s*, which causes significant RTT increase in the network path although overall throughput is low. This is because there are bursts of packets injected within a short time intervals (between 40 *s* and 60 *s*) as visible in Figure 2. These bursts may have caused the queues in routers to build up quickly. As a result, the RTT increased during the period, and remained high for another 20 *s*, then settling back to around 322 *ms*. There are a few more spikes at 80 *s* that may have been caused

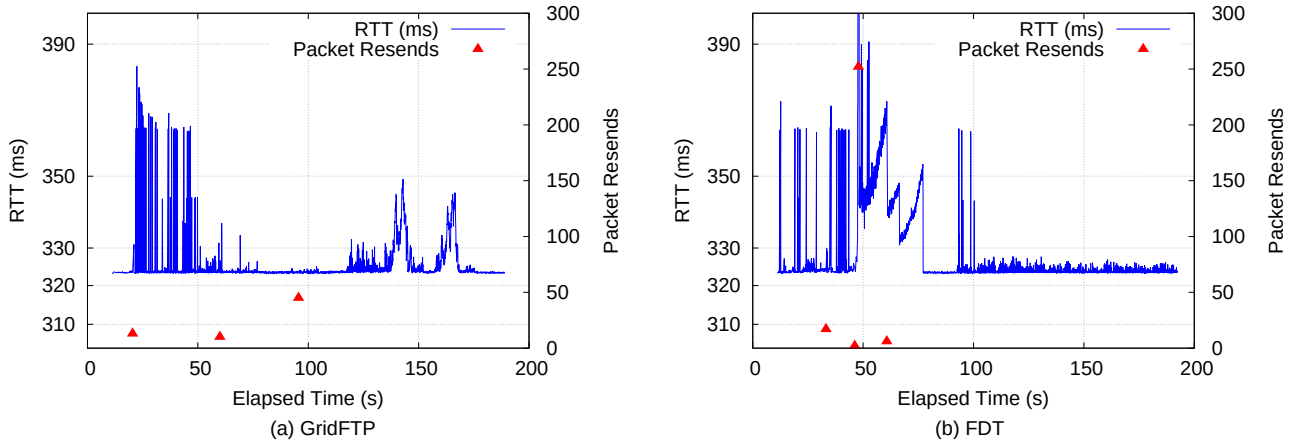

Fig. 4. RTT changes (left axis in log scale) and packet resends (right axis) for GridFTP and FDT

by the rapid increase in data in the network path.

We observe that RTT during GridFTP transfer is relatively stable, because the bursts are more distributed. Packets are lost at 20 *s*, 60 *s*, and 95 *s*, but there are no series of packets lost throughout the transfer. This results in less impact on the network path, and leads to more stable change in RTT. From 140 *s* onwards, there are some peaks. This may have been caused by the large amount of traffic injected as congestion window size increased, building up routing queues in the path.

These bursts of traffic are part of TCP CUBIC's effort to find optimal throughput over the long distance network, however, the different size of TCP's socket buffer caused the performance difference for the two applications. The characteristics of each flow depend on many factors, such as other TCP socket parameters and exogenous packet loss.

V. Multiple Data Flow Analysis

We measured the performance and fairness of multiple parallel data flows of GridFTP and FDT. For each test, multiple flows started and terminated at the same time using *memory-to-memory transfer*. We do not analyze UDT because it does not support multiple data transfer flows.

A. Performance

We found using multiple flows increases goodput for both GridFTP and FDT. The goodput increases from 3.2 Gb/s to 5 Gb/s when 5 flows are used for GridFTP. For FDT, the goodput increases from 1 Gb/s to 5.4 Gb/s when 6 flows are used. More than 6 flows did not increase the goodput for both protocols. Compared to our national testbed study [9], there is a significant increase in transfer speeds as the number of flow increases for both GridFTP and FDT.

The goodput for both protocols increases with increasing number of parallel data flows until it reaches a maximum (5 Gb/s for GridFTP and 5.5 Gb/s for FDT), where it remains stable. The goodput of FDT increases more rapidly compared to that of GridFTP, and the effect of using multiple flows appears higher.

When multiple flows increase their congestion window size at the same time at similar rates, the aggregate congestion window increase rate is much higher than that of a single flow. Even though there is a higher chance of losing packets for the aggregated flow, the recovery time to reach the maximum congestion window size is reduced. Along with TCP CUBIC, which steadily increases congestion window size when it is near a previously determined maximum, the shorter recovery time allows both protocols to increase their goodput with multiple flows in the long-haul network.

This behaviour was also observed in our national network study [9], which had shorter RTT, however, the recovery time was shorter compared to that of the longer international path. The benefit of having shorter recovery time by using multiple flows diminishes with small RTTs, because the recovery time for a single flow is almost as small as the recovery time for the multiple flows.

Another benefit of using multiple flows is to have more aggregated socket buffer resource. With a limited buffer size per socket, using multiple sockets increases the available buffer size, however, having a TCP socket with large socket buffer size would be as efficient.

Figure 5 shows the amount of data injected in transit from each of the five parallel flows for GridFTP and FDT. Most flows in both GridFTP and FDT are synchronized, thus loosing packets at the same times. This is expected behaviour because an exogenous packet loss is caused when too many packets are sent within a short time interval, overloading packet queues at one or more hops along the transfer path. Such dropped packets are likely to be distributed among the flows because each flow will increase its sending rate until it looses a packet. When the network path is overloaded, flows that have not yet lost a packet will keep sending bursts of packets while other flows are on hold (eventually these flows will also notice packet losses).

Recall that in Figure 2 we observed TCP's buffer limit of 50 MB was the major reason for FDT's lower performance. Using multiple flows (each with a 50 MB buffer) allows FDT

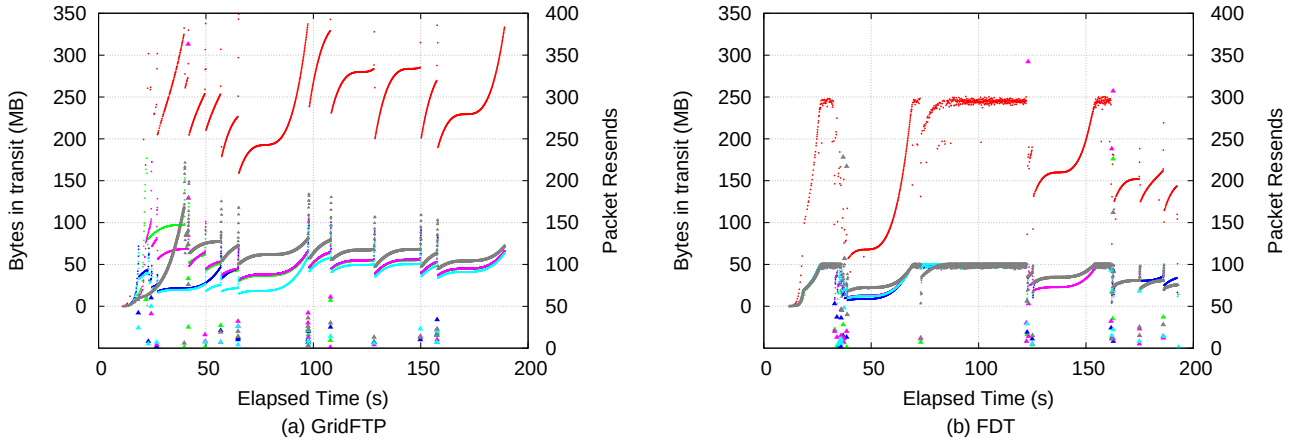

Fig. 5. Amount of data in transit (left axis) and packet resends (right axis) with multiple flows for GridFTP and FDT. Each flow has a unique color. Red represents the aggregated flow.

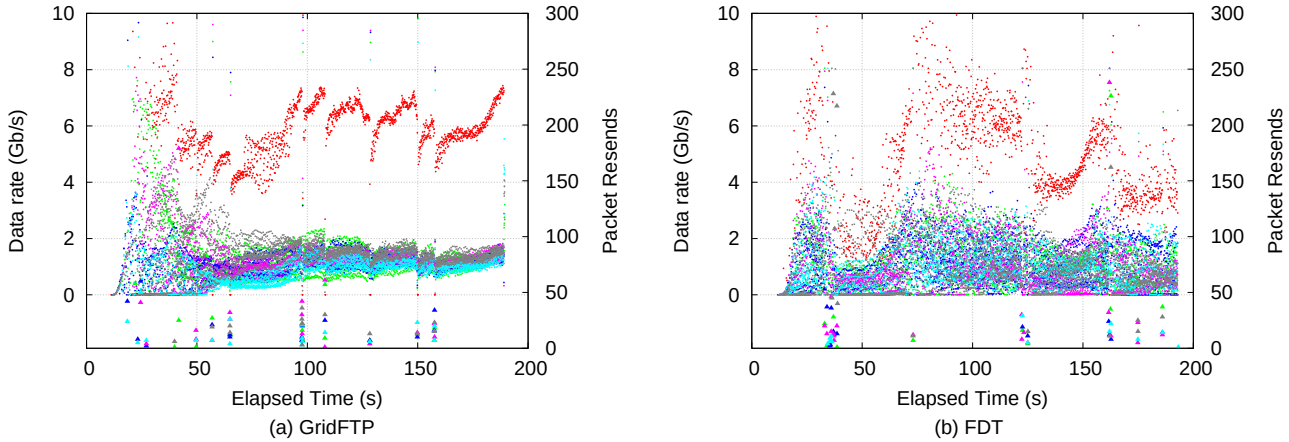

Fig. 6. Data rate (left axis) and packet resends (right axis) with multiple flows for GridFTP and FDT. Each flow has a unique color. Red represents the aggregated flow.

to use greater aggregate buffer space. This improves FDT performance overall compared to the single flow, and lowers the performance gap between GridFTP and FDT.

Because the RTT is high, it takes $322 \times 3\ ms$ to notice a packet loss. During this time, the TCP flow will keep sending packets as long as the congestion window size allows. TCP congestion window will grow when it receives the first acknowledgement packet, but not after it gets three duplicated acknowledgement packets. In this case, the number of packets sent before noticing the packet loss is larger than for flows with shorter RTT, which contributes more to the congestion (i.e., increase in RTT). Loss probability is proportional to a flow's throughput, so flows with higher throughput will be more likely to lose a packet. Although a flow with longer RTT will send more packets, flows with shorter RTT (that are likely to have higher throughput) will suffer greater packet losses, from which they will recover faster.

For multiple flows with the same RTT, a packet loss in one of the flows causes less packets to be sent during the period before it notices that packet loss. At the same time, the probability of packet loss for other flows will increase, again proportional to the throughput of the flow. If the other parallel transfer flows get packet losses, then their window size reductions will be similar, and the recovery time for the aggregated flow will be shorter compared to the single flow case, as shown in right hand side of Figure 5. Otherwise, the recovery time will be the same, but the aggregated congestion window size will be still higher compared to the single flow case.

In a network with shorter RTT (e.g., our national network testbed [9]), the benefit of using multiple flows is further reduced. With its smaller Bandwidth Delay Product (BDP) link, a smaller buffer size allows more throughput. With a small packet error rate, the effect of faster recovery gets diminished, and a single flow with moderate buffer size will be as efficient as multiple flows because the recovery time for any flow is relatively short.

Figure 6 shows the data rate of each flow in GridFTP and FDT. GridFTP flow data rates show less variation than those of FDT. This is because the GridFTP flows used a larger

TCP socket buffer size, allowing some flows to reach larger congestion window sizes before losing packets. That happens shortly before a packet is lost in a flow, but TCP CUBIC's congestion control uses the congestion window size before the last lost packet as the local maxima of the concave region of its congestion window size function. This means that GridFTP flows with a small burst before the packet loss have faster congestion window growth, and leads to more stable data transfer rates.

In contrast, flows in FDT do not allow their congestion window size to be larger than 50 MB, hence their local maxima is also limited to 50 MB. That causes some of the FDT flows to have packet loss while others do not, therefore they show less synchronized-flow behaviour. When the congestion window size for some flows reaches the limit, they have to wait for some data-acknowledging packets to arrive, stopping the flow for some time and thereby producing flow data rate fluctuations. These factors all contribute to the variability of the data rate of FDT flows, as we can see in Figure 6.

B. Impact on RTT

We measured RTTs while using GridFTP and FDT by measuring baseline RTT before running the experiment and then during the experiment using the `ping` command. We calculated the average time difference between the baseline RTT and observed RTT during the experiments for each protocol with varying number of flows to investigate the effect of congestion caused by the protocols building up queues at the routers in the path.

We found the average change in RTT increases as the number of flows increases. The average RTT change for two to five flows for GridFTP is higher, although it has high variability. Thus, it has more impact on RTT than FDT. GridFTP is able to generate greater aggregated traffic compared to FDT, and as a result, its impact on the network path is higher. Because both protocols use the same underlying congestion control algorithm, an increase in goodput for either protocol leads to an average increase in RTT change. Overall, the average RTT increase is higher with multiple flows because more traffic is injected by the multiple flows.

Figure 7 shows RTT during the flow, measured from arrival times of packet pairs, when five flows were used for the file transfer. Compared to RTT changes with a single flow, RTT increased more with five flows. Multiple data transfer flows have more aggregate throughput in our experiment, therefore more resources are required to process the packets. This causes network routers to take more time and it leads to an increase in RTT. RTTs in GridFTP flows show more peaks and spikes, because there are more bursts of traffic in GridFTP. The bursts of traffic overload the network path leading to instability in the network. When there are more bursts, RTT increases for both protocols. There was a steady increase of RTT between 80 *s* and 120 *s* in FDT for all flows. It was caused by the larger number of packets injected from multiple flows, causing the queues in routers to build up, increasing RTT for all the flows.

We have found that there are different baseline RTTs for each flow, one on top of another. We identified with `traceroute` that there is a load-balancer in the path that causes flows to be forwarded into different links for a part of their route. This produces the slight difference in RTT for each flow.

VI. Discussion

Our comparative analysis provided useful insights into performance and fairness of big data transfer protocols. We next discuss few important enhancements to file systems and protocol parameters to improve data transfer rates.

Faster file system: The bottleneck for most transfer systems lies in slow disk read/write rates. Although current mechanical Hard Disk Drives (HDDs) have file read rates faster than 200 MB/s, it is not easy to maintain sequential read rates greater than 100 MB/s over large files because of the mechanical structure of HDDs and file system overheads. Using multiple HDDs with data striping along with the ZFS file system or hardware RAID systems can provide significant improvements. Solid State Drives provide better performance, however, they are expensive.

Large TCP socket buffer size: Small TCP socket buffer size in either operating system parameters or TCP socket parameters limits the amount of data that can be sent at any given time. With larger buffers, even a single TCP flow can increase its congestion window size to an optimal point, and thereby achieve better performance. On long-haul links, the performance difference between GridFTP and FDT did not differ significantly, however, there are implementation problems that can limit transfer performance. We found technical difficulties with all the protocols we analyzed. GridFTP caused the fewest implementation problems in our experiments. GridFTP works better using both single flow and multiple flows in our intercontinental network compared to the other protocols, because it is able to set larger TCP socket buffer size.

Limited number of parallel flows: There are obvious benefits of using multiple flows such as overcoming small buffer size and reducing recovery time. The former benefit can be achieved with larger buffer size, as discussed above. Given that the monetary cost of the system memory is low, it is not hard to allocate a reasonable amount to TCP socket buffers. Rarely, the network link will have a BDP high enough to require multiple flows, even with larger TCP socket buffers. But, it is possible that high link speed can make the path's BDP large. In such cases, using multiple flows can be beneficial. In other cases, having more than three multiple flows can increase the RTT, causing more packets to be lost in the network path, reducing transfer performance.

Comparison with Sneakernet[7]*:* It is interesting to compare the performance of a high-speed network transfer with Sneakernet, which transfers data by moving physical storage media from one location to the other. Disregarding the need of disk read/write such as real-time streaming, one can send multiple

[7]https://en.wikipedia.org/wiki/Sneakernet

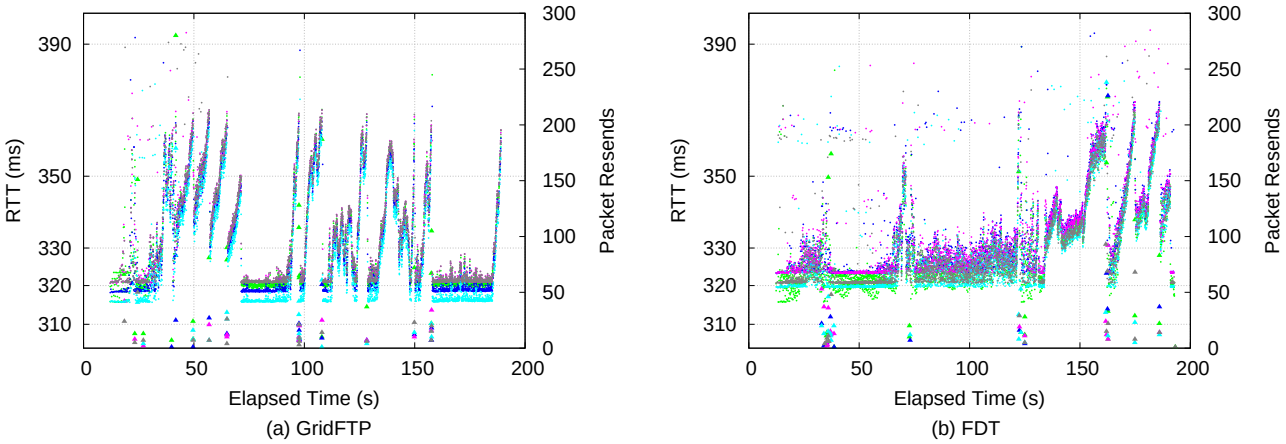

Fig. 7. RTT (left axis in log scale) and packet resends (right axis) with multiple flows for GridFTP and FDT. Each flow has a unique color.

HDDs via courier from New Zealand to Sweden. The fastest courier delivery service will take at least 24 hours. In 24 hours, a 10 Gb/s network can transfer 108 TB of data, if fully utilized. With the recent 8 TB shingled magnetic recording HDD, it would take at least 14 HDDs to hold the data. However, no matter how fast the courier can deliver physical media from one place to another, unless one's disk read/write speed is faster than 1.25 GB/s, transmitting over a 10 Gb/s network will always be faster because copying data from one disk to other will take longer than transferring data through high-speed network.

VII. Related Work

We present a discussion of prior research that have analyzed big data transfer protocols in high-speed networks.

Ha *et al.* [10] compared various TCP implementations in a 400 Mb/s bottleneck link. They found TCP Reno, FAST TCP, Scalable TCP, and Hamilton TCP had low utilization without background traffic on a long distance link. However, they only evaluated TCP implementations on a relatively slow bottleneck link and they did not consider using multiple flows of the same TCP variant to utilize more capacity.

Suresh *et al.* [11] evaluated throughput, fairness, and CPU utilization of GridFTP, GridCopy, and UDT. They found that UDT performed well compared to the rest of the tested protocols. GridFTP also performed better, except when transferring smaller files. While this study compared the performance of GridFTP with UDT, their testbed was restricted to using a 2 Gb/s campus network.

Tierney *et al.* [12] compared performance of RDMA over converged Ethernet, TCP, and UDP in ESnet's 10 and 40 Gb/s link with 47 *ms* RTT. Since, this work focused on comparing transport-layer protocols using a dedicated interface for RDMA, we cannot compare the results directly with our work. We use TCP or UDP over Ethernet and instead characterize performance and fairness of different application layer protocols. Furthermore, our experiments are performed in a very high RTT link, which is typical for transfers between New Zealand and Europe.

Cottrell *et al.* [13] compared TCP implementations with UDP protocols using a 1 Gb/s and a 10 Gb/s network. They found that Scalable, BICTCP, HTCP, and UDT performed well in the 1 Gb/s network, while UDT is more CPU intensive than the TCP implementations. In 10 Gb/s network, they found two sender-receiver pairs are required to saturate 10 Gb/s link capacity. Their work only focused on measuring the performance of transport layer protocols and did not consider behaviour of application layer protocols using different TCP parameters.

Zhaojuan *et al.* [14] compared UDP transfer protocols in a short distance 1 Gb/s network. They found PA-UDP is the most efficient and UDT is most convenient because it did not need user configuration. Their work was limited to comparing only UDP-based protocols in a limited capacity link.

Kissel *et al.* [15] developed an end-to-end measurement framework for identifying bottlenecks in big data transfers on multi-gigabit networks. Their system was able to monitor network and disk hardware statistics, and produce reports based on the data gathered through the monitoring process. They evaluated their system components by analyzing application and host metrics to understand bottleneck conditions during big data transfers using GridFTP. Their work was mostly focused on data collection for troubleshooting big data transfers, rather than understanding performance of the protocols themselves.

Yu *et al.* [8] compared GridFTP, FDT, UDT, and Tsunami in a 10 Gb/s local testbed network with the goal of understanding protocol behaviour over varying levels of emulated RTTs and background traffic. They found that UDP-based protocols were able to reach their maximum throughput much faster than TCP-based protocols. In a follow-up work [9], Yu *et al.* compared the fairness and performance of GridFTP, FDT, and UDT in a national 10 Gb/s link established between Auckland and Wellington. The analysis showed that GridFTP with jumbo frames provided fast data transfers, and GridFTP was fair in

sharing bandwidth with competing background TCP flows.

Our work complements [8] and [9] by analyzing performance and fairness of the big data transfer protocols in a high-speed international 10 Gb/s link. We use packet traces to provide detailed insights into the behaviour of each of the protocols. Our results provide a better understanding of the nuances of big data transfers using these open-source protocols over high RTT links. Furthermore, we also provide several recommendations on how to improve data transfer rates, which can be utilized by protocol developers and regular users in their own work. Note that an abridged (preliminary) version of this work appeared in [16].

VIII. Conclusions

We analyzed the performance and fairness of three big data transfer protocols, namely, GridFTP, FDT and UDT, in a long distance intercontinental network.

We found that UDT had an implementation issue that limited its performance, and FDT had problem working with large TCP socket buffer size limiting its performance, however, this problem is overcome by using multiple flows. To have better performance, faster file systems and larger TCP socket buffers in both the operating system and application are required. Running multiple flows for a high-speed transfer was beneficial, especially for FDT, where there was a practical limit on the TCP socket buffer size. It also reduced recovery time after packet loss events.

We did not find much difference between GridFTP and FDT data transfer rates, but GridFTP was more stable in terms of handling file system and TCP socket buffer. FDT is easier to set up because it is implemented as a Java application, which does not require a high level of system permission. However, administrator privilege is necessary to optimize the machine and TCP stack for improved performance.

References

[1] M. Allman, V. Paxson, and E. Blanton, *TCP Congestion Control*, ser. Request for Comments. IETF, Sep. 2009, no. 5681, published: RFC 5681 (Draft Standard). [Online]. Available: http://www.ietf.org/rfc/rfc5681.txt

[2] I. Legrand, H. Newman, R. Voicu, C. Cirstoiu, C. Grigoras, C. Dobre, A. Muraru, A. Costan, M. Dediu, and C. Stratan, "MonALISA: An agent based, dynamic service system to monitor, control and optimize distributed systems," *40 YEARS OF CPC: A celebratory issue focused on quality software for high performance, grid and novel computing architectures*, vol. 180, no. 12, pp. 2472–2498, Dec. 2009.

[3] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, and S. Tuecke, "GridFTP: Protocol extensions to FTP for the Grid," *Global Grid ForumGFD-RP*, vol. 20, pp. 1–21, Apr. 2003.

[4] Y. Gu and R. L. Grossman, "UDT: UDP-based Data Transfer for High-speed Wide Area Networks," *Comput. Netw.*, vol. 51, no. 7, pp. 1777–1799, May 2007. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2006.11.009

[5] M. Meiss, *Tsunami: A High-Speed Rate-Controlled Protocol for File Transfer*, 2009. [Online]. Available: www.evl.uic.edu/eric/atp/TSUNAMI.pdf/

[6] S. Ha, I. Rhee, and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, pp. 64–74, 2008.

[7] S. Ha and I. Rhee, "Taming the elephants: New {TCP} slow start," *Computer Networks*, vol. 55, no. 9, pp. 2092 – 2110, 2011. [Online]. Available: http://tinyurl.com/mptqu99

[8] Se-young Yu, N. Brownlee, and A. Mahanti, "Comparative performance analysis of high-speed transfer protocols for big data," *Local Computer Networks (LCN), 2013 IEEE 38th Conference on*, pp. 292–295, 2013.

[9] ——, "Performance and Fairness Issues in Big Data Transfers," in *Proceedings of the 2014 CoNEXT on Student Workshop*. Sydney, Australia: ACM, 2014, pp. 9–11.

[10] S. Ha, Y. Kim, L. Le, I. Rhee, and L. Xu, "A step toward realistic performance evaluation of high-speed TCP variants," *Elsevier Computer Networks (COMNET) Journal, Special issue on PFLDNet*, Feb. 2006.

[11] J. Suresh, A. Srinivasan, and A. Damodaram, "Performance Analysis of Various High Speed Data Transfer Protocols for Streaming Data in Long Fat Networks," in *Proc. International Conference on ITC*, Kochi, Kerala, India, Mar. 2010, pp. 234 –237.

[12] B. Tierney, E. Kissel, M. Swany, and E. Pouyoul, "Efficient data transfer protocols for big data," in *E-Science (e-Science), 2012 IEEE 8th International Conference on*, Oct. 2012, pp. 1–9.

[13] R. Les Cottrell, S. Ansari, P. Khandpur, R. Gupta, R. Hughes-Jones, M. Chen, L. McIntosh, and F. Leers, "Characterization and evaluation of TCP and UDP-based transport on real networks," *Annales Des Tlcommunications*, vol. 61, no. 1-2, pp. 5–20, Feb. 2006. [Online]. Available: http://dx.doi.org/10.1007/BF03219966

[14] Z. Yue, Y. Ren, and J. Li, "Performance evaluation of UDP-based high-speed transport protocols," in *Proc. IEEE International Conference on ICSESS*, Beijing, China, Jul. 2011, pp. 69 –73.

[15] E. Kissel, A. El-Hassany, G. Fernandes, M. Swany, D. Gunter, T. Samak, and J. Schopf, "Scalable integrated performance analysis of multi-gigabit networks," in *Network Operations and Management Symposium (NOMS), 2012 IEEE*, Apr. 2012, pp. 1227–1233.

[16] Se-young Yu, N. Brownlee, and A. Mahanti, "Characterizing performance and fairness of big data transfer protocols on long-haul networks," in *Proceedings of the IEEE Local Computer Networks (LCN)*, Clearwater, USA, 2015.