Efficient High Performance Protocols For Long Distance Big Data File Transfer

A Thesis Submitted to the
College of Graduate and Postdoctoral Studies
in Partial Fulfillment of the Requirements
for the degree of Master of Science
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By
Fadi AlMobayed

Permission to Use

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis. Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Dean
College of Graduate and Postdoctoral Studies
116 Thorvaldson Building, 110 Science Place
University of Saskatchewan
Saskatoon, Saskatchewan
Canada
S7N 5C9

Department Head
Department of Computer Science
176 Thorvaldson Building, 110 Science Place
University of Saskatchewan
Saskatoon, Saskatchewan
Canada
S7N 5C9

Abstract

Data sets are collected daily in large amounts (Big Data) and they are increasing rapidly due to various use cases and the number of devices used. Researchers require easy access to Big Data in order to analyze and process it. At some point this data may need to be transferred over the network to various distant locations for further processing and analysis by researchers around the globe. Such data transfers require the use of data transfer protocols that would ensure efficient and fast delivery on high speed networks.

There have been several new data transfer protocols introduced which are either TCP-based or UDP-based, and the literature has some comparative analysis studies on such protocols, but not a side-by-side comparison of the protocols used in this work. I considered several data transfer protocols and congestion control mechanisms GridFTP, FASP, QUIC, BBR, and LEDBAT, which are potential candidates for comparison in various scenarios. These protocols aim to utilize the available bandwidth fairly among competing flows and to provide reduced packet loss, reduced latency, and fast delivery of data.

In this thesis, I have investigated the behaviour and performance of the data transfer protocols in various scenarios. These scenarios included transfers with various file sizes, multiple flows, background and competing traffic. The results show that FASP and GridFTP had the best performance among all the protocols in most of the scenarios, especially for long distance transfers with large bandwidth delay product (BDP). The performance of QUIC was the lowest due to the nature of its current implementation, which limits the size of the transferred data and the bandwidth used. TCP BBR performed well in short distance scenarios, but its performance degraded as the distance increased. The performance of LEDBAT was unpredictable, so a complete evaluation was not possible. Comparing the performance of protocols with background traffic and competing traffic showed that most of the protocols were fair except for FASP, which was aggressive. Also, the resource utilization for each protocol on the sender and receiver side was measured with QUIC and FASP having the highest CPU utilization.

Acknowledgements

I would like to take this opportunity to express my appreciation to everyone who have helped me and made me succeed in completing my thesis. Firstly, I want to express my sincere appreciation and genuine gratitude to my supervisors Dr. Dwight Makaroff and Dr. Derek Eager who helped me from the beginning of my master program at the U of S. Both of my supervisors assisted me in finding the right direction and helped me in resolving any issues that I faced during my studies. In addition, they provided me with many suggestions and ideas to in order to choose a suitable thesis topic. I do also appreciate the time they spent on verifying and correcting any issues with my thesis. Besides my supervisors, I would like to thank the rest of my committee members: Dr. Kevin Schneider and Dr. Ray Spiteri for their suggestions. I would also like to thank Greg Oster, Carl Hofmeister, and my colleagues for helping me with my experimental setup and in resolving any technical issues I faced during my experiments. Lastly, I am so grateful for my parents and friends who always kept me in their prayers and provided me with endless encouragement and support.

Contents

List of Tables

List of Figures

List of Abbreviations

ACK	Acknowledgment
AQM	Active Queue Management
ASCP	Aspera Secure Copy
BBR	Bottleneck Bandwidth and Round-trip Propagation Time
BDP	Bandwidth Delay Product
CPU	Central Processing Unit
CWND	Congestion Window
FASP	Fast and Secure Protocol
FTP	File Transfer Protocol
GridFTP	Grid File Transfer Protocol
GNU/Linux	GNU is Not Unix/Linux
IP	Internet Protocol
LEDBAT	Low Extra Delay Background Transport
SCP	Secure Copy Protocol
NIC	Network Interface Card
OS	Operating System
QUIC	Quick UDP Internet Connections
RTT	Round Trip Time
SSH	Secure Shell
SACK	Selective Acknowledgments
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UDT	UDP-based Data Transfer

Chapter 1

Introduction

1.1 Problem and Motivation

For every data transfer that occurs on a network, several metrics such as goodput, fairness, packet loss, and delay may be examined to determine the quality of a network path and the performance of the protocol used to transfer data. System designers and network operators desire protocols that provide efficient use of resources and deliver the content as quickly as possible. Unfortunately, these goals often conflict and trade-offs must be made to satisfy the use cases that must be supported.

Goodput is described as the useful data (bits) transferred over a link in a given amount of time. The useful data means the original data sent excluding any re-transmission packets or protocol overhead such as frame headers and other data wrapped around application data. Therefore, goodput is defined as the number of useful data bits transferred over the transfer duration. The goodput is always expected to be lower than the throughput (the rate at which data is transferred over a network link) and the bandwidth (network connection speed).

Fairness is another measure to determine if protocols are receiving their fair share of network bandwidth. Most protocols achieve fairness by being equally aggressive. This implies that multiple flows can fairly share the bottleneck link by obtaining approximately equal shares of the link capacity. On the other hand, if a protocol such as TCP LEDBAT is sharing the bottleneck link with other flows, LEDBAT reduces its sending rate so that it does not add extra delay or impact the performance of the other flows.

Another metric is packet loss, which is described as the percentage of packets that fail to reach their destination. Packets may be dropped or corrupted on the network or at the receiver, so only part of the network path from source to destination deals with these undelivered packets. Another metric is delay, which is a measure of how long a packet takes to travel from one host to another across the network during a data transfer.

A data transfer can also be affected by congestion that occurs on the network because of a single flow or other competing flows. Congestion overloads some links or routers, causing delay in forwarding, or in extreme cases, dropping of packets due to capacity constraints [41]. To mitigate such congestion, loss-based, delay-based or hybrid congestion control mechanisms are implemented in most of the current data transfer protocol [36].

The main physical network characteristic considered in this thesis is long distance transfers, in which the bandwidth delay product is large. The bandwidth delay product differs based on the amount data sent during a round trip time. When the bandwidth delay product is small, the sender receives the feedback from the receiver for delay or loss indications quickly, thus the feedback for the transmitted data is received before any more data is sent. When the BDP is large, large amounts of data are being transmitted at high speeds. This requires the routers to have large buffers as the feedback delay can cause instability. Moreover, most of the work on congestion control mechanisms has been focused on dealing with large BDP networks and reducing latency. The high latency over long distance is also accentuated by the increasing number of hops between a sender and a receiver, as there are usually more network devices present along the network path. With more network devices on the path, there are more routers in which packets could queue, waiting for service, potentially increasing round trip time and packet losses on a network path. With increasing round trip times (RTTs) and packet loss, the performance of transport layer protocols may degrade according to their particular implementations and cause them to behave differently with respect to reliable file transfer.

There are several transport and application level protocols used nowadays to transfer Big Data over long distances that use Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) as their underlying transport protocol. TCP sends data reliably over established connections. It has mechanisms to reduce packet loss and make effective use of network bandwidth (by flow control, rate control and congestion control). By default, UDP is more lightweight than TCP, as it has none of these mechanisms and does not require connection establishment. Thus, in many situations, it is more aggressive than TCP, as it does not ensure that the packets are received at the destination, leaving that for a higher level protocol.

Several studies [4, 36, 50] examined various congestion control algorithms that can achieve reduced packet loss rates and better link utilization for single and multiple transfer flows, but their performance was examined in simulation-based environments. Also, there are studies [29, 51, 53] that compared the performance of several higher layer data transfer protocols using TCP and UDP at a coarse level of analysis.

Additionally, current implementations of data transfer protocols provide various features that improve the performance of data transfers. Such features include high utilization of available bandwidth, multiple data streams, multiple transfer flows, fairness with competing flows, and data pipelining [1, 47]. The utilization of available bandwidth feature provides a bandwidth control mechanism, which includes pre-setting data transfer rates by end users in Fast and Secure Protocol (FASP) [47]. The bandwidth is also influenced by the number of data streams and transfer flows established. With multiple flows, data connections are established in parallel between a sender and receiver and multiple streams are used to provide high utilization of available bandwidth. With data pipelining, commands and requests are sent one after another, without waiting for a response or an acknowledgement before sending the next one and this is used in Grid File Transfer Protocol (GridFTP) [1].

Moreover, data transfer protocols can perform better when they run on an optimized host with a faster disk system, large socket buffers, and a high-speed link [53]. With faster disk systems, the host will allow data

transfer protocols to read/write data more efficiently and large socket buffers provide an increased congestion window size.

This thesis aims to quantify the performance of several data transfer protocols and their characteristics with particular emphasis on various congestion control mechanisms in various network environments. Data transfer protocols such as GridFTP [1] and FASP [47] are included in this work, because they were designed for long distance big data transfers. Quick UDP Internet Connections (QUIC) [26] and Bottleneck Bandwidth and Round-trip propagation time (BBR) [7] are included because they are considered new and under development protocols, and they are currently undergoing an expansion of interest in the research community [8, 17, 23, 40]. Additionally, Low Extra Delay Background Transport (LEDBAT) [43] was selected for investigation because it is used by the Bittorrent application and by Apple for software updates, and provides good performance when dealing with background traffic.

1.2 Thesis Objectives

The performance of data transfer protocols can be affected by various factors such as the congestion control mechanism used, the bandwidth of the link, and the distance a transfer has to travel [44]. This thesis contributes to the research body of existing knowledge by examining how the data transfer protocols perform under various network conditions.

The main research question is the following: How do data transfer protocols and congestion control algorithms perform when used for short and long distance transfers?

The scale and topology of a network affect its performance with respect to packet delivery. A network environment can be inefficient due to buffering issues, increased traffic and router configurations along the network path. By testing the data transfer protocols in different network environments, the thesis experiments determine how these protocols deal with such network issues and which protocol would be more suitable for users to implement in a certain network setup. The performance of data transfers in local, national, and international networks will be examined. Table 1.1 presents a detailed list of research questions.

Table 1.1: List of Research Questions

No.	Cause	Question	Task
1	Protocol	How do the data transfer protocols perform when using different file sizes and various RTTs (transfer locations)?	Examine the performance results of protocols using different file sizes and various RTTs
2	Network	What is the effect of data transfer protocols and background traffic on each other?	Analyze the effect of protocols on background traffic performance and the opposite.
3	Network	How do the data transfer protocols perform with multiple flows/streams?	Analyze the effect of multiple transfer flows on the protocols performance.
4	Network	How do protocols perform when competing traffic exists?	Analyze the effect of competing traffic and the fairness of each protocol.
5	Host	What effect does each protocol have on CPU utilization?	Observe the CPU usage for each protocol during a data transfer.

1.3 Thesis Contributions

The goal of this work is to evaluate the performance of several data transfer protocols in various scenarios. The experiments compared the performance of the protocols in terms of goodput, packet loss, and fairness to competing traffic.

The first contribution is an evaluation of several data transfer protocols [1, 26, 47] and congestion control mechanisms [7, 43] that were not compared side-by-side in a real 1 Gb/s network environment. Some of these protocols were evaluated independently using emulated network conditions and were not evaluated for long distance transfers with large file sizes [17, 23, 45]; therefore, their performance is evaluated in a real network environment with different types of file transfers.

As a particular deployment scenario, images collected for the Plant Phenotyping and Imaging Research Centre (P2IRC)[1] project were bundled as tar files and used for the transfers. This is an example use case where data such as crop images are collected on daily basis and a large number of images are required to be transferred to various research sites for further analysis. This is a critical process and requires efficient data transfer protocols that can transfer these files at high speeds. In this thesis, the tar image files were used in the transfers.

Therefore, experiments were conducted using various data transfer protocols in a real-world network to help understand the performance of these protocols in a real network with conditions not under laboratory control. The experiments evaluated the behaviour of the chosen protocols using single flows and multiple flows. The protocols were also tested in the presence of various network events such as competing traffic and background traffic. All these experiments were run using various file sizes transferred to multiple locations.

[1] https://p2irc.usask.ca/

1.4 Thesis Statement

This thesis determines whether or not the congestion controls and protocols cause different responses to network conditions and quantifies the difference in these performance characteristics. The network conditions evaluated are transfer latency, background traffic and congestion. This insight can help network operators and developers utilize the protocol best suited for their environment with proper configurations.

To validate this claim, a 1 Gbps physical network connection from the sender machine was used to connect to machines distributed over different geographical locations (local, national, and international), where the Internet was treated as a black box. Additionally, the following experiments were performed: various file sizes experiments, transfers with background traffic, transfers with multiple flows, and transfers with competing traffic. Also, statistics are gathered about the goodput and packet loss for each protocol. This gives realistic results about the performance of the protocols and congestion control mechanisms, where there are routers that impose certain policies and other protocols that act as competing traffic on the same network link. The protocols chosen in this thesis are GridFTP, FASP, QUIC, TCP BBR, and TCP LEDBAT.

The overall results of these experiments showed that GridFTP had the best performance in most of the scenarios, while FASP performed better than all the protocols on the international link. On shorter links, TCP BBR was able to perform better than the other protocols. TCP BBR and FASP showed less variation between the runs for most of the setups compared to the other protocols. The variation over time between the runs demonstrated the different response to packet loss, delay and congestion. In some cases, packet loss affected the goodput of the protocols while in other cases there were other reasons. Moreover, QUIC showed a poor performance when compared to the performance of all the protocols, but the results were consistent for most of the scenarios.

1.5 Thesis Organization

The rest of the thesis document is organized as follows. Chapter 2 covers the background of transport layer and application file transfer protocols along with their associated congestion control mechanisms. It also contains recent related work on studies of data transfer protocols evaluations. Chapter 3 describes the experimental design and configuration. Chapter 4 presents the description of the results and analysis. Chapter 5 outlines the conclusion and ideas for future work.

Chapter 2

Background and Related Work

This chapter provides a brief background of TCP and UDP and a description of the data transfer protocols and congestion control mechanisms included in this thesis. Following that, related studies to the area of evaluating data transfer protocols and congestion control mechanisms are evaluated. Section 2.1 presents the general structure of the transport layer protocols and congestion control mechanisms followed by an overview of TCP, UDP, and Related Protocols. Section 2.2 provides a description of the candidate protocols' features and target deployment. Section 2.3 explores related work in the area of comparative analysis and evaluation of data transfer protocols. Section 2.4 provides a summary of chapter 2.

2.1 TCP, UDP, and Related Protocols

The transport of data between a sender and receiver is implemented by transfer protocols in the transport layer. The current transport layer protocols being used on the Internet are Transmission Control Protocol (TCP) and User Datagram Protocol (UDP).

TCP is connection oriented and has additional features such as connection establishment, connection release, error and flow control [46]. The data in the a TCP connection is delivered reliably. However, the additional TCP features can add overhead and potentially delay for the sake of reliability. The use of congestion control mechanisms on a network is crucial. Without it, the sender may transmit too many packets at high speed for the network capacity, and may cause network congestion at the routers [46]. In some cases, intermediate routers cannot store packets to be forwarded because of congestion, and thus they are dropped. Even if the capacity exists, long router queues result in additional latency.

The only way to control congestion is by reducing the rate at which each flow sends packets over the network according to the corresponding protocol policies. Additionally, a good congestion control mechanism should be able to allocate bandwidth to transport protocols fairly across competing traffic and track any changes in traffic to modify the bandwidth allocation accordingly. In order to obtain a desirable bandwidth allocation, sending rates need to be regulated depending on factors such as flow control and congestion.

2.1.1 TCP Congestion Control and Reliability

TCP was desgined to provide reliable delivery of data by re-transmitting lost packets [46]. TCP uses the congestion window size (CWND), defined as the number of unacknowledged data segments that can exist at any point in time to limit the sender from overwhelming the network. It also limits the speed of the transfer and the data segments a sender can send at any time [46]. The congestion window size is determined by TCP slow start, which occurs after a three-way handshake is initiated, and provides congestion avoidance features in TCP. The TCP slow start mechanism allows the sender to send a packet with an initial congestion window (CWND). Once the receiver sends an acknowledgement (ACK) to the sender, the number of packets in the cwnd is doubled by the sender. The sender keeps increasing the number of packets sent until the receiver stops replying with ACKs, which means ACKs could be lost or the receiver's congestion window limit has been reached. Then, the connection transitions into congestion avoidance mode, in which the sender may send more packets over time or the transfer may slow down if packet loss is inferred. Packet loss can cause the sending rate to decrease while in congestion avoidance mode and TCP would re-transmit lost packets, after which the sending rate increases again.

TCP slow start helps prevent a network path from becoming congested by controlling the amount of data sent until a reasonable share of the achievable network bandwidth is determined. The TCP congestion avoidance mechanism probes the sending rate that a link can handle by increasing TCP's congestion window size slowly. This allows the sender to send more packets over time and it would slow down the speed of the transfer if it infers packet loss.

Moreover, a handshake can add latency to the start of data transmission, but in a lossy network, it could reduce overall latency, by ensuring that all the packets get there using system-level mechanisms, rather than having to use application-level mechanisms in other UDP-based reliability schemes.

2.1.2 TCP Variants

There have been other enhancements to TCP for various purposes such as high-speed data transfers on wide area networks. The last 20 years have seen many more enhancements to TCP than are within the scope of this thesis. The remainder of this section describes those TCP variants that have been used in the related work section.

Scalable TCP (STCP) [24] is a loss-based algorithm and it enhances the standard TCP congestion control mechanism to improve the data transfers on wide area network (WAN) links. It increases its congestion window aggressively to enable high utilization of the link and slowly decreases its window after packet loss occurs.

HighSpeed TCP (HSTCP) [12] is also loss-based and focuses on the performance over high bandwidth delay product links. It modifies TCP's response function and grows its window quickly, and recovers from packet loss more efficiently than standard TCP.

YeAH (Yet Another Highspeed) [3] is a hybrid algorithm that operates with two modes: fast mode and slow mode. It rapidly increases its congestion window in fast mode and implements a de-congestion algorithm during slow mode. It also adjusts its sending rate to be fair to other TCP variants.

Westwood+ [31] is a loss-based algorithm that estimates the bandwidth of the link and uses that estimate to adjust its congestion window. It also performs bandwidth sampling every RTT.

Hamilton-TCP (H-TCP) [28] is a loss-based congestion control algorithm, that controls the TCP congestion window using additive-increase/multiplicative-decrease (AIMD). It uses the elapsed time instead of the congestion window to determine the bandwidth-delay product of the link.

Fast Data Transfer (FDT) [27] is a protocol that uses TCP as its transport layer protocol. It uses parallel TCP connections and concurrent threads during a transfer to send data at high sending rates.

TCP Lola [18] is a delay-based congestion control mechanism that at achieving high network utilization and low queuing delay in wide area networks. it uses a mechanism called "fair flow balancing" to achieve high network utilization and fairness to other flows with each flow having a different RTT. In this context, fairness means that each flow should keep a low but similar amount of data in the queue of the link.

Performance-oriented Congestion Control (PCC) [9] is a new congestion control mechanism that allows the sender to monitor the connection and perform actions that results in a high performance. PCC tests the link by sending packets at a certain rate for a short period of time and uses a utility function to determine an ideal sending rate.

FAST AQM Scalable TCP (FAST) [21] was introduced as a hybrid type of congestion control algorithm that uses both queuing delay and packet loss to determine if a network is congested. FAST uses an estimation component which provides the average RTT and the average queuing delay to update its congestion window.

The remaining TCP variants in this section are considered for evaluation in this thesis because they are new and popular for long distance big data transfers.. A brief description is provided here for completeness and further details will be provided later in this Chapter.

The TCP variants used in thesis are 1) Bottleneck Bandwidth and Round-trip propagation time (BBR) [7] is an experimental congestion control algorithm developed by Google which attempts to operate the TCP session more efficiently at a path with a bottleneck link. 2) GridFTP [1] is based on the File Transfer Protocol (FTP) [38] extended with additional enhancements for a grid-based environment. 3) Low Extra Delay Background Transport (LEDBAT) [43] is a delay-based congestion control algorithm that seeks to utilize the available bandwidth while trying to reduce the queuing delay present on the network path. The specific reasons for choosing these protocols are give in each subsection.

2.1.3 UDP Variants

The User Datagram Protocol (UDP) was designed to provide light-weight data transmission without reliability guarantees [37]. This makes it beneficial for time sensitive applications such as video streaming and VoIP calls when most, but not necessarily all, packets are successfully delivered. By default, there are no mechanisms

for flow control or congestion control in UDP. UDP provides applications and upper layer protocols with an opportunity to implement their own features on top of it. One of these could provide reliable transmission [46]. Data transfer protocols such as FASP [47], QUIC [26], and UDP-based Data Transfer Protocol (UDT) [14] can support various types of congestion control algorithms and provide reliable communication on top of UDP for long distance data transfers.

UDT [14] is a UDP-based data transfer protocol that was designed to achieve high throughput and fairness with other flows. It implements its own rate-based congestion control mechanism called DAIMD (AIMD with decreasing increases), which limits the number of unacknowledged packets.

Performance Adaptive UDP PA-UDP [10] is data transfer protocol that uses UDP and consists of a sender and receiver applications. The sender sends packets based on the sending rate specified by the receiver. This is achieved by sending a three-way handshake from the sender and the receiver various metrics such as receiving rate, packet loss and buffer size for determine the sending rate.

Reliable Blast UDP (RBUDP) [15] is a UDP-based protocol that tries to utilize the available bandwidth by sending data using at a certain target rate set by the user. Once a data transfer is completed, the receiver sends a report with all the lost packets to the sender using TCP.

Tsunami [32] is another UDP-based data transfer protocol that uses its own congestion control algorithm and rate control mechanism to send data over high-speed networks. Tsunami divides the data into blocks that get encapsulated into IP packets and sends the data to the receiver. Any lost blocks are reported by the receiver after a predefined period of time.

The remaining UDP variants in this section are considered for evaluation in this thesis since they were developed by major companies such as Google and IBM and they were not evaluated much in the literature, especially FASP. A brief description is provided here and further details will be provided later in this chapter.

The Fast and Secure Protocol (FASP) [47] a proprietary protocol from IBM, also aims to eliminate the issues with TCP based file transfer technologies such as FTP and HTTP. It uses TCP to initiate the connection and UDP to transmit the data packets efficiently, but has a proprietary reliability mechanism. FASP aims to achieve high file transfer speeds regardless of the file size, transfer distance or network conditions.

Quick UDP Internet Connections (QUIC) [26] is as a multiplexing protocol which runs UDP internet connections placed in the transport layer in an attempt to replace TCP (Transmission Control Protocol) for file transfers and for web objects on the Internet

These protocols use UDP, but differ in the mechanisms they use to adjust the sending rate, react to congestion and other traffic. For example, UDT uses a rate-based congestion control mechanism (DAIMD) while QUIC uses CUBIC (cubic function of Binary Increase Congestion control) as its congestion control mechanism. Tsunami divides a file into 32 KB sized chunks and sends them to the destination endpoint and uses its own congestion control mechanism. PA-UDP has its own performance-based system flow control but not RB-UDP. RB-UDP does not consider fairness and would cause high levels of congestion on the network. Compared to these protocols, FASP focuses on encrypting the packets and sending these packets as fast as

possible regardless of packet loss.

2.1.4 Congestion Control Algorithms Details

Many congestion control algorithms were developed in an attempt to control network congestion, improve fairness and increase throughput in networks. These congestion control algorithms are either loss-based, delay-based, or a hybrid type [36]. Loss-based algorithms such as Tahoe, Reno, New Reno, and CUBIC rely on packet loss events to determine if a network is congested while the delay-based algorithms such as Vegas detects congestion by looking at the increasing amount of queuing delay over time [36]. A Hybrid type of congestion control algorithms such as FAST AQM (Active Queue Management) Scalable TCP (FAST) relies both on queuing delay and packet loss to determine if a network is congested [39]. Some alternate data transfer protocols only use the default TCP congestion control (CUBIC) [1, 55], while others implement congestion control mechanisms that run over TCP [7, 18, 43] and these will be briefly described in the remainder of this Section.

The first loss-based algorithm developed was TCP Tahoe, which was deployed in 4.3 BSD Tahoe in 1988 by adding new algorithms such as slow-start, congestion avoidance, and fast re-transmit [11]. The fast-retransmit algorithm is used to re-transmit lost packets before the re-transmission timer expires. This happens when TCP receives three duplicate acknowledgments (ACK packets) that belong to the same data segment during data transmission [11]. Thus, a packet loss is inferred by the sender, enabling earlier re-transmission and potentially high throughput. Tahoe returns to slow start upon recognizing packet loss.

Enhancements to Tahoe were undertaken by the TCP Reno congestion control algorithm; it modified the fast-retransmit feature to include Fast Recovery [11]. TCP Reno implements fast recovery to prevent a network path from being empty after Fast Retransmit by refraining from using slow start after a packet loss event has occurred. For every duplicate acknowledgment packet received, Fast Recovery assumes that a packet has left the network path and it is able to provide an estimate of the amount of data left for transmission. TCP Reno Fast Recovery mechanism improved the performance of TCP Tahoe in the case when one packet is dropped but faces performance issues when multiple packet losses occurs in a single sending window. Fall and Floyd [11] showed how this issue of TCP Reno can occur in a large congestion window TCP connection where multiple packets are being lost. Furthermore, Reno will perform a fast re-transmit without entering slow start and will halve the congestion window after receiving three duplicate ACKs.

A modified version of TCP Reno called TCP New Reno [16] was introduced to enhance the Fast Recovery feature in TCP Reno. The modification involved removing TCP from the fast recovery phase by reducing the window size used for sending data to the same size of the congestion window, which efficiently handled partial acknowledgments [11]. Also, New Reno ensures that all lost packets are re-transmitted and stays in the fast recovery stage until acknowledgments for all the queued data processed in Fast Recovery have been received.

TCP Vegas [5] is one of the delay-based congestion control algorithms that uses RTT instead of packet

loss to detect and prevent further packet loss in data transmissions on the network. It looks at the RTT and throughput achieved on a network path and it modifies its congestion window accordingly. Also, TCP Vegas tries to ensure that a minimum number of extra packets are queued at the bottleneck link by increasing or decreasing its sending rate.

TCP CUBIC [49] is a modified version of the original BIC (binary increase congestion control) which improves RTT fairness and the window control used in BIC. It uses a cubic function to increase the window size rapidly and then steadily until it reaches a target window size, which is the window size before the last congestion event. Once it reaches the target window size, it will slow down the growth of its window. CUBIC starts probing for more bandwidth and creates a new window size. The new window size is calculated using the elapsed time since the last lost packet and the last target congestion window size. This enhances the protocol by stabilizing and improving its utilization of the network. Besides window control, CUBIC achieves good RTT fairness by allowing various CUBIC flows running on the same bottleneck link to acquire the same window size regardless of the RTT for each flow. This is achieved by depending on the time elapsed instead of RTT after a packet is lost to control the window growth rate. This process improves RTT fairness and allows CUBIC to fairly share throughput with other protocols on the network bandwidth. Additionally, TCP flow control is used to ensure that the receiver's buffer is not overwhelmed by the sender's packets. This is achieved by the receiver sending its own receive window size, which shows the available size of the receive buffer. For each packet received, the receiver sends an ACK packet to the sender, which confirms the receipt of each packet and includes the current receive window size, so the sender knows how much data it can send. Moreover, the growth rate of CUBIC could be slow with short RTTs, because the window growth rate is fixed. However, the TCP-friendliness of the protocol is improved by this feature. CUBIC is considered to be the most widely used congestion control for TCP and it is deployed by default in the Linux kernel.

2.2 Candidate Protocols

2.2.1 BBR

Bottleneck Bandwidth and Round-trip propagation time (BBR) [7] is an experimental congestion control algorithm developed by Google which attempts to operate the TCP session more efficiently on a network path. BBR does not depend on packet loss to determine a congested link; rather it is model-based. BBR measures two parameters: bottleneck bandwidth and round-trip propagation (RTT) to characterize a path as congested or not. It also depends on delay as it uses RTT, which is affected by delay on the network.

Based on the measurements gained by BBR, it attempts to fully use the bandwidth available on the link despite any packet loss. The values of RTT and bottleneck bandwidth are independently measured, and the values change independently without impacting each other. Furthermore, RTT and bottleneck bandwidth change over the life of a connection, so they must be continuously estimated. With each ACK packet received, BBR records both the RTT and the sending rate of that packet. When packets are sent, the

estimated bottleneck rate is used to avoid network queuing (delay). The consequences of network queuing are explained in the next paragraph.

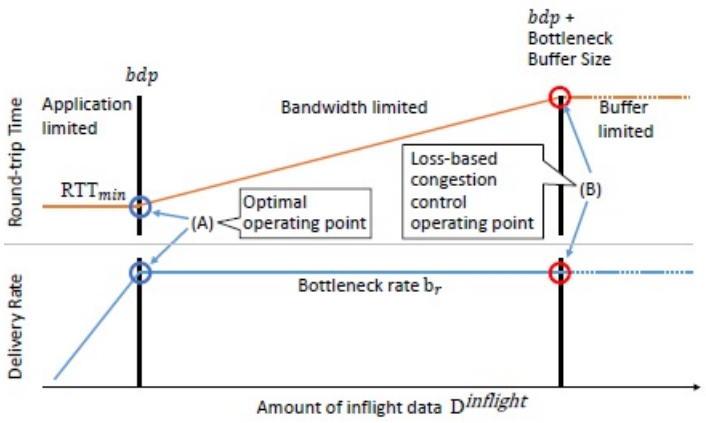


Figure 2.1: Congestion Control Operating Points: Delivery Rate and Round-Trip Time vs. Amount of Inflight Data [17]

As can be seen in Figure 2.1 for a single sender, the operation of BBR involves measuring round-trip time and delivery rate, both of which vary with the amount of inflight Data (Dinflight). When the amount of inflight data is equal to the bandwidth delay product, there is full utilization of the bottleneck link [7]. This is considered to be the best operating point (A), since the bottleneck is fully utilized. If the link is congested, this could cause a buffer overflow. Once the bottleneck is fully utilized, the delivery rate does not change and data cannot be delivered at a faster rate.

A queuing delay is caused when excess data is in the buffer and it rises with as the amount of inflight data increases. The loss-based congestion control algorithms operate at point (B) as shown in Figure 2.1, which has a high end-to-end delay and leading to excess buffering packets of packets when the buffer size is large. This produces an increased amount of delay and packet loss when the bottleneck bandwidth is fully utilized [7]. Additionally, BBR calculates the BDP based on estimated values of the available bottleneck bandwidth rate (br) and the minimal RTT (RTTmin). The bottleneck bandwidth rate is used to control the transmission rate of the sender (sr) along with pacing. Therefore, BBR does not depend on a congestion window [7]. BBR goes through two main phases to determine the BDP: 1) Bandwidth probe phase and 2) RTT probe phase.

BBR probes for bandwidth and RTT in an operational phase called steady state operation. In this mode, the BDP is derived after obtaining estimates of the bottleneck rate (br) and the minimal RTT(RTTmin). Also, BBR uses TCP acknowledgments to calculate the target sending rate and the minimal RTT.

BBR probes for bandwidth by changing its sending rate. It first increases its sending rate by setting the pacing gain (sending rate factor) to 1.25 for an estimated minimal RTT value, after which the pacing gain is reduced by 0.75. This helps in compensating for any excess amount of inflight data, which might fill the bottleneck queue [17]. If the queue gets filled with excess inflight data, it would be drained by the

same amount of inflight data later on. In addition, BBR probes for bandwidth using an approach called gain cycling. This involves going through an eight-phase cycle with eight different pacing gain values that are used to probe for bandwidth. Each phase is first used to probe for bandwidth with a pacing gain above 1 and then drains any existing queues with a pacing gain below 1. This helps BBR in gaining more throughput, reducing the queuing delay, and providing fairness to other flows. Overall, the aggregate pacing gain used across all the phases is 1 as BBR aims at having a pacing gain equal to the available bandwidth with a small queue on the link.

In BBR's startup phase, BBR uses a pacing gain of 2 and 3 bandwidth delay product (3BDP) in an attempt to increase its sending rate for every RTT value as the delivery rate increases [7]. Moreover, BBR discovers the bandwidth limit of the link by looking at how the delivery rate varies during a data transfer. This is achieved by observing how the delivery rate reacts to the increase in the sending rate. [7].

2.2.2 QUIC

QUIC was developed by Google in 2013 as a multiplexing protocol which runs UDP internet connections placed in the transport layer in an attempt to replace TCP (Transmission Control Protocol) for file transfers and for web objects on the Internet.[1] This Section provides an overview of QUIC as described by Langley *et al.* [26]. Google aims to mitigate some of the design issues with TCP using QUIC and this includes reducing the handshake and head-of-line blocking (HOL blocking) delays, having zero RTT connection establishment, and applying FEC (Forward Error Correction) to recover lost packets. The handshake that QUIC uses contains both cryptographic and transport handshakes in an attempt to reduce the handshake delays or setup RTTs [26]. This is achieved by eliminating similar handshake overhead at multiple layers and using server credentials that are already recorded for the same connection.

For security, QUIC uses Transport Layer Security Protocol (TLS)-like security mechanism called QUIC-crypto which encrypts and authenticates packets to prevent fabricating or spoofing packets payload when running through different network hops. Furthermore, QUIC recovers lost packets by providing packets with unique numbers and by using special ACK packets to measure RTT accurately [26].

For connection establishment between the QUIC server and client, the client connects to the server by performing a handshake, which includes cryptographic information and transport handshake combined [26]. This information is retrieved when the client sends a client HELLO (CHLO) packet to the server to produce a REJ message from the server which includes the required information. Once the client has received the server's configuration, it verifies the configuration by using the server's certificate and sends a complete CHLO after which the handshake would be successful which leads to initiating a stable connection which involves sending requests to the server to retrieve files.

Since the client is able to send or receive data from the server and wishes to obtain zero round trip time (0-RTT) connection establishment, it can achieve this by sending data with its initial keys without waiting

[1] https://www.chromium.org/quic

for the server's reply. The server replies to the client with a server hello (SHLO) message which contains the cryptographic information. The public key is required by both the client and the server to start sending packets encrypted with forward-secure keys, which are a type of secret key that aim to prevent decryption of past encrypted sessions. This type of cryptography provides confidentiality through encrypting client data using initial keys and encrypting data from the server and client after a SHLO message using forward-secure keys [26]. For any future connections between the server and the client, a connection is started by sending a complete CHLO without needing a response message from the server since the client has already cached the server configuration and the source-address token of the server. With that being said, a complete CHLO might not work if the server configuration or the server certificate expires which leads to the server sending a REJ message and starting a new handshake process with the client. This can be seen in Figure 2.2.

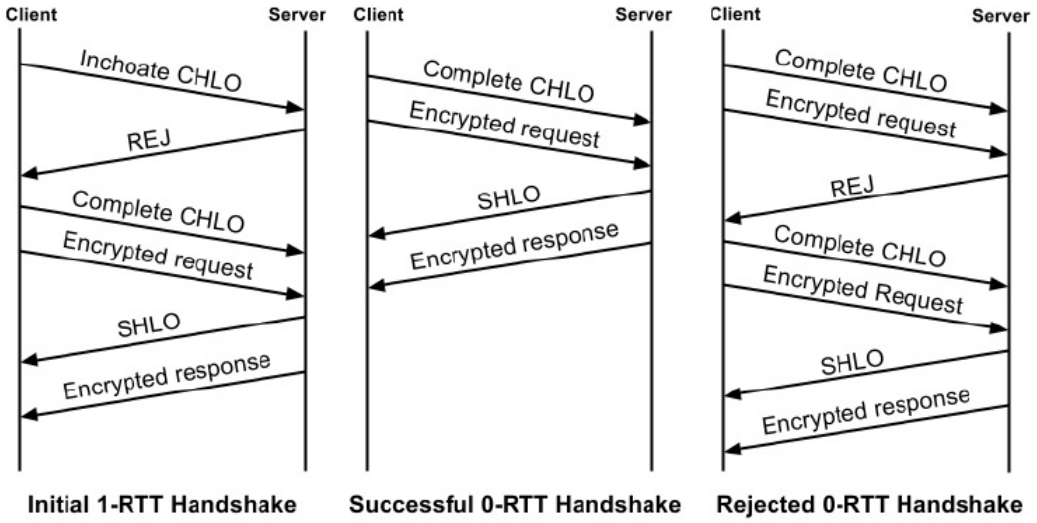


Figure 2.2: Connection Establishment - Timeline of QUICs Initial 1-RTT Handshake, a Subsequent Successful 0-RTT handshake, and a Failed 0-RTT Handshake [26]

A single QUIC packet can contain stream frames from various streams. This capability is provided by QUIC stream multiplexing. When sending data using QUIC, a packet can carry data from multiple streams and when a packet is lost, only the streams that had frames within that packet are affected. Every stream is distinguished by a stream ID, and this is statically given to every stream initiated by the client as odd IDs and for every stream initiated by the server as even IDs. This makes it easier to identify the streams and prevent collisions. [26].

The QUIC packet header contains a Packet number, version number, connection ID, and flags. Since every QUIC packet has a packet number, even the re-transmitted packets, this helps in the process of loss recovery as the packet number serves as a time ordering feature. It also assists in differentiating between the ACK of a re-transmission and ACK of an original transmission.

QUIC has its own acknowledgment packets, and they record the delay between sending the acknowledgement packet and the receipt of a packet. This also aids in accurately estimating the RTT with the help

of the packet number feature [26].

A stream can utilize the full receive-buffer size for a connection specified by flow control, which is considered a major issue [26]. This issue is mitigated in QUIC by limiting the buffer size for every single stream in a connection through the use of connection-level flow control and stream-level flow control. Connection-level flow control provides the sender with a limited total buffer size to be used across all streams in a connection, while stream-level flow control restricts the amount of buffer a sender can utilize for every stream [26]. Besides these two flow controls, QUIC uses credit-based flow control, which allows the receiver to broadcast an absolute byte offset for every stream and increases the offset as the data transfer goes for a specific stream using window update frames. This is used to inform the sender that more data can be sent on that stream.

The implementation of QUIC does not require the use of a specific congestion control algorithm and can be modified to use either CUBIC [49], NewReno [16], or BBR [7]. By default, CUBIC congestion control algorithm is enabled in QUIC's implementation. QUIC's implementation can be found on the Chromium project page [19], available as an open source project. QUIC is currently used in Google servers, YouTube and Chrome [26].

2.2.3 GridFTP

FTP is an IETF-standard protocol and has a well-defined code base which can be used for building extensions. GridFTP [1] is based on the File Transfer Protocol (FTP) [38] extended with additional enhancements for a grid-based environment, which is a network of computing nodes sharing each other resources. GridFTP features include Data striping, Parallel data transfer, Partial file transfer, Automatic negotiation of TCP buffer/window sizes, and Security.

One of the features of GridFTP is data transfer control by a third party. This is achieved by providing an interface to manage data transfers between servers by a third-party operation which includes initiating, monitoring and controlling a data transfer operation between two endpoints (a sender and receiver). The data transfers are secured using Generic Security Services (GSS) which provides an API authentication for the control channel (RFC 2228) and GridFTP extensions to the data channel. The authentication mechanism is crucial for identifying hosts that run third party data transfer to know if it is being sent to the right endpoint. This is because the IP address of the host connecting on the data channel is different from the one connected on the control channel [2]. Additionally, GridFTP provides data confidentiality and integrity using an interface controlled by a user.

GridFTP supports data striping by providing extensions that partition data among multiple endpoints. For parallel data transfer, GridFTP uses multiple TCP streams in parallel between two endpoints to improve the bandwidth utilization and achieves that by using FTP command extensions with data channel extensions [2]. GridFTP supports partial file transfers for applications that require specific parts of a massive file and supports requests for certain file regions. GridFTP also extends basic FTP features for re-initiating failed

data transfers providing reliable data transfers. Moreover, GridFTP can also be used with UDT instead of TCP to experiment with UDP [6].

2.2.4 FASP

The Fast and Secure Protocol (FASP) [47], a proprietary protocol from IBM, also aims to eliminate issues with TCP-based file transfers such as under utilizing link bandwidth and severely reducing data transfer rate when packet loss events occur. FASP aims to utilize the bandwidth as much as possible to increase the transfer rates regardless of network conditions and based on a target data rate specified by the configuration of the connection. FASP implements its own mechanism, which is not publicly available, to identify and re-transmit packets that are lost. It uses TCP to initiate the connection and UDP to transmit the data packets efficiently. FASP aims to achieve high file transfer speeds regardless of the file size, transfer distance or network conditions.

Moreover, the FASP protocol also introduces a security mechanism of its own for the file transfers. The security mechanism consists of Secure Shell (SSH) authentication, on-the-fly data encryption using strong cryptography (AES-128) for the transferred data, and an integrity verification per data block, to safeguard against man-in-the-middle and anonymous UDP attacks.

Furthermore, FASP deals with congestion by reducing its rate to allow TCP traffic to flow and it shares the link with TCP flows. This approach benefits TCP traffic by applying TCP-friendly rates when links are congested without stressing other TCP traffic. In addition, it tries to fully utilize the bandwidth regardless of latency and packet loss [47]. FASP protocol measures round trip times to derive an actual queuing delay on the network path [33]. The actual queueing delay is used as a parameter for the rate control mechanism in FASP. The data transfer rate is increased when the queuing delay is reduced. This is claimed to be a more efficient way of utilizing the network rather than looking at packet timeouts [33].

The FASP protocol facilitates data transfers between a sender and a receiver by configuring specific parameters. The sender is configured to use an injection rate for transmission of data and each block of data is identified by a sequence number [35]. Once the data block is received at the receiver side, the sequence number of every data block is checked to identify any lost data blocks. This is confirmed if a data block has a sequence number greater than the next sequence number, after which all lost data blocks are scheduled for retransmission at the receiver side by recording them in an index array. To send a retransmission request to the sender, path RTT is calculated by using a path RTT predictor [20]. Re-transmission requests are sent regularly at a similar injection rate until the lost data block is received from the sender. The sender re-transmits these data blocks at a rate similar to the injection rate in an attempt to reduce the amount of re-transmission requests stored at the sender side.

Bandwidth utilization between the sender and the receiver is independent of packet loss. The bandwidth usage is detected by using a hysteresis model [34]. If unused bandwidth is detected, FASP uses a *bandwidth utilization mode* which sets the injection rate to the equivalent of the unused bandwidth rate. The target

injection rate is determined by measuring parameters between a sender and a receiver. These parameters include queueing delay, RTT, and packet loss [35]. The queuing delay is calculated by measuring the difference between the base RTT and the measured RTT, which is used for determining when to use a bandwidth sharing mode. This also helps in determining if a network is congested, which readjusts the injection rate until the congestion is reduced [33]. Furthermore, the injection rate is recalculated during the data transfers depending on other ongoing flows on the same network path. This is achieved by measuring the base RTT, calculating the smoothed average RTT, and probability of packet loss.

2.2.5 LEDBAT

Low Extra Delay Background Transport (LEDBAT) [43] is a delay-based congestion control algorithm that seeks to utilize the available bandwidth while trying to reduce the queuing delay present on the network path. LEDBAT was designed for background data transfer applications and aims at being fair to standard TCP congestion control mechanisms and to limit its interference with other competing flows. LEDBAT implements one-way delay measurements to estimate the queueing delay and it uses the changes in the measurements to limit congestion that the flow creates in the network. It uses one-way delay instead of RTT to prevent unrelated traffic on the backward path from interfering with data transfer [43]. If the queueing delay is less than the anticipated queuing delay, then LEDBAT concludes that there is no congestion and increases its throughput (sending rate) to use the remaining capacity of the network. Similarly, if the queueing delay is greater than the anticipated delay, then LEDBAT reduces its throughput as a reaction to any available congestion. Therefore, it yields in the presence of competing TCP or UDP flows and responds to congestion earlier than a standard TCP congestion control mechanism [43].

LEDBAT manages the congestion window by using a controller, which adapts the sending rate to the queuing delay. It uses two parameters TARGET and GAIN, where TARGET refers to the maximum queuing delay that LEDBAT creates in the network and GAIN provides the congestion window (cwnd) with the rate at which it would react to any changes in queuing delay [43]. If the queuing delay decreases and is smaller than the target value, LEDBAT will increase its sending rate along with increasing its congestion window. The congestion window is increased to the value of the difference between the current queuing delay and the TARGET. When the estimated queuing delay increases more than the TARGET, the sending rate would be decreased. Also, the congestion window is decreased by LEDBATs controller to the value of the difference between the current queuing delay and the TARGET.

LEDBAT aims to be fair to TCP flows by setting its highest congestion window growth rate to be the same as the TCP's congestion window growth rate. In this case, LEDBAT does increase its sending rate more than TCP's sending rate running on the same network path. Moreover, LEDBAT does not rely on packet loss to determine its sending rate. However, LEDBAT reacts the same as TCP when packet loss occurs. It does that in attempt to avoid false queuing delay estimates; therefore, it halves its congestion window like TCP Reno [43]. To further limit LEDBAT's impact on other flows, the delay induced should be lower than 100 ms

which would work well with traffic that is delay-sensitive. A delay value greater than 100 ms could affect the one-way delay measurement process by exposing it to noise, and the bottleneck link would be underutilized due to less throughput used.

2.3 Related Work

2.3.1 TCP Variant Performance Evaluation

Measurement Techniques

Several studies used a simulation-based environment to conduct their experiments on TCP variants and congestion control mechanisms. Xue *et al.* [50] compared the performance of different variants of TCP using a 10 Gbps emulation-based environment called CRON, which emulates a high-speed optical network and they used *iperf*[2] as their traffic generator. They created a dumbbell topology, which consisted of three senders and three receivers connected to each other by a 10 Gbps bottlneck link with two routers and a delay node in between. In another study, Nguyen *et al.* [36] used a 6-ary fat tree topology in a simulation-based environemnt ns-3[3] to mimic the setup of a data center network. Similarly, Martin *et al.* [4] compared high-speed variants with TCP NewReno in a simulation-based environment *ns-2*,[4] where two senders and two receivers are connected by 1 Gbps bottleneck link. Sangtae *et al.* [41, 42] evaluated the performance of high-speed variants in an emulated network, which consisted of a dumbbell setup with two senders and two receivers with two traffic generators at each side connected over a 1 Gbps bottleneck link. They used two *dummynet*[5] routers (emulated routers) on the bottleneck link to control the bandwidth and the buffer size.

Besides using a simulaton-based environment, Martin *et al.* [4] also used a real testbed for their experiments to compare the performance of the TCP variants in both testing environments. The real testbed consisted of a dumbbell setup that had two senders and two receivers connected by a router on a 1 Gbps speed link. For generating traffic they used *iperf*, and they used *netem*[6] on the router to introduce delay to the transfer flows. Li *et al.* [30] also used a physical dumbbell setup which consisted of two senders and two receivers connected by a dummynet router on a 1 Gbps bottleneck link. Kumazoe *et al.* [25] also ran experiments using a server/client setup with two server machines and two client machines with varying RTTs connected over a 10 Gbps bottleneck link. Hock *et al.* [17] used a dumbbell setup that consisted of a sender and a receiver connected by DPDK-based software switches over a 1 Gbps and 10 Gbps bottleneck links. The DPDK-based software switch was used to emulate delay and control the buffer on the link. All these authors used *iperf* to generate traffic at particular rates to stress the network links.

[2]https://iperf.fr/
[3]https://www.nsnam.org/
[4]https://www.isi.edu/nsnam/ns/
[5]http://info.iet.unipi.it/ luigi/dummynet/
[6]http://www.linux-foundation.org/en/Net:Netem

Metrics and Results

Most of the studies presented compared the performance of TCP variants and congestion control mechanisms by various metrics. Nguyen *et al.* [36] evaluated the performance of multiple TCP control algorithms: NewReno, Vegas, High-Speed, Scalable TCP (STCP), Westwood+, BIC, CUBIC, and YeAH. They measured the queue length, throughput, packet delay, and packet loss for each protocol. They found that Vegas outperformed all the other TCP variants in terms of queue length, throughput, packet delay and packet loss. This is because of its delay-based mechanism which allows it to reduce the queue length with low queueing delay. On the other hand, their results also showed that CUBIC and Scalable TCP had the poorest performance among all the other TCP variants.

Sangtae *et al.* [41, 42] used a slightly different set of high-speed TCP variants: BIC, CUBIC, FAST, High-speed TCP, H-TCP, and STCP in their experiments. Their experiments examined the performance of the protocols in terms of fairness, link utilization, packet loss, and their behaviour with background traffic. In their first study, they found that none of the high-speed protocols performed well in terms of fairness with other flows when delay was increased. In terms of link utilization, they observed that when high-speed TCP protocols were run with background traffic, link utilization was improved. Furthermore, packet loss rates for H-TCP and STCP were the highest among all the protocols. However, STCP showed the least amount of packet loss when background traffic was induced. In their second study, they also found that H-TCP and STCP were not fair with other TCP flows because they both had higher link utilization rates than the other protocols.

Li *et al.* [30] evaluated the performance of the following TCP variants: STCP, high-speed TCP (HS-TCP), BIC-TCP, FAST TCP, and H-TCP. They found that FAST TCP, STCP, and Binary Increase Congestion control (BIC) TCP showed the most unfair behaviour with standard TCP flows. The results also showed that BIC TCP and High-speed TCP took a longer time to converge when a second flow was started.

As mentioned earlier, Martin *et al.* [4] compared the performance of several high-speed TCP variants with TCP NewReno in a simulation-based environment and in a real testbed. It was found that experiments in both environments provided similar results in terms of throughput used by each protocol. They also found that the high-speed TCP protocols had better fairness with TCP New Reno flows in the simulation environment, but less fair in the real testbed environment.

The performance of multiple TCP variants was also evaluated by Kumazoe *et al.* [25] with single and multiple flows. They found that all TCP variants had the highest throughput when using a single flow. For multiple flows, the throughput for each flow varied with time, which showed that the network bandwidth was not shared efficiently. Additionally, when H-TCP, CUBIC, and High-Speed TCP were used, the throughput fairness among flows was improved. Furthermore, they also observed that starting a new TCP flow after an existing TCP flow would cause a decrease in the throughput of the existing TCP flow. They found that these protocols can recover their original throughput rate after a new standard TCP RENO flow is initiated while TCP RENO flow cannot recover its original throughput rate.

Xue *et al.* [50] presented a comprehensive study of fairness for flows that consist of different variants of TCP, which are referred to as Heterogeneous TCP flows. These flows consisted of TCP variants such as TCP SACK, High-Speed TCP, and CUBIC were used to in their study. They investigated the fairness among the heterogeneous TCP flows by using different queue management schemes used by routers. They found that queue management schemes had a demonstrated effect on the performance of the TCP flows in terms of fairness. Moreover, they observed that RTT fairness does not improve when using heterogeneous flows. Overall, the results they obtained showed that heterogeneous TCP flows do not improve fairness and that fairness depends on queue management and buffer sizes.

Widmer *et al.* [48] presented a survey on various congestion control mechanisms used for protocols to achieve TCP-friendliness. The protocols were differentiated as single-rate and multi-rate along with specifying the type of congestion control mechanisms as window based or rate based. They evaluated these protocols based on their TCP-friendliness, which means that a protocol can be TCP friendly if it is fair to TCP flows when competing for bandwidth and the throughput over time varies less compared to TCP flows.

BBR congestion control mechanisms were examined by Hock *et al.* [48] in terms of throughput, packet loss, fairness, and queueing delay. They revealed that BBR works well for single a single flow, but not for multiple flows. They also compared BBR with CUBIC TCP and found that when multiple BBR flows are present with CUBIC TCP flows, BBR does not share the link fairly with CUBIC TCP. In terms of queuing delay, BBR did not respond to increased queuing delays properly, which caused high levels of packet loss. In addition, the authors also observed that BBR would still keep sending at high rates even though there is congestion and packet loss occurs.

Most of the experimental designs used in the above studies were either simulation-based or real testbed, but they were mostly using simulated network conditions. They also did not focus on analyzing only TCP variants for long distance transfers in the wild. Simulation-based and emulation-based environments may not provide realistic results about the performance of each protocol [13]. The results in previous studies will be extended in this thesis by comparing a set of protocols that includes some new protocols that have not been compared against each other before in real network conditions with varying transfer distances. Moreover, these studies provide an insight of what metrics are important to measure.

2.3.2 TCP/UDP Variant Performance Evaluation

In this Section, studies that compare the performance of TCP and UDP-based protocols together in various setups are presented. This also includes various congestion control mechanisms used with multiple data transfer protocols for short and long distance transfers.

Measurement Techniques

Most of the studies that combine TCP-based and UDP-based data transfer protocols used a network emulator in their setup to emulate network conditions, while relatively few studies relied on real network conditions

in their testbeds. Se-young *et al.* [51] compared the performance of high-speed transfer protocols in a local testbed environment with a 10 Gbps bottleneck link. They used *netem* to emulate various network conditions and they used *nuttcp*[7] to generate TCP or UDP background traffic with every 10 GB file size transfer for each protocol. The testbed also consisted of one sender and one receiver connected by two switches. In a similar setup, Yue *et al.* [54] also analyzed the performance of several UDP protocols in a 1 Gbps link testbed consisting of two servers, sender and receiver, with *netem* in between to emulate various network conditions. The traffic generated for each transfer consisted of different file sizes to observe the performance of each protocol at different scales. Kachan *et al.* [22] also used a testbed consisting of two servers connected by a switch with a network emulator, but using a 10 Gbps link. For their transfers, they used a 30GB file transfer on different RTT links. In a study on QUIC, Kakhki *et al.* [23] used a client running the Google Chrome browser and a server machine running an apache server and a QUIC server application, both of which were connected to each other by a router running a network emulator.

In later studies conducted by Se-young *et al.* [52, 53], they used a real testbed with a 10 Gbps international link that connected a sender located in Queenstown, NZ to a receiver located in Stockholm, Sweden. For their experiments, they used a 30 GB file for their transfers with each protocol. In a similar setup, Cottrell *et al.* [29] used a real testbed without any network emulator to compare the performance of several TCP and UDP data transfer protocols. This testbed consisted of two local sender machines and three receiver machines connected by a 1 Gbps link and distributed over various locations. They also used the same testbed over a 10 Gbps link. Traffic for their experiments was generated and measured using *iperf*. Suresh *et al.* [45] also used a real testbed over a 2 Gbps link on their campus network to compare the performance of TCP and UDP-based protocols.

Metrics and Results

Se-young *et al.* [51] compared the performance of several high-speed transfer protocols, namely GridFTP, Tsunami UDP protocol (uses UDP for high-speed file transfers over high bandwidth-delay product links), Fast Data Transfer (FDT) a TCP-based protocol for transferring files, and UDT in terms of their throughput usage with varying RTTs and in the presence of background traffic. They found that the TCP-based protocols performed better when there was no background traffic and when the path used had shorter RTTs. However, the TCP-based (GridFTP with TCP) protocols performed poorly when there was congestion induced by background traffic, unlike the UDP-based protocol (GridFTP with UDT) which provided a more stable performance. They also found that Tsunami was a better choice for links with high background traffic and longer RTTs. In addition, FDT and GridFTP with TCP had the highest throughput rate compared to that of the other protocols when there was no background traffic and with increasing RTTs. In their later studies [52, 53], they conducted experiments that focused on single flow analysis and multiple flows analysis along with measuring goodput for disk to disk and memory to memory transfers using each protocol. For single

[7] https://www.nuttcp.net/

flows, they found that GridFTP had the highest data transfer rate while FDT and UDT performed poorly because of implementation issues. FDT performed better with multiple flows than with a single flow.

Suresh *et al.* [45] analyzed the performance of GridFTP, Gridcopy, and UDT in terms of throughput, resource usage, file size and fairness to competing flows. Their experiments showed that the throughput rate used by GridFTP and GridCopy was higher when one large file was transferred instead of transferring a number of small file sizes. Additionally, GridFTP reduces its throughput rate when the number of connections used exceeds 100. The performance of UDT was the more stable than the other protocols because its throughput rate does not get affected by the file size and it provided a slightly better fairness performance than the other two protocols.

Other UDP-based protocols such as UDT, PA-UDP, RBUDP, and Tsunami were evaluated by Yue *et al.* [54] in terms of throughput, file size, packet loss, varying RTTs, fairness, and CPU usage. The experiments show that PA-UDP provided the best performance in terms of throughput for various file sizes, while RBUDP showed the worst performance since its throughput decreased with larger file sizes. Moreover, the performance of PA-UDP was more stable than other protocols when reacting to packet loss and varying round trip times. In terms of fairness, all the protocols showed good fairness with their own flows and with a TCP flow. Tsunami and PA-UDP showed higher CPU utilization than the other protocols on the sender side.

Cottrell *et al.* [29] examined the performance of STCP, FAST TCP, High-Speed TCP, H-TCP, BICTCP, Reno, and UDTv2. The performance of these protocols was compared in terms of throughput, fairness, utilization (CPU usage), and stability. The results of their experiments in the 1 Gbps network showed that BIC TCP, H-TCP, and STCP performed the best in terms of throughput achieved. Also, H-TCP and BIC TCP performed the best in terms of stability and fairness. UDT had a similar performance to the TCP implementation, but was more CPU intensive. On the 10 Gbps link, they found that UDTv2 was not able to achieve high link utilization, unlike TCP Reno which was able to nearly utilize the full link bandwidth.

Kachan *et al.* [22] also evaluated various types of TCP-based and UDP-based data transfer protocols. This work compared the performance of proprietary TCP-based and UDP-based solutions such as Velocity, Catapult server, ExpeDat, FileCatalyst Direct, and TIXstream. Velocity is a TCP-based file transfer application developed, and it aims at fully utilizing the available bandwidth. The Catapult Server is also a TCP-based tool and uses a client-server architecture. ExpeDat is a UDP-based data transport solution and uses the Multipurpose Transaction Protocol (MTP). Another UDP-based solution is FileCatalyst and has its packet loss management, rate and congestion control mechanisms. Like FileCatalyst, TIXstream also uses UDP and utilizes only one UDP socket on the sender and receiver.

The metrics they measured in their experiments were data transfer rate and transfer duration. The results of these experiments showed that the UDP-based protocols performed better than the TCP-based solutions in terms of link utilization. Among these solutions, TIXstream had the highest link utilization and FC direct had the most stable performance. From this work, a similar set of metric measurements can be

used to examine the performance of the candidate protocols in this thesis.

Kakhki *et al.* [23] evaluated QUIC performance and compared it with TCP in terms of page load times, fairness, video QoE, packet reordering, and proxying. They conducted experiments focused on loading webpages with images with various sizes from the chrome browser and using Youtube content. After running their experiments, they found that QUIC outperforms TCP in every scenario for the desktop environment. However, QUIC consumed more bandwidth than its fair share; thus, it was unfair to competing TCP flows. In addition, QUIC does not perform well in the presence of packet reordering as it considers such behaviour as packet loss.

2.4 Chapter Summary

In this chapter, most of the mentioned studies evaluated and compared the performance of different TCP variants and data transfer protocols using simulation and real testbed environments. Some of these studies relied on simulation-based networks or network emulators to emulate various network conditions rather than experimenting with real network environments. This would not provide a realistic performance of every protocol being used; however, protocols used in a real-world environment would provide a better understanding of the protocol's overall performance at the expense of understanding the particular effects of more controlled experiments. To validate this, a side-by-side comparison of existing or under development data transfer protocols and congestion control algorithms in a real network setup is required.

In this thesis, previous studies are extended by the experiments conducted. The candidate protocols in this thesis were chosen for three reasons. The first reason is that these protocols are developed by major companies such as Google and IBM and they are also widely used in in the internet for file transfers. The second reason is because some of these protocols were not evaluated much in the literature or were not examined side-by-side before. The third reason is that not all the protocols mentioned in this thesis had robust implementations, and were modified in specific research labs.

The behaviour of each protocol was analyzed with background traffic and with competing flows to observe their fairness. It is essential that each protocol does not negatively impact other users' traffic, which shares the same network path. Other protocols and congestion control mechanisms such as MDTM (Multicore Aware Data Transfer) [55], TCP Lola [18], and PCC [9] are not included in this thesis, because they were either focused on a specific network setup or did not have a usable implementation. Other Protocols mentioned earlier were not used in this thesis, because some of the ideas in these other protocols are incorporated in BBR, QUIC, GridFTP, and FASP. Also, the implementation of these other protocols is widely available or easily configurable. To make this study more manageable, the number of protocols and congestion control mechanisms used were limited.

Chapter 3

Experimental Design and Configuration

This chapter describes the experimental design, configuration parameters for each protocol used, and the experiments used to measure and compare the performance of each protocol. Section 3.1 describes the hardware configuration and the testbed environment used. Section 3.2 discusses the configuration of each protocol used in the experiments. Section 3.3 contains the experimental tools and the performance metrics used in the experiments. Furthermore, a description of every experiment is provided in Section 3.4.

3.1 Hardware Setup and Configuration

To observe the performance of each protocol and congestion control, local, national, and international testbeds were set up. Both machines are connected to each other with a 1 Gbps Ethernet link in a server/client architecture. The Internet is used directly to provide realistic network conditions and results; therefore, no devices were used for simulation or emulation. The sender role in all of the testbed environments is to transfer tar files of collections of crop images of various sizes.

3.1.1 Sender Machine

The physical machine on the sender side had two virtual machines sharing the same network card. The first virtual machine had FASP, GridFTP and QUIC installed and configured on it and the second virtual machine was configured with BBR. The reason for having two virtual machines was that the BBR congestion control module was not available in the TCP stack of the first VM's Linux kernel as BBR is only available in Linux kernel version 4.9 or above. In addition, the other protocols were already installed and configured on the first virtual machine; therefore, a second virtual machine was created with Linux version 4.10 to run the experiments with TCP BBR. These virtual machines were also used as the sender in the National and the International testbed.

3.1.2 Local Testbed

The local testbed consists of two Linux virtual machines on separate physical machines, a sender and a receiver, located on the campus of the University of Saskatchewan connected by a 1 Gbps Ethernet link

as shown in Figure 3.1. Traceroute showed that the RTT between those machines is 0 ms and traffic goes through one hop. Details of the virtual machines used in this network are shown in Table 3.1.

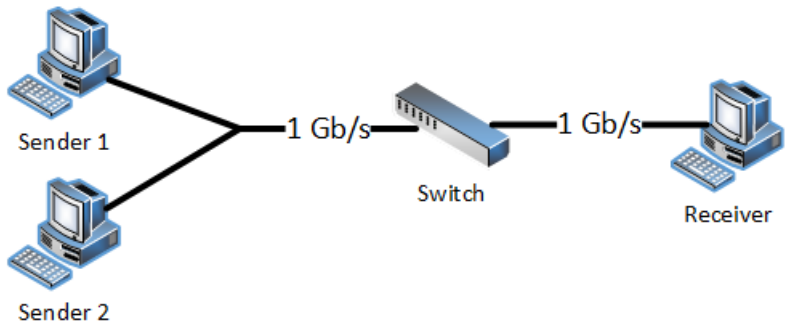


Figure 3.1: Local Testbed

Table 3.1: Virtual Machines Used in Local Testbed

Role	CPU	Memory	NIC	OS
Sender (VM1)	Intel Xeon E312xx (Sandy Bridge) 3.4 GHz	12 GB	Red Hat Virtio network device	Linux 4.4.0-109-generic
Sender (VM2)	Intel Xeon E312xx (Sandy Bridge) 3.4 GHz	4 GB	Red Hat Virtio network device	Linux 4.10.6-generic
Receiver	Intel Core Processor (Haswell) 2.29 GHz	4 GB	Red Hat Virtio network device	Linux 4.4.0-108-generic

3.1.3 National Testbed

In collaboration with the University of Waterloo, a receiver machine was used to act as the destination for transfers between Saskatoon, SK, Canada and Waterloo, ON, Canada. The path between the sender and the receiver contains 18 hops with an average RTT of 34.49 ms, which was obtained from traceroute. A second receiver machine was set up using Compute Canada,[1] which provides advanced research computing resources shared among various research organizations. The particular Compute Canada virtual machine was created on a cloud platform called Eastcloud. This machine happens to also be in Waterloo, but uses a different network path with 19 hops and an average RTT of 40 ms. On both of the Waterloo and Eastcloud setups, the packets go through different routers as they get closer to their destination. The national testbed is shown in Figure 3.2 and the hardware used for each is shown in Table 3.2.

Table 3.2: Virtual Machines Used in National Testbed

Role	CPU	Memory	NIC	OS
Receiver 1	Six-Core AMD Opteron(tm) Processor 2439 SE 2.6GHZ	32 GB	nVidia MCP55 Ethernet	Linux 4.4.0-92-generic
Receiver 2	Intel Xeon E3-12xx v2 2.6GHZ	7 GB	Red Hat Virtio network	Linux 4.4.0-109-generic

[1] https:/www.computecanada.ca/

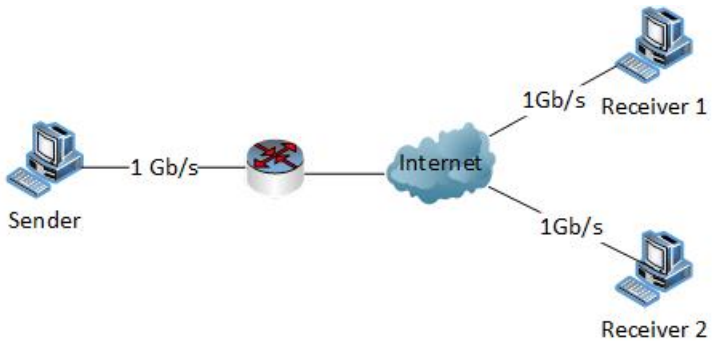


Figure 3.2: National Testbed

3.1.4 International Testbed

In collaboration with the University of Auckland in New Zealand, an international testbed was deployed between Saskatoon, SK, Canada and Auckland, New Zealand. Each machine has a 1 Gbps connection to its local network. The speed of the link from each network to the other is unknown. The path between the two machines contains 15 hops and has an average RTT of 182 ms when not running experiments. Table 3.3 shows the machine specifications of the receiver used in this testbed.

Table 3.3: Virtual Machine Used in International Testbed

Role	CPU	Memory	NIC	OS
Receiver	X5650 Intel(R) Xeon(R) processor 2.67GHz	24 GB	Intel 82576 Gigabit Ethernet	CentOS 6.8

3.2 Protocol Configuration

For every machine used in the experimental setup, FASP, GridFTP, QUIC and congestion control implementations for TCP BBR and TCP LEDBAT were installed and configured. Each machine was also setup to use CUBIC as its default congestion control algorithm for GridFTP and QUIC. The implementation of each data transfer protocol and congestion control was different from each other and specific steps were followed to install them on the machines used in the experiments.

- **FASP**: The implementation of FASP involved downloading and installing the Aspera point-to-point software with trial licenses on every machine, which was provided by the Aspera team[2] since it is a proprietary tool. To permit a transfer between any two machines, the software has to be installed on both machines. The software provides a peer to peer connection, where any device can act as the sender or the receiver. For transferring data, one uses a command similar to Linux's "scp", called "ascp". For parameter setting, there is not much to be configured; the main options that can be set is

[2]http://downloads.asperasoft.com/en/downloads/7

the target transfer rate. The target transfer rate was set to 600 Mbps for all the experiments, because after testing several target transfer rates, this was the optimal target transfer rate for a single flow. This target transfer rate was chosen, because in most the test transfers this was the highest transfer rate reached by FASP. Since FASP is a commercial solution, a lot can be inferred by examining its behaviour, but the internal mechanisms used would be unknown.

- **GridFTP:** The configuration of GridFTP was achieved by downloading and installing the Globus connect server package,[3] which can act as a server or an endpoint to send or receive data. To use this software, Globus credentials had to be created through Compute Canada. Furthermore, the functionality of GridFTP is embedded within this software and parameters can be set. The Globus server package has to be installed on the sender and the receiver machine before initiating any transfers between the two machines. In addition, GridFTP's parallelism feature was set to the default depending on the network conditions and the type of Globus subscription used. Parallelism was set by default to 4 parallel streams as the preferred number of streams and a maximum of 8 parallel streams. Thus, GridFTP uses 4 streams for every file transfer.

- **LEDBAT:** was implemented on the sender machine only and installed as a module in the linux kernel. The LEDBAT source code,[4] was retrieved from GitHub and modified to be compatible with the Linux kernel version used in the experimental setup. After the installation process, data transfers can be started by using the Linux **"scp"** command.

- **BBR:** Similar to LEDBAT, BBR was also configured on the sender virtual machine only. BBR was configured in the Linux 4.10 kernel TCP stack[5] as the default congestion control for all outgoing TCP traffic. Moreover, fair queuing was enabled in BBR and was configured to use TCP as its transport layer protocol. Transfers were initiated by using the **"scp"** Linux utility.

- **QUIC:** The setup of the QUIC protocol included building the QUIC source code (version 25) from the Chromium project[6] on every machine used in the experiments. At first, a standalone QUIC server and client[7] was used, which was found in an alternative implementation of QUIC on GitHub. This strips down unnecessary dependencies available in the chromium project and makes it easier to build the code. After the code is built, server and client programs are started on the sender and receiver machines respectively to initiate any transfers. The server application is kept running in the background and has several UDP ports in the listening state, waiting for a data transfer request from the client. To initiate a transfer, the server and client are started with certain parameters such as specifying the file

[3] https://docs.globus.org/globus-connect-server-installation-guide/
[4] https://github.com/silviov/TCP-LEDBAT
[5] https://github.com/google/bbr/blob/master/Documentation/bbr-quick-start.md
[6] http://www.chromium.org/developers/how-tos/get-the-code
[7] https://github.com/google/proto-quic

to be sent and the receivers host name or IP address to initiate a transfer.[8] In terms of the congestion control used, QUIC was setup with the default CUBIC congestion control mechanism.

This implementation of QUIC had several drawbacks. Data transfers initiated by the server and client were configured to use the machine's memory rather than using disk to read and write data. This caused the server and client to crash when the size of the file increased above 500 MB and the machine's memory was not able to handle it anymore. Furthermore, every file that is to be transferred using this implementation had to be embedded with an HTTP header because the server hosts the files on a dummy website link and the client requests for a certain file using that link. With the help of colleague Carl Hofmeister, the implementation of the QUIC server application was modified to read the data directly off disk into chunks and send the data as chunks over the network to avoid the memory limitation for large data sizes. Between every chunk sent, there is a small interval of time (approx. 50 ms) where the network is not being used. This small amount of time involves a new chunk being setup using one thread and another thread to send it over the network. The implementation was also modified to allow files to be transferred without the need of HTTP headers.

3.3 Experimental Tools and Performance Metrics

This section introduces all of the experimental tools used in the experiments: Wireshark, Bash Shell Scripts, iperf3, dd, top, and matplotlib. Also, the performance metrics measured for each protocol are described and justified.

3.3.1 Experimental Tools

- **Wireshark**[9] is an open source packet analyzer used for capturing and analyzing packets produced by protocols running data transfers on a network interface. Wireshark contains a utility called *dumpcap*, which is used to record packet headers or packet data into a file. Statistics about the transfers, such as the number of number of bytes sent, elapsed time, and number of packets were computed by tshark, the command line interface of Wireshark. AWK scripts were used to extract the data from the fields displayed by *tshark* to obtain the results of every capture file.

- **Bash Shell Scripts** were used to automate the experiments and to extract the data from capture files by running a collection of commands along with constructing loops and functions to suit every scenario.

- **Iperf3**[10] is an open source utility used as a traffic generator and could also be used as a bandwidth measurement utility. It has a client and server functionality, and can be used to generate TCP or UDP traffic between two endpoints.

[8] https://www.chromium.org/quic/playing-with-quic
[9] https://www.wireshark.org/
[10] https://iperf.fr/

- **dd** is a command-line utility used mainly used in UNIX-based operating systems to copy files from one location to another on the hard disk. It is a useful tool to test the disk read/write speed to ensure that there is no limitation induced by the hard disk when reading a file and transferring it over the network.
- **top** provides the user with information about every process running on the machine, specifically CPU and memory utilization. It is used to identify CPU consumption by every process or program.
- **Matplotlib** is a python library used in PyCharm,[11] an integrated development environment, to plot graphs for the results extracted from the capture files. The results were extracted into CSV format files and PyCharm was used to process and plot the data points accordingly.

3.3.2 Performance Metrics

- **Goodput** is described as the **Useful** data (bits) transferred over a link in a given amount of time. The useful data means the original data sent excluding any re-transmission packets or protocol overhead such as frame headers and other data wrapped around application data. Therefore, goodput is always expected to be lower than the throughput (the rate at which data is transferred over a network link) and the bandwidth (network connection speed). Packets may be dropped or corrupted in the network or at the receiver, so only part of the network path from source to destination deals with these undelivered packets. Moreover, goodput provides a useful and clear comparison between the different file sizes used in every transfer.

  In all the experiments, the time taken to complete a data transfer was measured and this time measurement is used to compute the goodput (3.1).

$$Goodput = \frac{file\ size\ in\ Mbits}{file\ transfer\ time} \tag{3.1}$$

- **Packet loss** is the percentage of packets that fail to reach their destination. Packets can be lost due to network congestion and router policies on a network link. In the event of network congestion, packets are dropped because the packets are being sent at a rate that is greater than the rate that a network segment can accommodate.

  To measure packet loss for TCP-based protocols, the total number of re-transmission packets was obtained for every destination's packet capture file using the "tcp.analysis.retransmission" filter in Wireshark. Then, the estimated number of re-transmission packets was divided by the total number of packets sent and multiplied by 100 to obtain the percentage of packet loss for each experiment (3.2).

  The reason for choosing the number of re-transmission packets as a measure of packet loss was because the packets received at the destination were possibly fragmented by IP fragmentation or the TCP

[11] https://www.jetbrains.com/pycharm/

segmentation offload mechanism (TSO), thus a very high number of packets was received by the destination machine. The IP fragmentation mechanism is responsible for breaking IP packets into fragments, so that small sized packets pass through the link. Once these fragments reach the destination, they are reassembled by the receiving host. On the other hand, the TCP segmentation offload mechanism breaks down large chunks of data into smaller segments for TCP packets. This process is handled by the network card on the sending machine, and the segments do not get reassembled when received by the destination machine. Therefore, TSO is possibly causing the destination machine to receive a very high number of packets compared to the number of packets sent. Consequently, comparing the total number of packets sent and received would not be ideal for measuring packet loss and it would not an easy task to reassemble all the fragmented packets.

$$TCP\ Packet\ Loss\% = \frac{Retransmission\ Packets}{Packets\ sent} \times 100. \tag{3.2}$$

Unlike TCP-based transfers, UDP packets were not fragmented when reaching the receiver. Therefore, packet loss was measured by obtaining the total number of packets sent on the sender side and the total number of packets received on the receiver side. The difference between both the number of packets sent and the number of packets received is obtained and is divided by the number of packets sent multiplied by 100 in order to obtain the packet loss percentage (3.3). The best way to obtain packet loss is by measuring bytes instead of packets. However, this involved using more complex analysis scripts, which were not straight forward to implement.

$$UDP\ Packet\ loss\% = \frac{Packets\ sent\ -\ Packets\ received}{Packets\ sent} \times 100. \tag{3.3}$$

- **CPU Usage** is a key performance metric in determining how much CPU does each data transfer protocol use when initiating a data transfer. CPU utilization is measured as a percentage of CPU being used by a certain process of program. The CPU percentage was measured using *top*.

- **Fairness** is a measure to determine if protocols are receiving their fair share of network bandwidth. Most protocols achieve fairness by being equally aggressive. This implies that multiple flows can fairly share the bottleneck link by obtaining approximately equal shares of the link capacity. On the other hand, if a protocol such as TCP LEDBAT is sharing the bottleneck link with other flows, LEDBAT reduces its sending rate so that it does not add extra delay or impact the performance of the other flows.

3.4 Experiments

In this section, a description of the all the experiments conducted is presented to compare the performance of GridFTP, FASP, QUIC, TCP BBR, and TCP LEDBAT in various scenarios. Each experimental configuration

was repeated twenty-three times for each protocol for 11.5 days with each experiment being run twice a day. One run was conducted on the first day for testing purposes, and was included in the results, then each experiment was run twice a day for the remaining days. Each set of experiments took between 7 and 9 hours to complete in a day. The experiments are based on the research questions in Table 1.1 listed in Chapter 1.

3.4.1 Experiments with Various File Sizes And RTTs

In this set of experiments, each protocol was used to transfer three tar file sizes (200 MB, 1 GB, and 5 GB) to all the locations mentioned in Section 3.1 with varying RTTs over a 1 Gbps output link. These files are sent to /dev/null on the receiver side to avoid disk overheads. Wireshark was started on the sender side and the receiver side to capture the data packets sent and received.

3.4.2 Experiments with TCP and UDP Background Traffic

In this set of experiments, each protocol was used to transfer a 1 GB tar file size with TCP and UDP traffic generated by *iperf* as background traffic. Each protocol was used with each type of background traffic separately.

For each type of traffic, iperf was run in client mode to send traffic from the sender machine and in server mode to receive traffic on the receiver machine. Once iperf starts generating traffic, a transfer for one protocol at a time is started. Additionally, different settings were used to generate TCP and UDP background traffic, which included setting the sending rate and the size of data to be sent. For TCP background traffic, the bandwidth was set to 200 Mbps and the default settings for the background traffic duration and TCP window size were used.

For UDP background traffic, the bandwidth was set between 200 Mbps and 300 Mbps for each testbed. Since UDP uses datagrams, the data size was set between 5 GB and 7 GB depending on how long each protocol takes to send a file to the destination machine. This allowed each protocol to complete a file transfer while background traffic was still running in the background. In terms of measurements, goodput and packet loss were measured for each protocol along with the goodput and packet loss of the TCP and UDP background traffic.

3.4.3 Experiments with Multiple Flows

This experiment investigates the performance of protocols when multiple flows of the same protocol are running in parallel, which can also be referred to as intra-protocol fairness. More specifically, it shows how each protocol behaves when there is more than one flow of the same protocol running at the same at time and competing for the bandwidth of the link. This experiment involves running parallel data transfers from one server to multiple receivers using a 1 GB tar file in 2, 4, 6 flows for each protocol over a 1 Gbps link. The aggregate goodput and packet loss were measured for each type of flow using each protocol in all the

testbed environments. Since each testbed has a different RTT, this will show us how the distance affects the performance of the protocols with multiple flows.

3.4.4 Experiments with Competing Traffic

In this experiment, the candidate protocols are tested by running a protocol against another in parallel to understand the effect of competing traffic, which is referred to as inter-protocol fairness. This experiment involves running a single flow of a protocol against a flow of another protocol using a 1 GB tar file from one server to multiple receivers in all the testbeds. It is essential to understand the behaviour of protocols in such a situation, because there are various network applications that use different data transfer protocols to transfer data.

3.4.5 Measuring CPU Utilization

It is important to use a protocol that uses the CPU efficiently to avoid using up the CPU before reaching the optimum data transfer rate [14]. In this experiment, the efficiency of each protocol was measured by observing the percentage of CPU utilization in user mode, which is important when transferring large data sizes at high speeds. The average CPU usage was measured for a 1 GB file transfer. The program *top* is used on both the sender and receiver machine to collect the CPU usage for each protocol during a file transfer. The CPU % was recorded for every one second during a transfer using a script and the average CPU % was calculated.

Chapter 4

Experimental Results and Discussion

For experiments 1-4, each experiment was run twenty-three times for each testbed scenario. For all the experiments, the first quartile (Q1), mean, median, third quartile(Q3) values are shown. Also, the goodput and packet loss were calculated for each protocol. These values are presented in a box plot graph. The box plot graph also shows the whiskers that extend from the edges of the box to show the range of the data points. The position of the lower whisker was set to the 5th percentile and the upper whisker to the 95th percentile of the data. Any points that are past the whiskers are considered outliers. In addition, the values of each metric for every instance of each protocol are presented individually in bar graphs wherever necessary for further clarification.

In Section 4.1, the results of the first group of experiments. Section 4.2 describes the results of the second group of experiments, which consists of transfers with TCP and UDP background traffic. Section 4.3 discusses the results of experiments with multiple flows. Lastly, Section 4.4 shows the results of a single flow of each protocol competing against single flows of the other protocols along with the CPU usage for each protocol is discussed.

Data transfer protocols can be affected by a) protocol overhead in the network stack on sender or receiver, b) the network physical elements between the two endpoints, and c) disk bandwidth or CPU limitations. The comparison between the protocols over various distances with various bandwidth limits can verify how robust they are with respect to performance characteristics. It also shows how each protocol's mechanisms react to various network conditions. In a network that is a blackbox, there would be many hops such as routers and firewalls that enforce policies and rules that can limit or affect the performance of protocols during a data transfer. However, the specific effects cannot be associated to any device, only emergent characteristics can be inferred. Additionally, the disk bandwidth on the sender machine can affect the speed at which bytes are being read from the disk and the time it takes to get these bytes over to the network interface.

To determine if the disk bandwidth limitation would affect each protocol's performance, the disk bandwidth is measured using the *dd* Linux utility in Table 4.1. This includes measuring the bandwidth used and time taken to transfer a file from one location to another. The results for the various file sizes show that disk bandwidth is less than the network interface limitation, which is the speed at which the data is being sent on the network. In order for any network comparisons to be relevant, the disk bandwidth must be greater than the network bandwidth. Otherwise, the experiments only compare the sending rate of the disk system

and are not related to network conditions.

Table 4.1: Measuring Disk Bandwidth Using dd Linux Utility

Configuration	**200 MB**	**1 GB**	**5 GB**
Bandwidth (Mbps)	891.2	846.4	741.6
Time (s)	2.074	13.98	56.77

4.1 Results for the first Group of Experiments

The first series of the experiments measures the goodput and packet loss for each protocol when transferring 200 MB, 1 GB, and 5 GB file sizes in every testbed scenario to observe the performance of the protocols over various distances and round trip times. A 10 GB file size was used in the initial results for a single run, but was changed to 5 GB for multiple runs in order to be able to run all the experiments twice a day. Each set of experiments took between 7 and 9 hours to complete. This varied depending on various factors: namely, network traffic not imposed by the experiments and potentially receiver machine activity. These factors were not within the control of the experiment, hence the need for substantial replication over varying periods of time.

4.1.1 Goodput Measurements for the Initial Experiments

The goodput results for each protocol in Local, Waterloo, and Auckland setup are shown for a single run in Figures 4.1, 4.2, and 4.3. FASP and GridFTP showed consistently better performance than the other protocols in all the setups. TCP BBR had high goodput in the local setup, but its goodput decreased in longer distance setups because of the high Round trip times and potentially increased congestion.

The performance of QUIC is rather consistent in all the setups and it shows slightly reduced goodput in longer distance setups. This could be because of its implementation, which sends a chunk of data at a time.

The fact that most of the protocols showed high goodput values in the Local setup compared to the Waterloo and Auckland setup verifies that they perform better on hosts that are geographically close to each other. The other observation is that some protocols are still able to achieve a high goodput as the distance increases.

The results for LEDBAT are particularly disappointing and could not be reproduced with any consistency. It appeared that there was an implementation bug that could not be easily diagnosed, so the remainder of the experiments do not include LEDBAT. The transfer would take substantially longer time to complete, sometimes longer than all the other protocols combined.

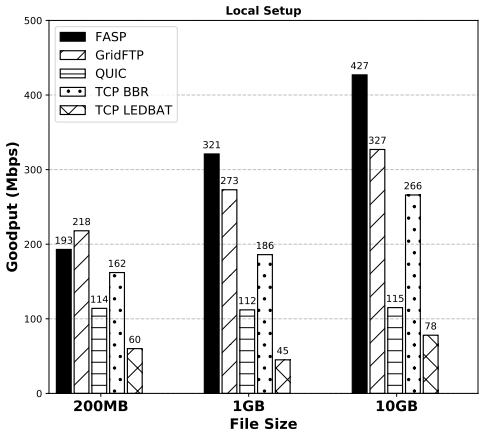


Figure 4.1: Goodput for Each Protocol in Local Setup

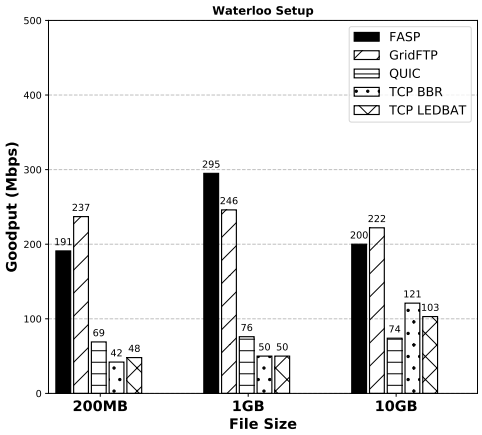


Figure 4.2: Goodput for Each Protocol in Waterloo Setup

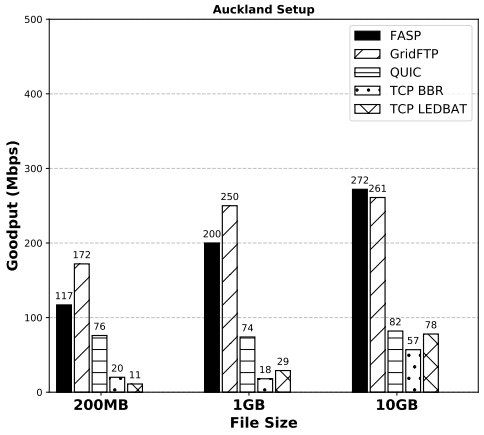


Figure 4.3: Goodput for Each Protocol in Auckland Setup

4.1.2 Extended Measurements - Goodput and Packet Loss

As seen in the initial results, the performance of all the protocols for a single run varied depending on the file size and the distance, especially for TCP BBR and TCP LEDBAT. After conducting a second run of the experiments, there were differences found between some of the protocols that were substantial, which led to an assumption that there would be experimental error in the factors that could not be controlled. Thus, experiments were repeated over a period of 2 weeks to potentially observe any variation in the performance of the protocols depending on the day of the week and the time of day.

Local Testbed Results for Single File Transfers

In Figure 4.4, TCP BBR has the highest variation in the goodput results for all the runs among all the protocols. BBR also had the highest goodput for 75 percent of the runs compared to the other protocols. However, the results for all the twenty-three runs varied significantly, which is a sign of instability in the

results. This is because BBR calculates the RTT and the bottleneck bandwidth several times over the life of a connection to determine if there is delay and it will either increase or decrease its sending rate. GridFTP did not show a high variation in the results as TCP BBR for the twenty-three runs, but had a similar median to TCP BBR for the 200 MB file transfer. This shows that GridFTP had more stable results than TCP BBR. For larger file sizes, BBR clearly achieves high goodput results compared to the other protocols. For smaller file sizes, it is not possible to determine which protocol is the best, because each protocol takes a very short time to transfer the file and it's not enough time to observe any differences in longer term time scales.

FASP had a low goodput compared to GridFTP and TCP BBR in this scenario, especially for 200 MB and 1 GB file transfers. This is because it suffered from high packet loss as observed in Figure 4.5. However, FASP achieved similar goodput results for the all the twenty-three runs of this experiment, which shows that the results were more stable than the other protocols. QUIC also showed a similar behaviour between the runs, but the results were even more consistent. In terms of outliers, GridFTP and TCP BBR had more outliers than the other protocols. This shows that their performance varies depending on the iteration number and that the network or host conditions influence performance.

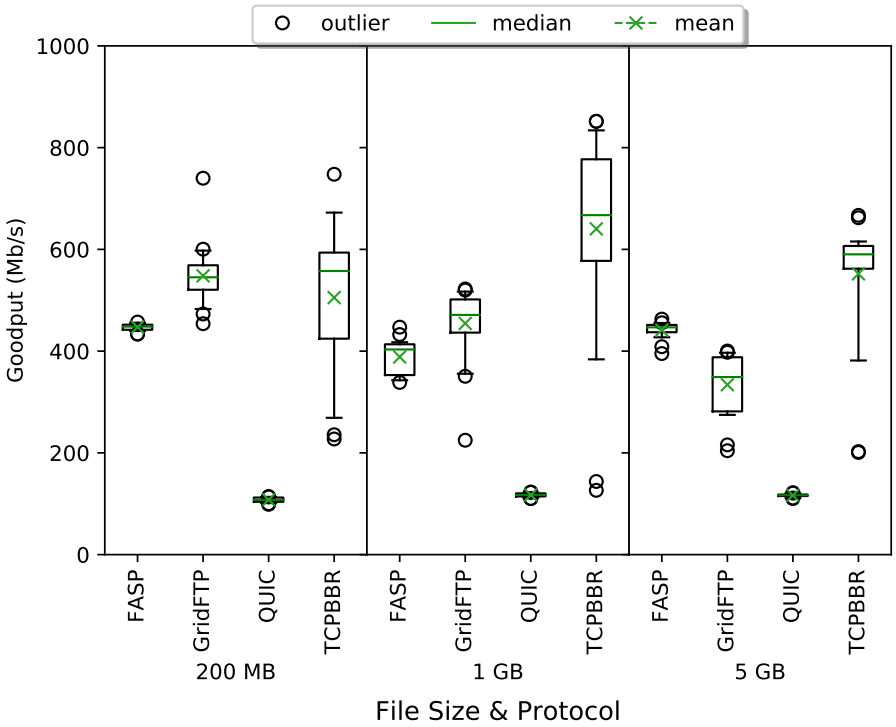


Figure 4.4: Aggregate Goodput for Each Protocol with Various File Sizes - Local setup

In terms of packet loss, Figure 4.5 shows the mean, median, Q1 and Q3 values of all the runs for each protocol in this experiment. Most of the protocols have shown close to zero packet loss in this setup except for FASP, which had a high packet loss for the 1 GB and 5 GB transfer scenario. This is potentially caused by the destination machine, which has a high number of users using that machine. This causes a reduction

in the system's processing speed and congestion on the network interface.

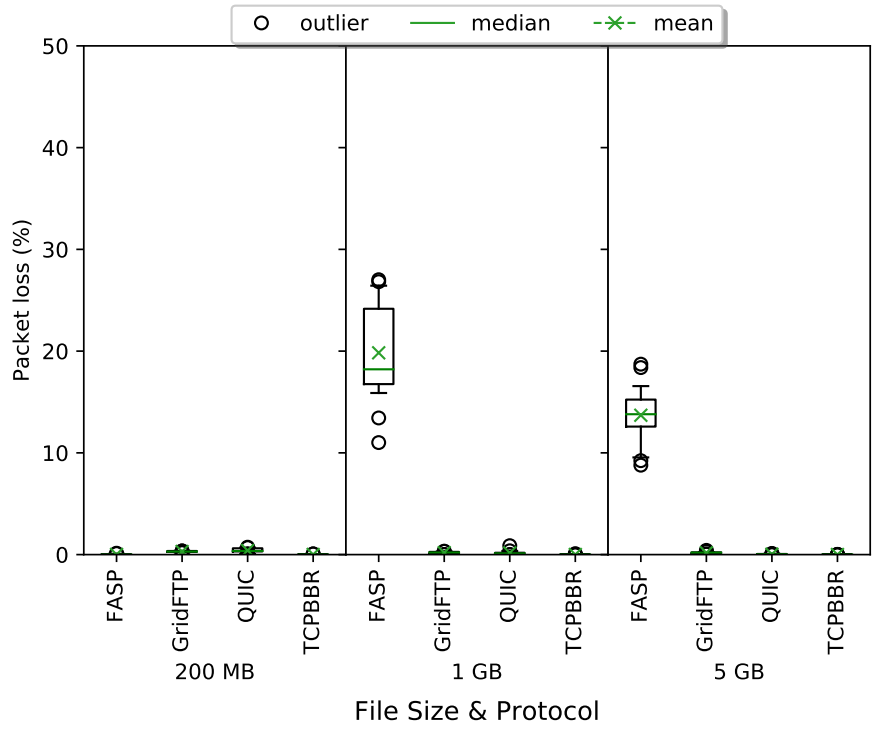


Figure 4.5: Aggregate Packet Loss for Each Protocol with Various File Sizes - Local setup

The results of each replication for protocols that showed inconsistent performance are shown in Figure 4.6 and 4.7. The main observation here is that GridFTP and TCP BBR showed a greater difference in the results between the replications than the other protocols. This may be related to the different amounts of congestion and delay experienced by these protocols each day. The packet loss for FASP varied for all the runs, especially for the 1 GB file transfer. The goodput results for FASP were more consistent between the runs.

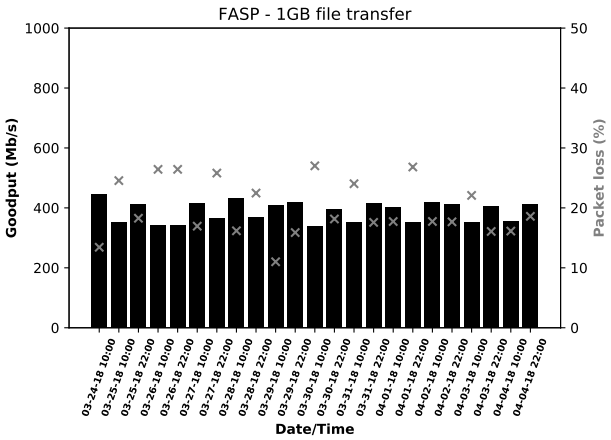
In some replications, TCP BBR has the same packet loss for high and low goodput values. This is because BBR is not as affected by packet loss, but is affected by delay or long RTTs. The low goodput means that BBR used a low sending rate, which is caused by delay on the network path and potentially caused by the receiver machine as the propagation time is minimal. Both of these protocols have the highest goodput among all protocols, and their packet loss is fairly low, and thus they do not have any significant impact on the data sent. As shown in Figure 4.6, the goodput of FASP was similar for most of the replications, but the 1 GB file transfer had a lower goodput than the 5 GB file transfer.

In terms of packet loss, FASP had the highest packet loss among all the protocols for all the runs, especially for the 1 GB and 5 GB file sizes. The packet loss for the 1 GB file transfer is higher than the 5 GB file transfer and this is potentially related to the varying amount of congestion caused by traffic going to the receiver machine. In Figure 4.8, the sending rate over time for FASP is shown for a sample run of the 1

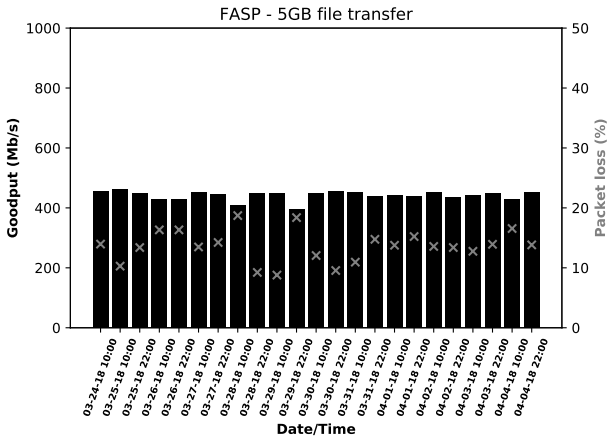
GB and 5 GB file sizes. This shows that FASP achieves a similar sending rate for the 1 GB file and the 5 GB file with the 5 GB file transfer having a higher sending rate. FASP experiences a few drops in its sending rate for the 5 GB file transfer due to variable network conditions.

Figure 4.9 shows the sending rate over time for TCP BBR for the 1 GB file transfer for replications 16 and 22. In replication 16, TCP BBR has a reduced sending rate compared to that of replication 22 and it undergoes congestion avoidance throughout the transfer. This implies that there is high delay on the network at the time of the transfer, which reduces the performance of BBR. Also, this shows that TCP BBR is not able to probe for more bandwidth in that replication and the sending rate does not increase. The activity on the destination machine may have affected the return of TCP BBR's ACK packets as claimed by the tech. However, replication 22 experiences a higher packet loss than that of replication 16, but it does not severely affect the sending rate of TCP BBR since it is delay-based. Therefore, it keeps a high sending rate compared to replication 16. QUIC had more consistent results than the other protocols, therefore it is not shown.

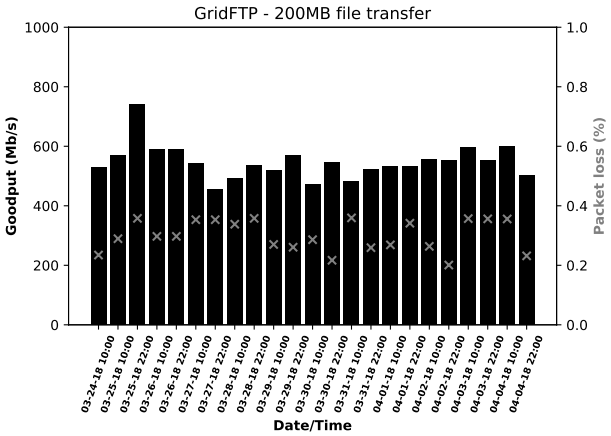
As seen in Figure 4.4, the goodput of GridFTP decreases as the file size increases in this setup, which would imply that the sending rate is low. To validate this, Figure 4.10 shows the sending rate over time for sample run of GridFTP for each file transfer. It can be seen that GridFTP is experiencing a high number of drops in its sending rate at different times during its transfer, especially for the 1 GB and 5 GB file size. This confirms that GridFTP takes a long time to get out of slow start and is unable to sustain the same sending rate through out the transfer, thus the goodput is also affected.



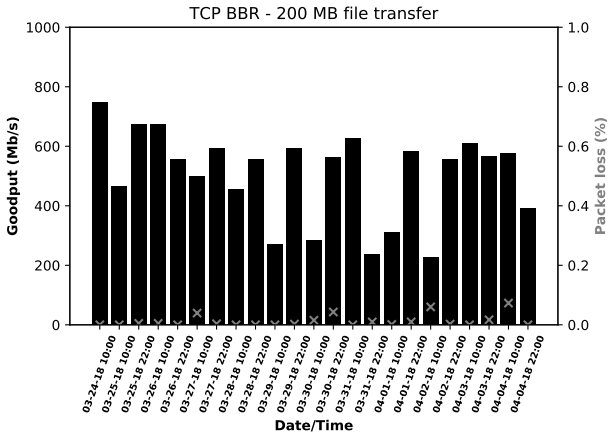
a) Goodput & Packet loss for FASP (1 GB)



b) Goodput & Packet loss for FASP (5 GB)

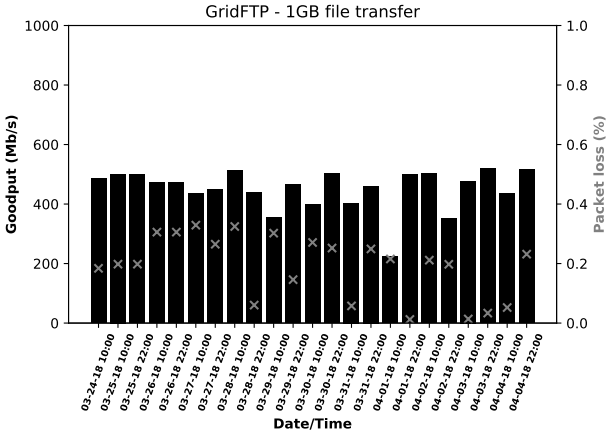


a) Goodput & Packet loss for GridFTP (200 MB)

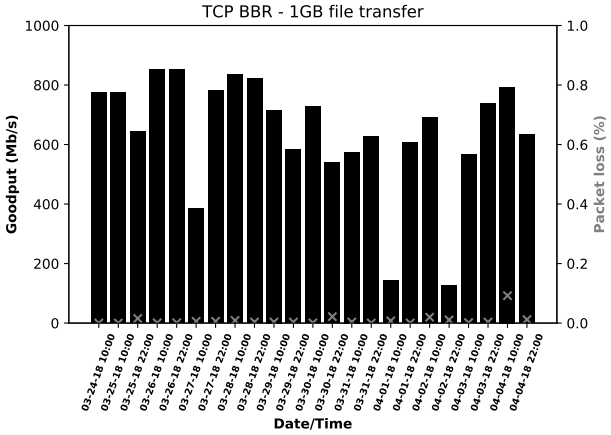


b) Goodput & Packet loss for TCP BBR (200 MB)

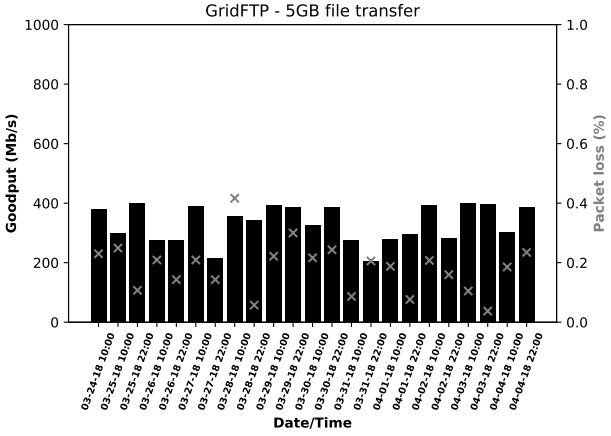
Figure 4.6: Goodput & Packet loss for protocols with various file sizes (1) - Local Setup



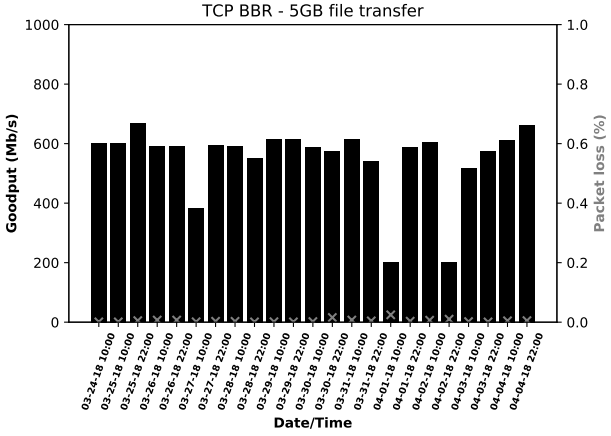
c) Goodput & Packet loss for GridFTP (1 GB)



e) Goodput & Packet loss for TCP BBR (1 GB)

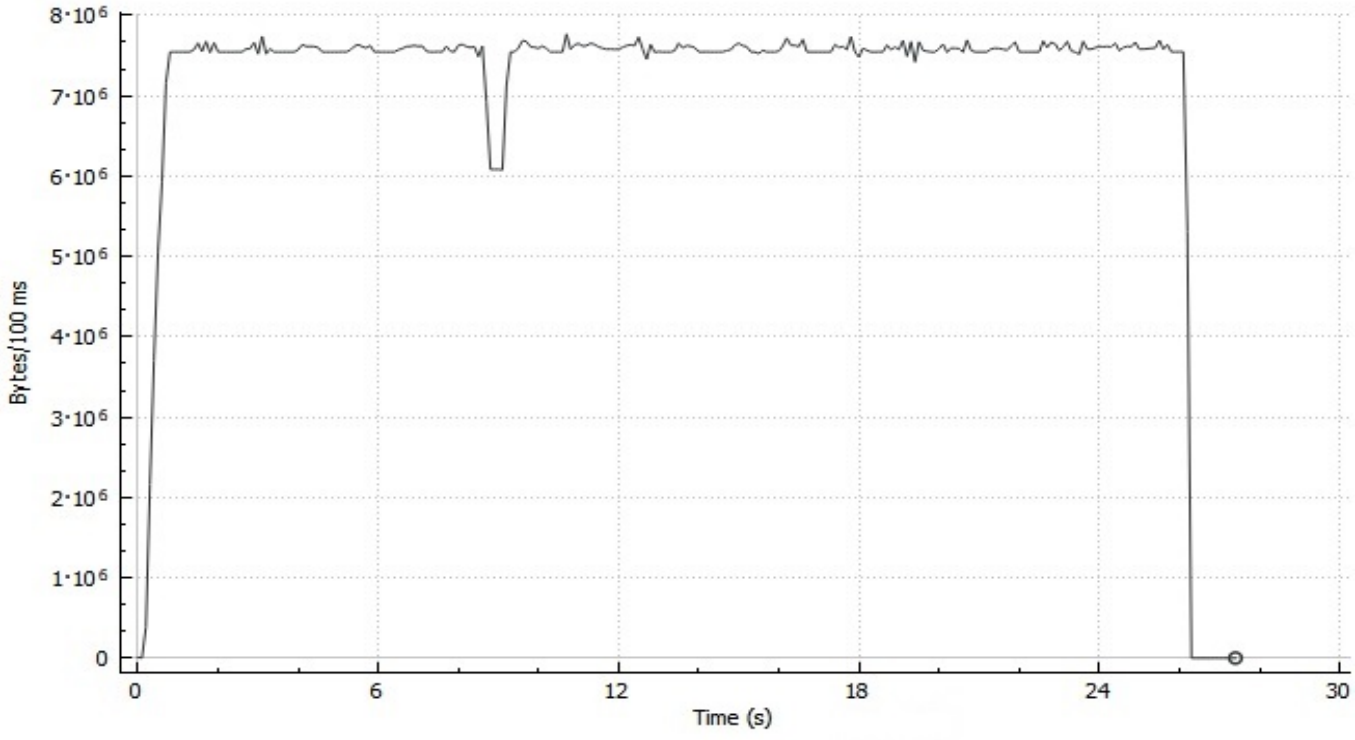


d) Goodput & Packet loss for GridFTP (5 GB)

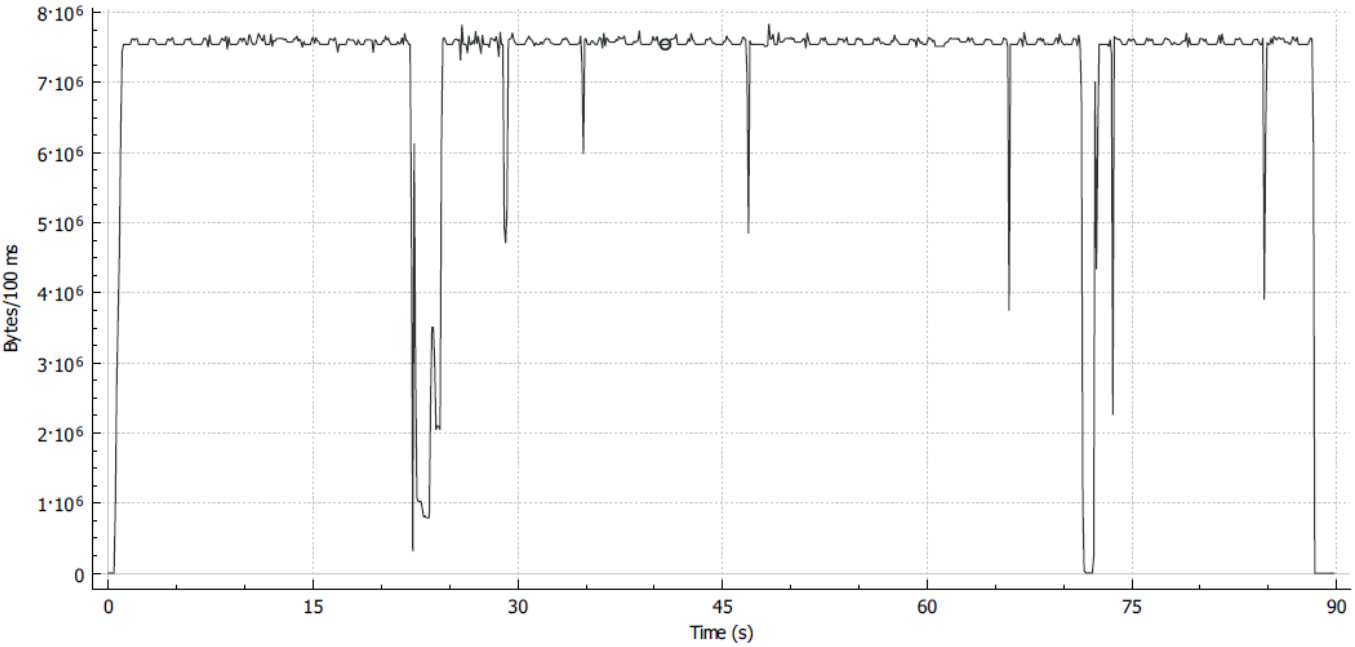


f) Goodput & Packet loss for TCP BBR (5 GB)

Figure 4.7: Goodput & Packet loss for protocols with various file sizes (2) - Local Setup

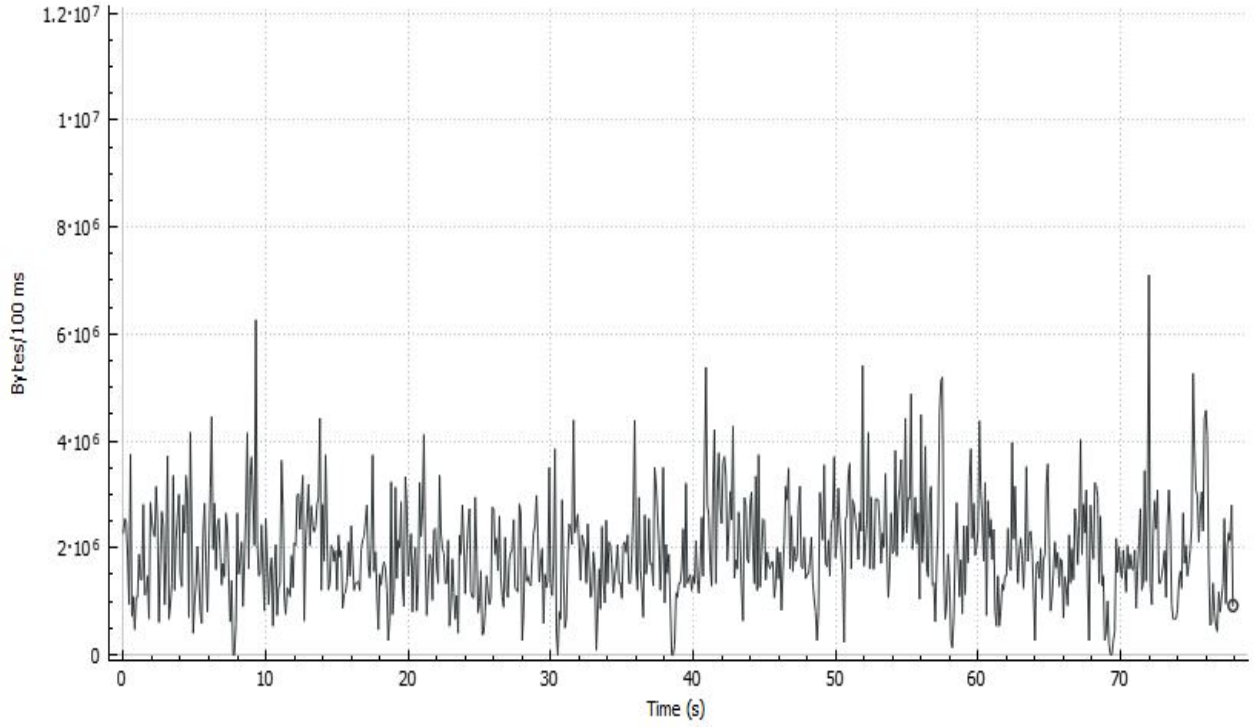


a) Sending Rate over Time for FASP (1 GB) - Replication 3

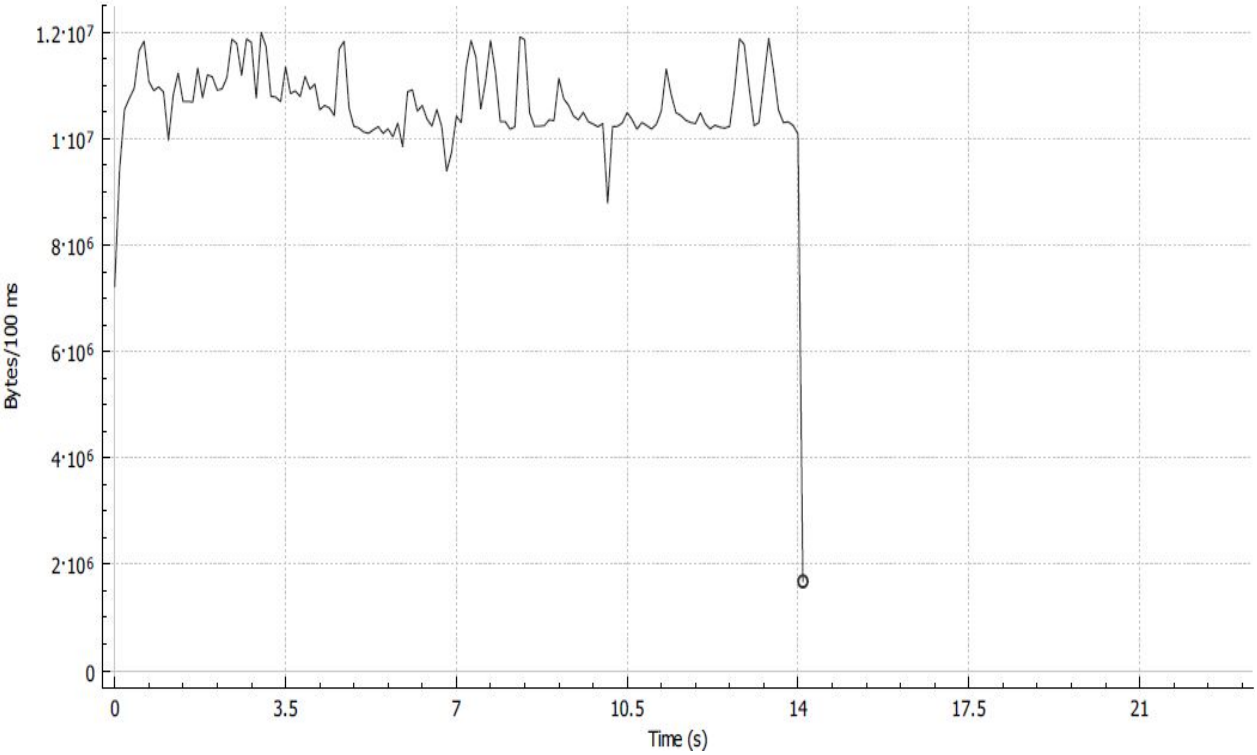


b) Sending Rate over Time for FASP (5 GB) - Replication 3

Figure 4.8: Sending Rate over Time for FASP - Local Setup

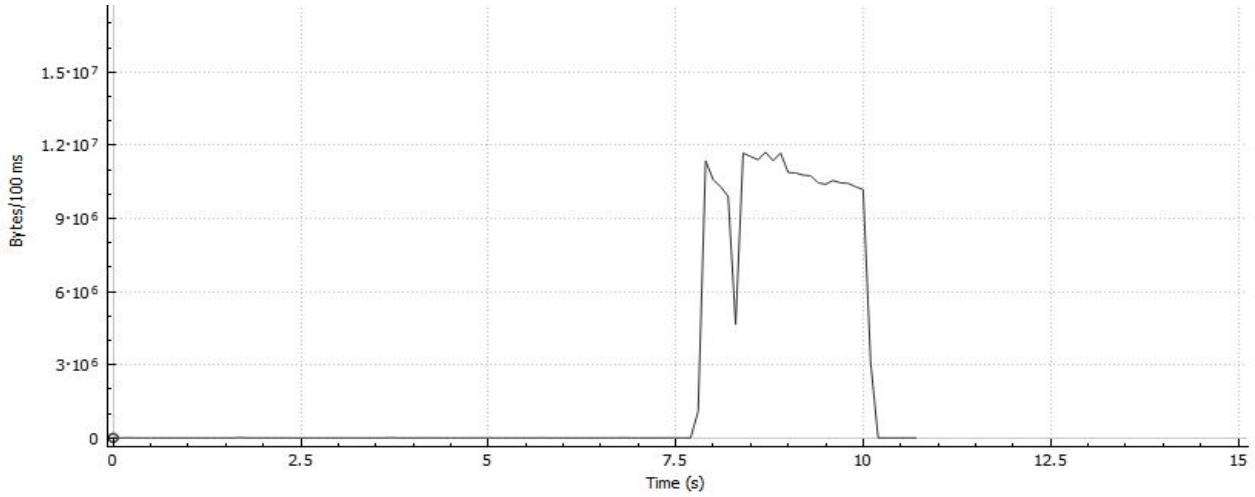


a) Sending Rate over Time for TCP BBR (1 GB) - Replication 16

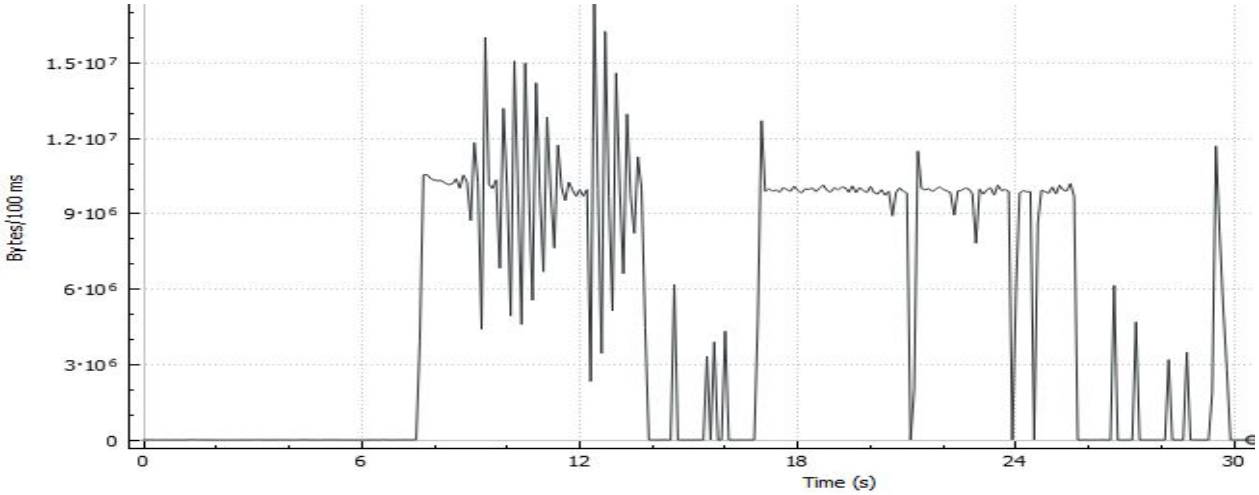


b) Sending Rate over Time for TCP BBR (1 GB) - Replication 22

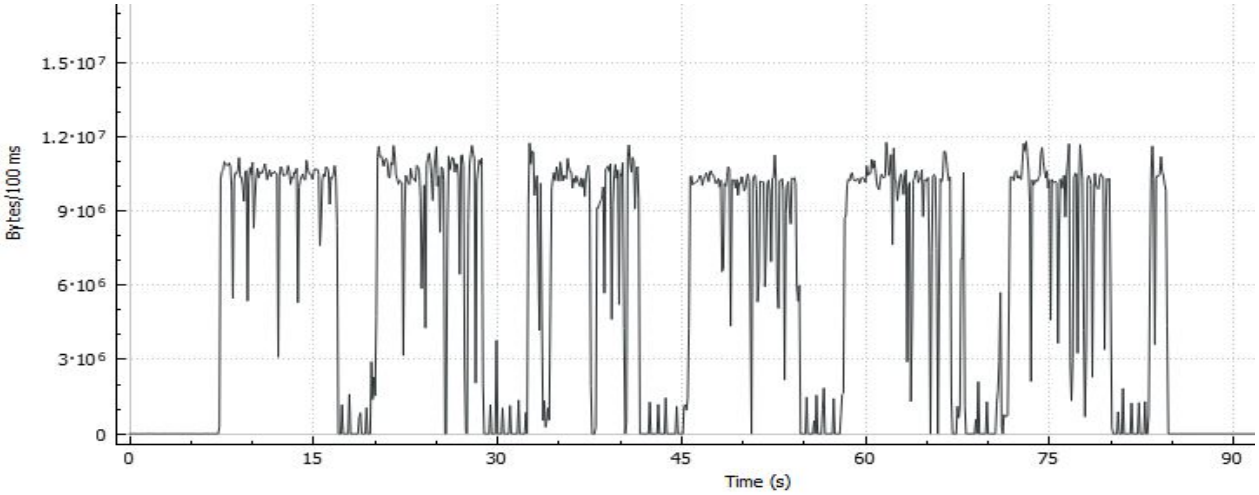
Figure 4.9: Sending Rate over Time for TCP BBR - Local Setup



a) Sending Rate over Time for GridFTP (200 MB) - Replication 2



b) Sending Rate over Time for GridFTP (1 GB) - Replication 2



c) Sending Rate over Time for GridFTP (5 GB) - Replication 2

Figure 4.10: Sending Rate over Time for GridFTP - Local Setup

Waterloo Testbed Results for Single File Transfers

The average goodput for all the runs is shown in Figure 4.11. In this category, TCP BBR achieves the highest goodput by a wide margin while FASP and GridFTP have similar average goodput. The goodput of GridFTP increases as the file size increases, which is the opposite of what was observed in the local setup. This same behaviour is observed in Eastcloud in Figure 4.20, which confirmed that GridFTP does not experience a high number of drops in its sending rate. The aggregate goodput of QUIC reduced compared to the local setup and went below 100 Mbps.

Overall, the average goodput obtained by all the protocols is lower than the goodput rates in local. Logs confirmed that this is caused by a firewall in the destination's network, which dropped many UDP packets. The firewall perceived the FASP packets to be causing a UDP flood attack since FASP sends many UDP packets as fast as possible. Also, the network interface card on the destination machine showed overrun errors, which suggests that the system may have had issues keeping up with traffic.

In this setup, the goodput results for all the protocols increases as the file size increases, except for QUIC, which has similar goodput for all the file sizes. This is because QUIC is limited by the maximum sending rate it can use for a single flow, which cannot be controlled. In terms of consistency, TCP BBR had similar goodput results for both the 1 GB and the 5 GB file size, but showed a high difference between the runs compared to the other protocols. This may be related to the amount of delay on the link at the time of each run.

As shown in Figure 4.12, the aggregate packet loss for all runs is the highest for FASP compared to the other protocols, which showed near zero packet loss. Furthermore, the other protocols show that they deal well with bandwidth limitations and congestion on this network link.

The results for each run in Figure 4.13 and 4.14 show that the goodput values for GridFTP vary from one run to the other. QUIC also showed a similar behaviour with a high variation between each of its runs. This could be caused by the different amounts of traffic present on the network path, which affects the performance of GridFTP and QUIC accordingly. The average goodput for QUIC is still lower than 100 Mbps in this scenario. Both of these protocols try to be fair to other traffic by lowering their sending rate. In addition, the goodput in this setup has reduced compared to the local setup. However, the packet loss for GridFTP and QUIC is lower than the one in Local setup, which suggests that the goodput does not decrease because of packet loss. The packet loss for FASP is significantly high and this might be related to the fact that FASP tries to send packets at a high sending rate, but the destination's firewall is dropping many of these packets. Moreover, the goodput and packet loss percentage of TCP BBR is more consistent in this setup for all the runs and goodput does not seem to be severely affected. The performance of all the protocols in this setup is lower than in the local setup.

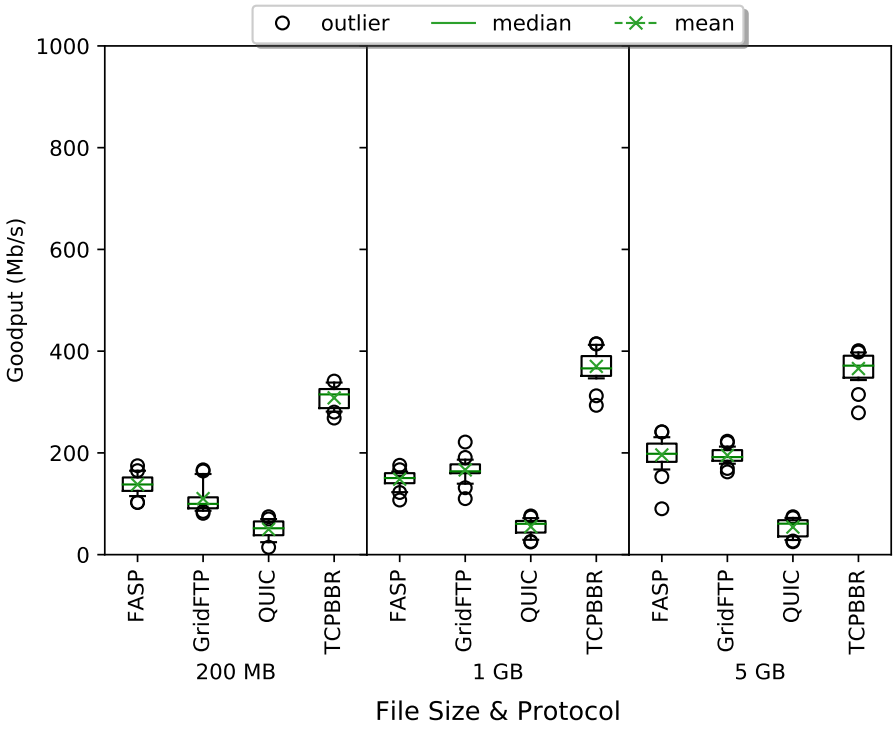


Figure 4.11: Aggregate Goodput for Each Protocol with Various File Sizes - Waterloo Setup

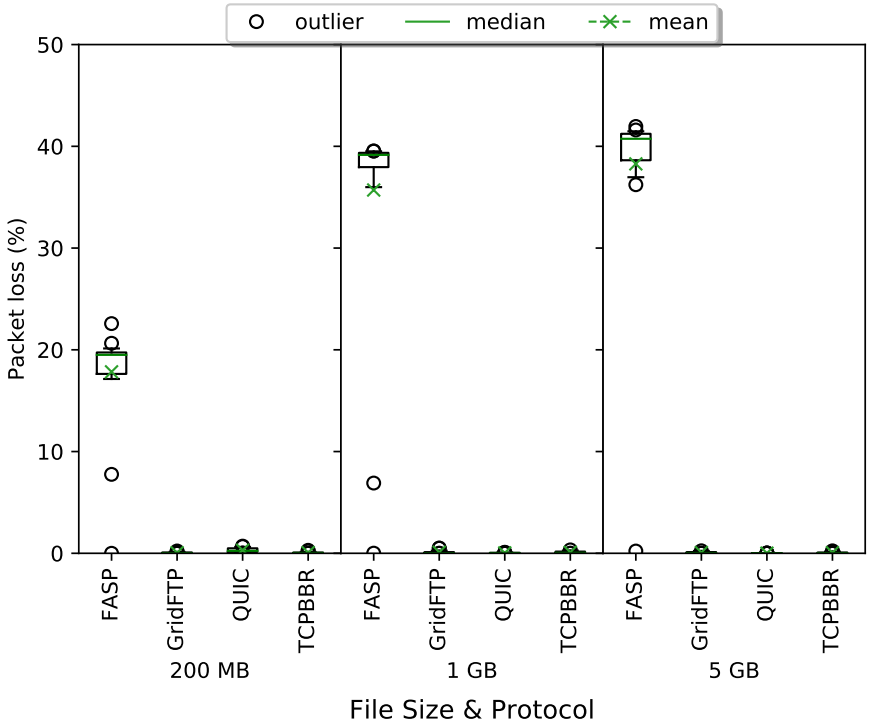
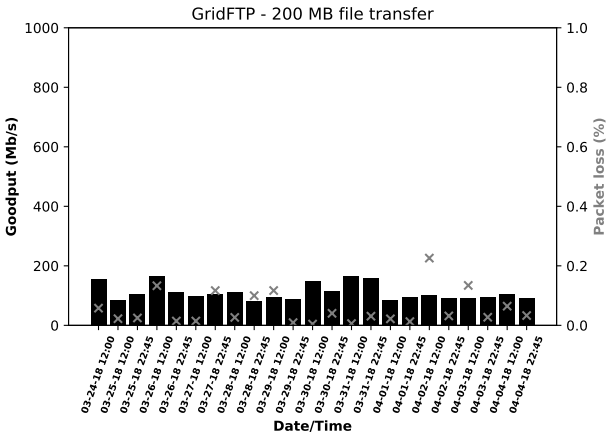
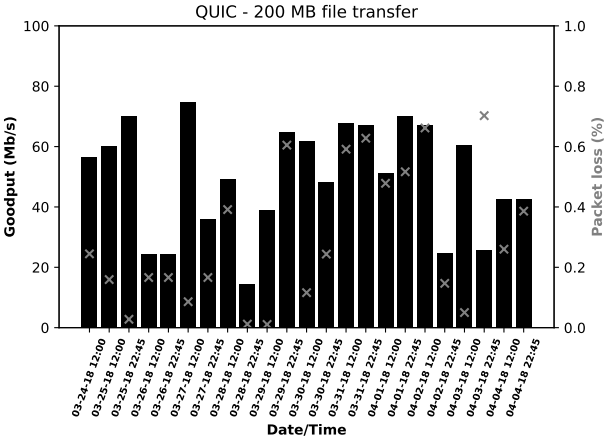


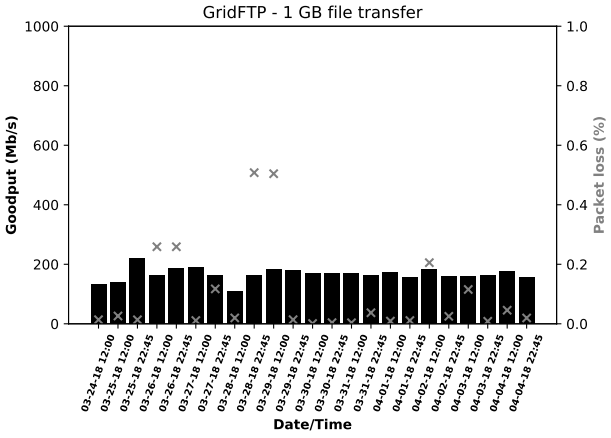
Figure 4.12: Aggregate Packet Loss for Each Protocol with Various File Sizes - Waterloo Setup



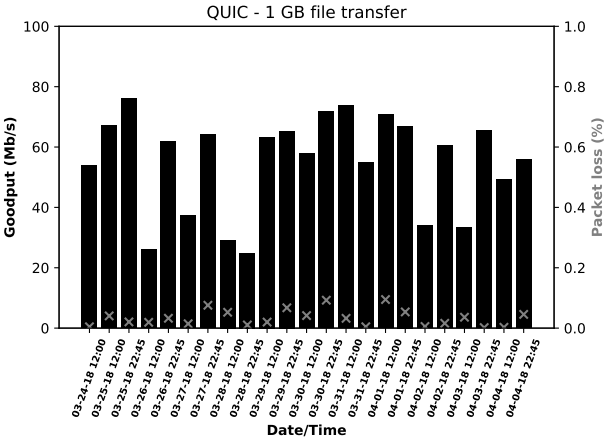
a) Goodput & Packet Loss for GridFTP (200 MB)



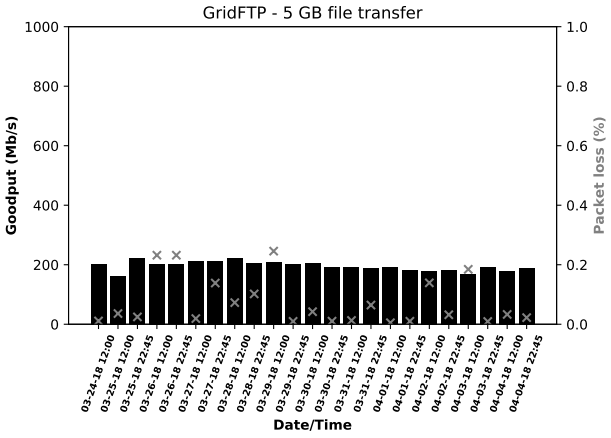
d) Goodput & Packet Loss for QUIC (200 MB)



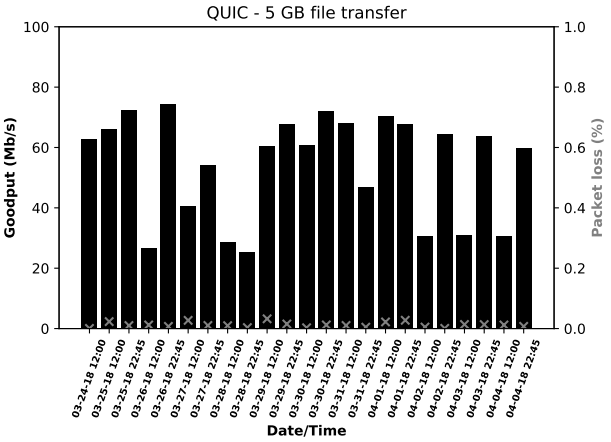
b) Goodput & Packet Loss for GridFTP (1 GB)



e) Goodput & Packet Loss for QUIC (1 GB)

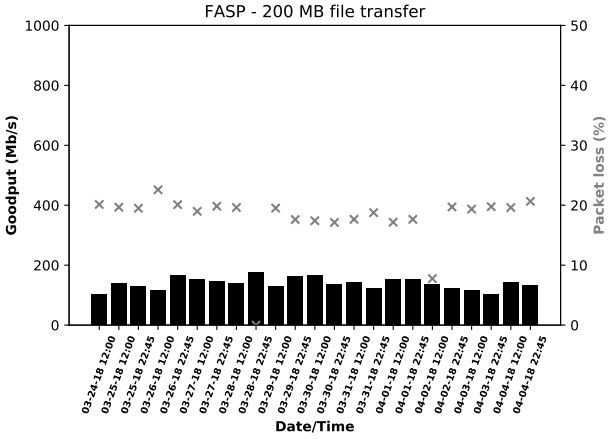


c) Goodput & Packet Loss for GridFTP (5 GB)

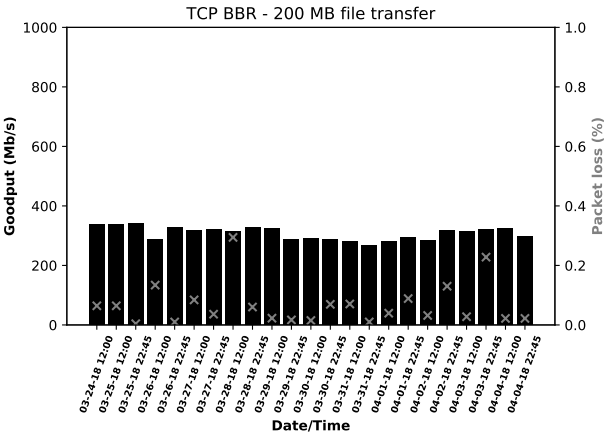


f) Goodput & Packet Loss for QUIC (5 GB)

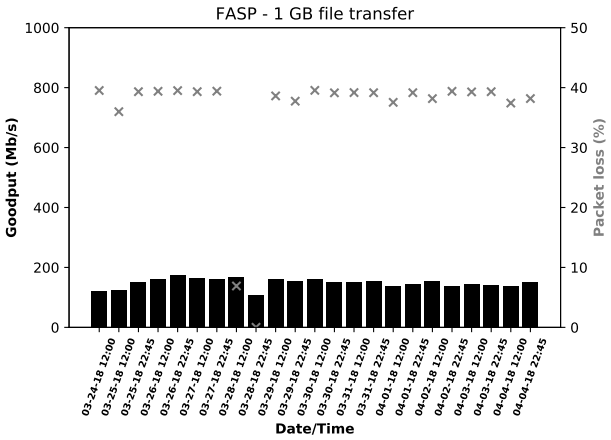
Figure 4.13: Goodput & Packet Loss for Protocols with Various File Sizes (1) - Waterloo Setup



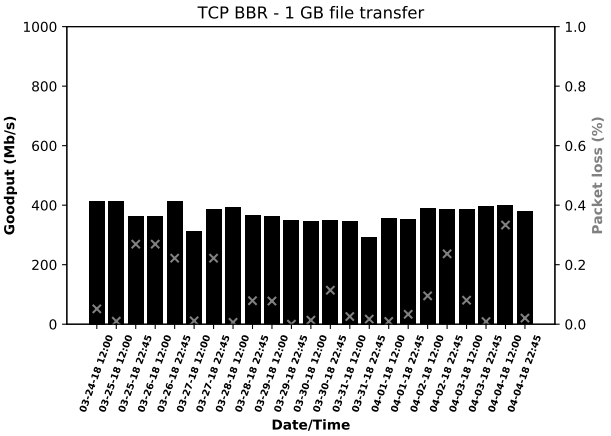
a) Goodput & Packet Loss for FASP (200 MB)



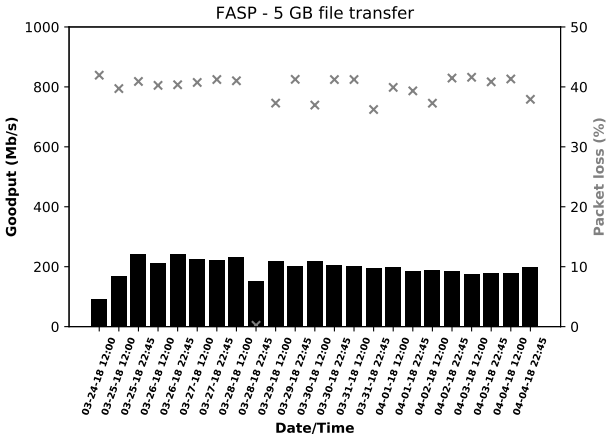
d) Goodput & Packet Loss for TCPBBR (200 MB)



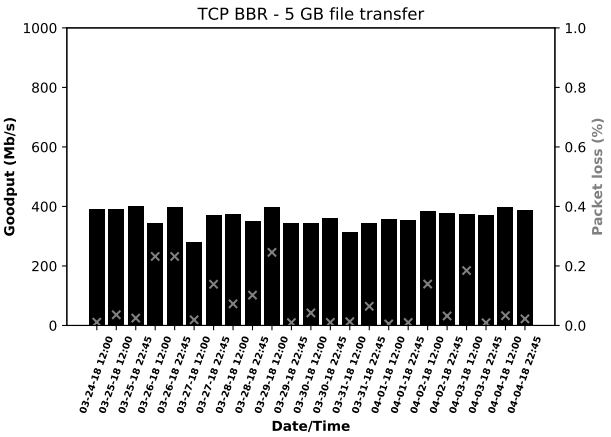
b) Goodput & Packet Loss for FASP (1 GB)



e) Goodput & Packet Loss for TCPBBR (1 GB)



c) Goodput & Packet Loss for FASP (5 GB)



f) Goodput & Packet Loss for TCP BBR (5 GB)

Figure 4.14: Goodput & Packet Loss for Protocols with Various File Sizes (2) - Waterloo Setup

Eastcloud Testbed Results for Single File Transfers

For all the experiments in Eastcloud, TCP BBR was excluded because it had a surprisingly low performance compared with its performance in the Waterloo setup, which should have similar network characteristics and RTT. It is possible that there is middle box or a traffic shaping device on the path used in this setup.

As shown in Figure 4.15, FASP and GridFTP had similar goodput results in this setup, except for the 200 MB scenario. Both of the protocols increased their goodput as the file size increased. This shows that both protocols are not able to ramp up to a high sending rate in a short period of time. For QUIC, the goodput results for all file sizes were similar and it had the lowest goodput among all the protocols.

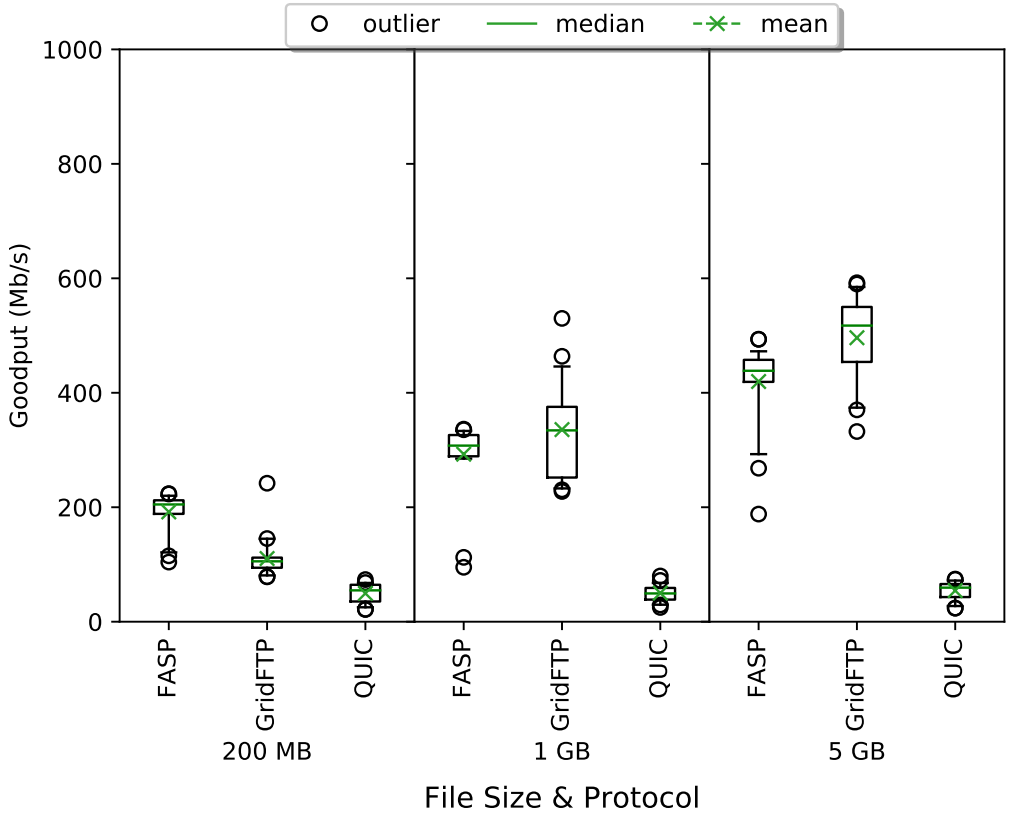


Figure 4.15: Aggregate Goodput for Each Protocol with Various File Sizes - Eastcloud setup

Compared to the local setup results, the goodput is lower especially for the 200 MB and 1 GB scenarios. This is caused by the increased RTT, more hops for packets to pass through, and potentially an overload on the destination's machine. FASP performs better than GridFTP as it is able to achieve consistent goodput between the replications and with lower packet loss.

The packet loss percentage for FASP and QUIC were lower than GridFTP as shown in Figure 4.16. The high packet loss for GridFTP could be caused by network congestion; it still performs better than the other protocols, especially with the 5 GB file size. This may be related to the fast recovery algorithm, which tries to quickly recover lost data packets without affect the sending rate. Also, FASP showed some packet loss

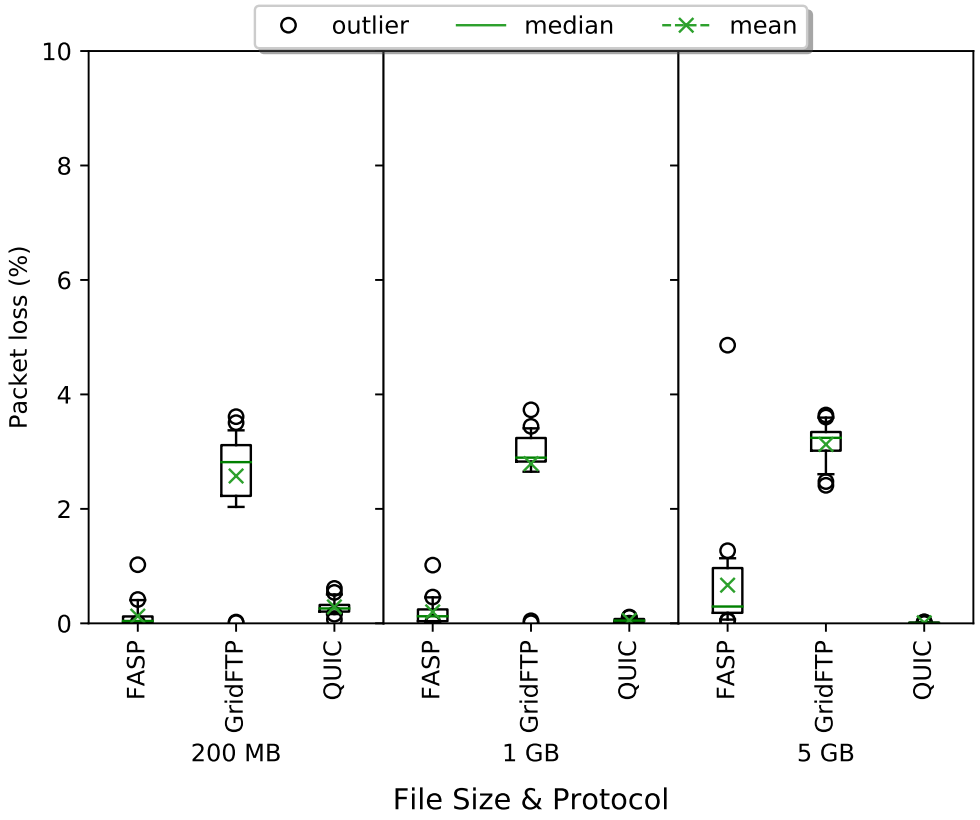


Figure 4.16: Aggregate Packet Loss for Each Protocol with Various File Sizes - Eastcloud setup

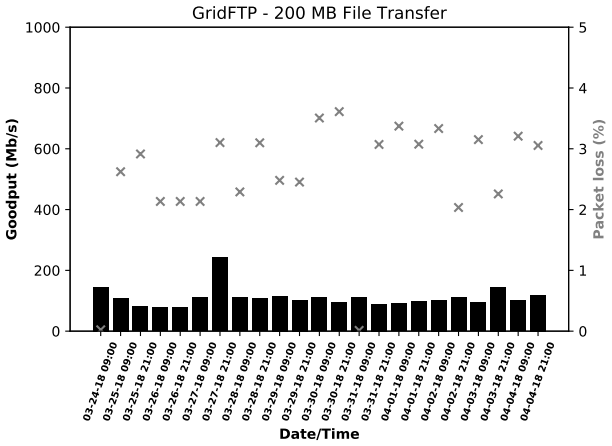
especially for the 5 GB file size, but it is not substantial since the packet loss results for most of the runs were below 1 percent.

For every individual run of 1 GB and 5 GB, the goodput and packet loss results in the Eastcloud setup are shown in Figure 4.18. This shows that the performance of FASP can be replicated for most of the days while the performance of GridFTP and QUIC varies more often. However, packet loss rates for GridFTP and QUIC varied from one run to another, which can be possibly caused by the different amounts of congestion induced on the network for each replication. In terms of goodput, FASP achieved a more consistent goodput than the other protocols, while GridFTP was performed poorly in the 200 MB scenario and had a high variation in the results for both the 1 GB and 5 GB file transfers. However, GridFTP and FASP were able to send data in a shorter amount of time than QUIC because of GridFTP's parallel flows feature and FASP's target transfer rate.

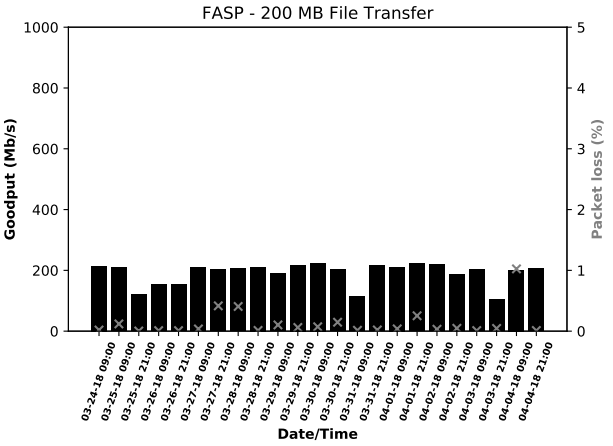
The performance of the protocols for the 200 MB file transfer was more stable and each day's results was similar to the other and so is not shown. The fluctuation in the results for GridFTP and QUIC for the 1 GB and 5 GB file transfers could be caused by the congestion control mechanisms that each use in responding to available bottleneck rate and network congestion. The performance of all the protocols can also vary from day to day depending on the state of the host machine.

With 1 GB and 5 GB file transfers, FASP seems to have a lower goodput for 1 GB compared to the 5

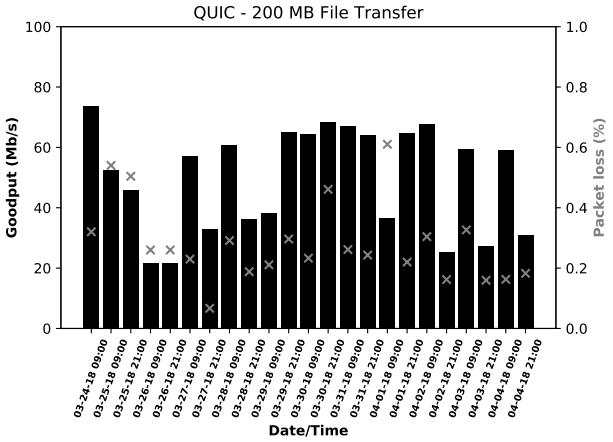
GB file transfer. This happens for the same reason as the one observed in the local setup. A sample run was chosen for the 1 GB and 5 GB file transfer for FASP and GridFTP, which would show the sending rate over time for both protocols. The sending rate used by FASP over time is shown in Figure 4.19 for a sample run, which confirms that FASP operates at peak rate for a shorter percentage of time with the 1 GB file. The peak rate is the target bit rate (600 Mbps) for FASP, which was set by the user. Additionally, the probes for sending rate by FASP might be slower due to the longer RTT on this link. A similar behaviour is observed with GridFTP in Figure 4.20. GridFTP take a long time between 6 and 8 seconds to ramp up to a high sending rate while FASP takes between 3 and 6 seconds. This behaviour can be investigated further in future work to find out the cause of the very slow start up. For the 200 MB file transfer, GridFTP takes a long time to even start using slow start and the file transfer gets completed before it is able to enter the congestion avoidance phase. Also, the sending rate is not as high as the one observed for the 1 GB and 5 GB file transfers, which confirms why the goodput was low for the 200 MB file transfer in this setup. With the 1 GB and the 5 GB file transfers, GridFTP takes longer to send the file while experiencing some packet loss events, which causes it to decrease its sending rate a couple of times during the transfer. This is observed in the congestion avoidance phase after exiting the slow start phase and it tries to increase its sending rate again after recovering from a packet loss. GridFTP also reaches a sending rate of 800 Mbps, but cannot sustain it throughout the transfer because of the packet loss events. The main difference between the 1 GB and 5 GB file transfers is the amount of time that both take to complete the transfer.



a) Goodput & Packet loss for GridFTP (200 MB)

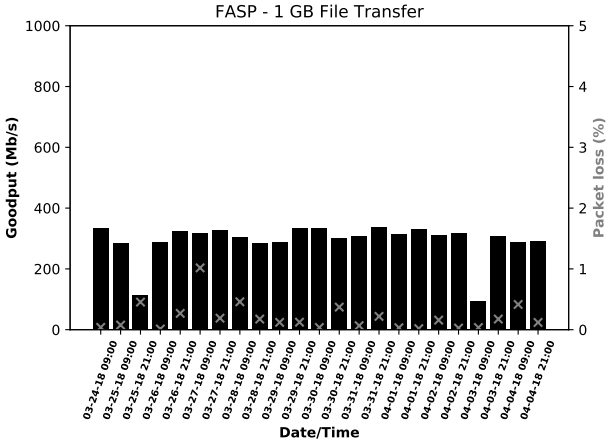


b) Goodput & Packet loss for FASP (200 MB)

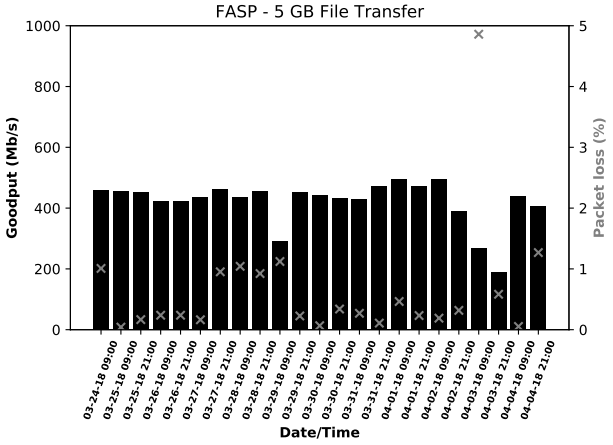


c) Goodput & Packet loss for QUIC (200 MB)

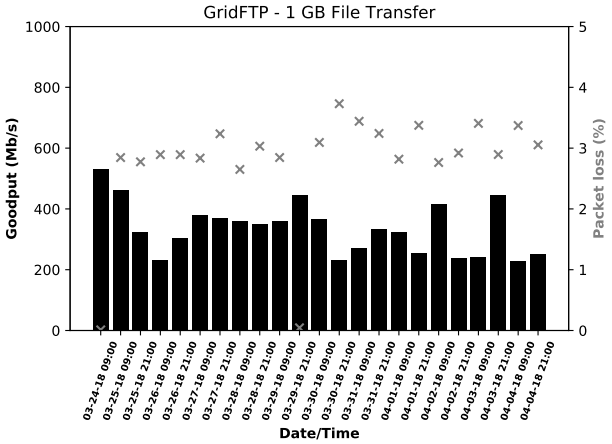
Figure 4.17: Goodput & Packet loss for protocols with various file sizes (1) - Eastcloud Setup



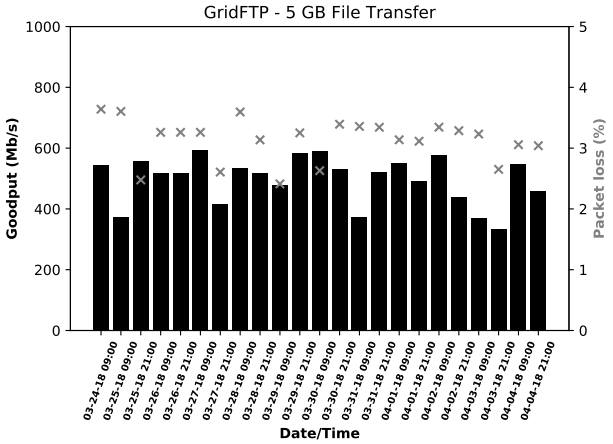
d) Goodput & Packet loss for FASP (1 GB)



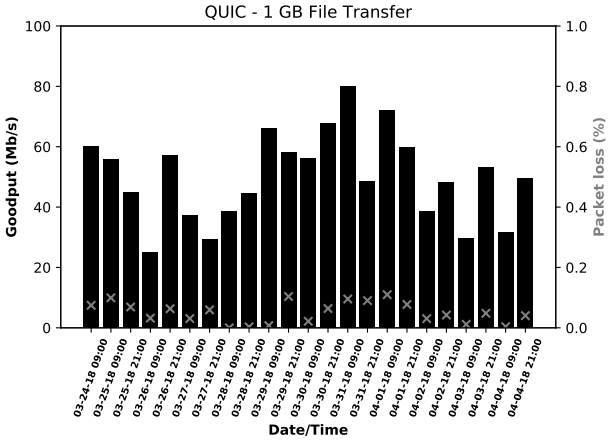
e) Goodput & Packet loss for FASP (5 GB)



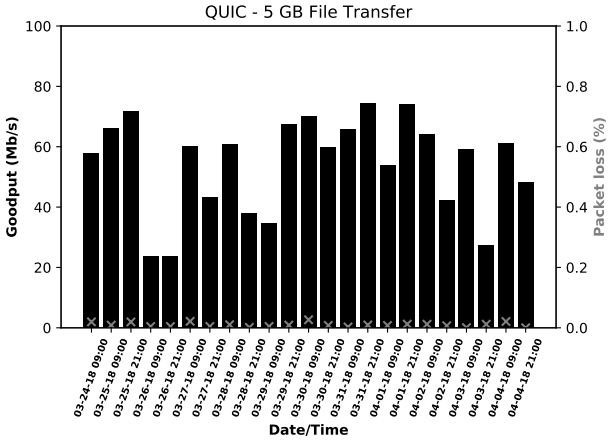
f) Goodput & Packet loss for GridFTP (1 GB)



g) Goodput & Packet loss for GridFTP (5 GB)

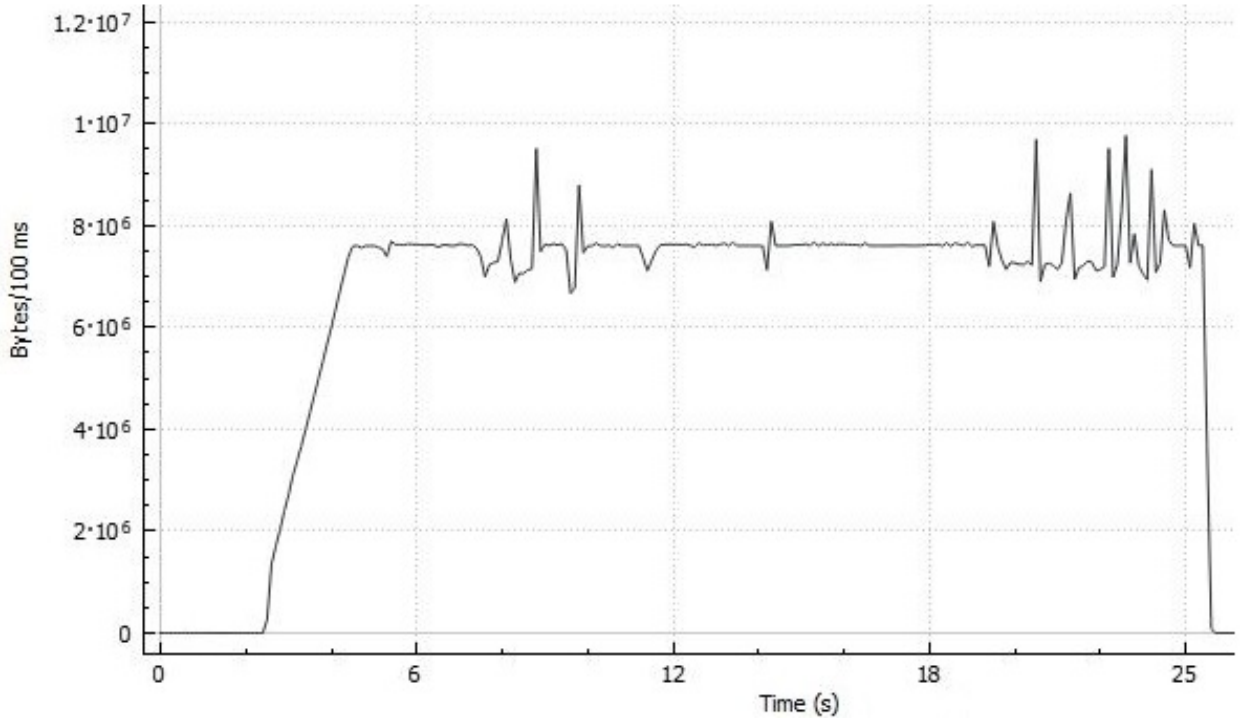


h) Goodput & Packet loss for QUIC(1 GB)

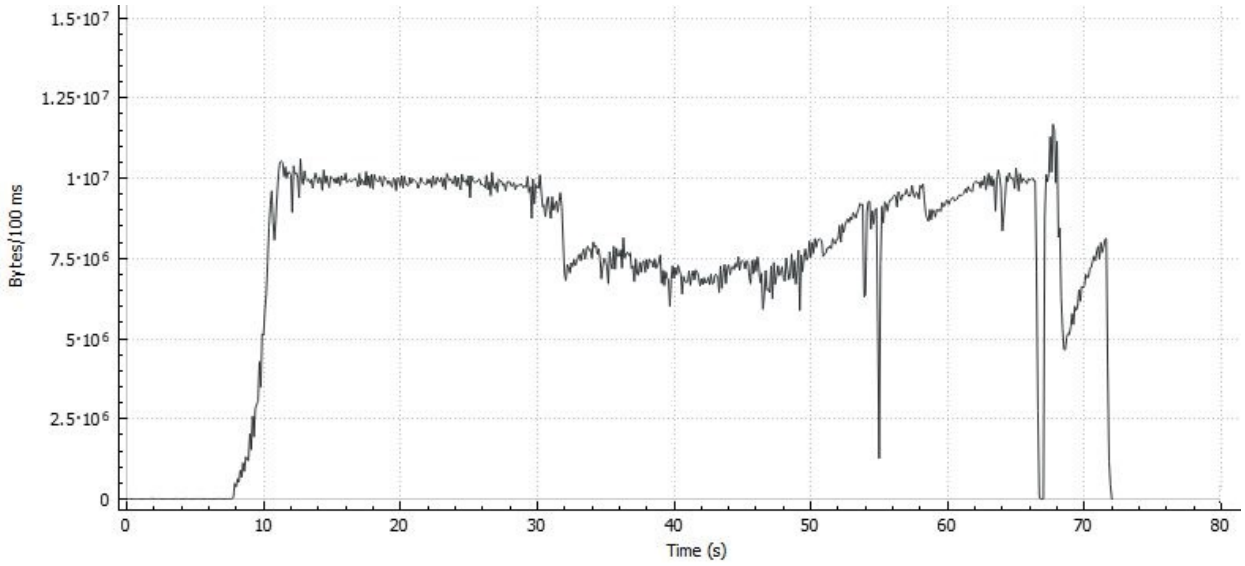


i) Goodput & Packet loss for QUIC (5 GB)

Figure 4.18: Goodput & Packet loss for protocols with various file sizes (2) - Eastcloud Setup

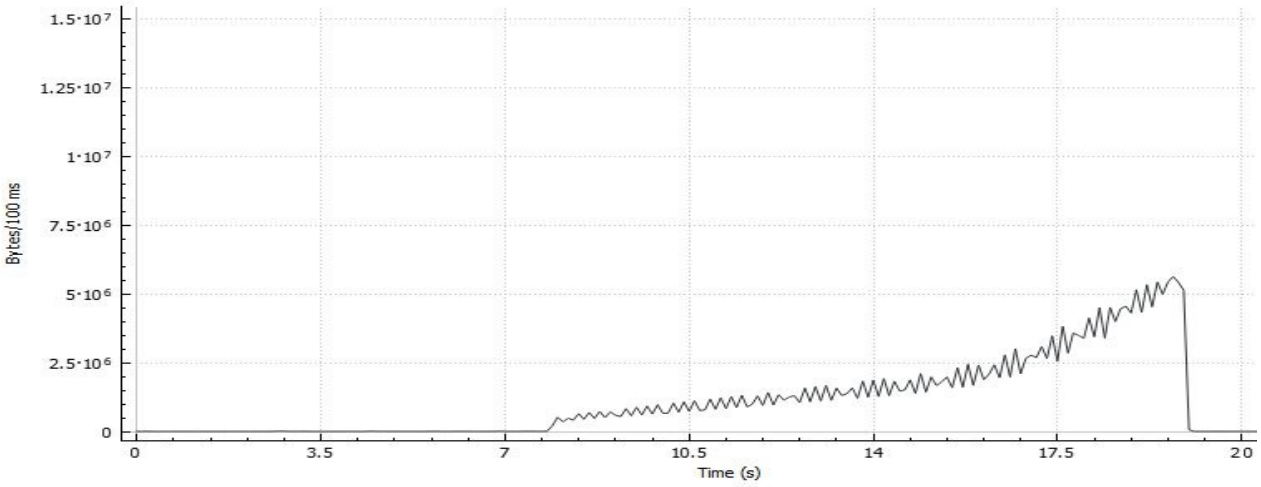


a) Sending Rate over Time for FASP (1 GB)- Replication 7

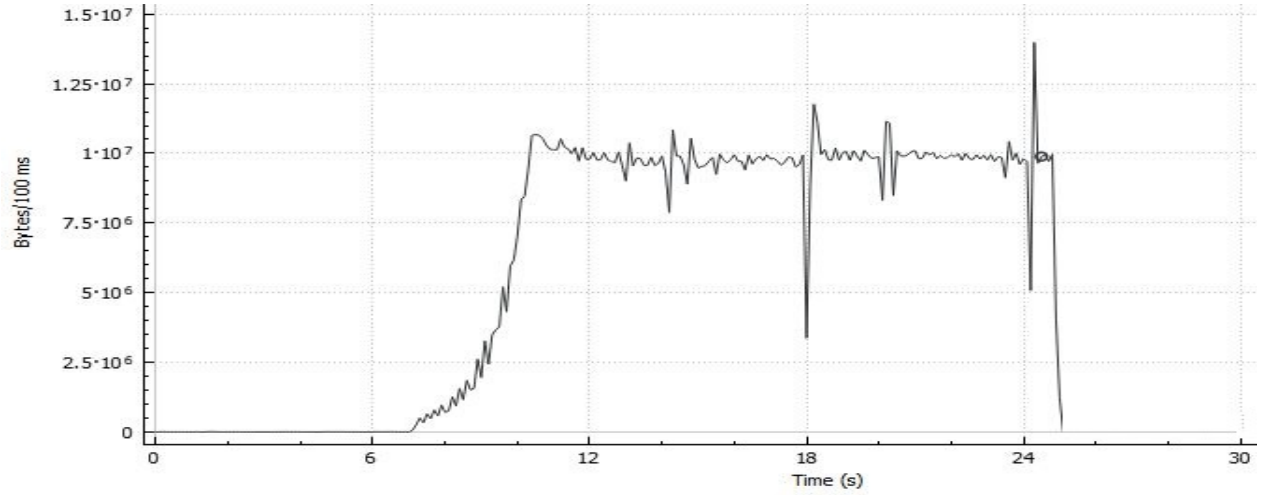


b) Sending Rate over Time for FASP (5 GB) - Replication 7

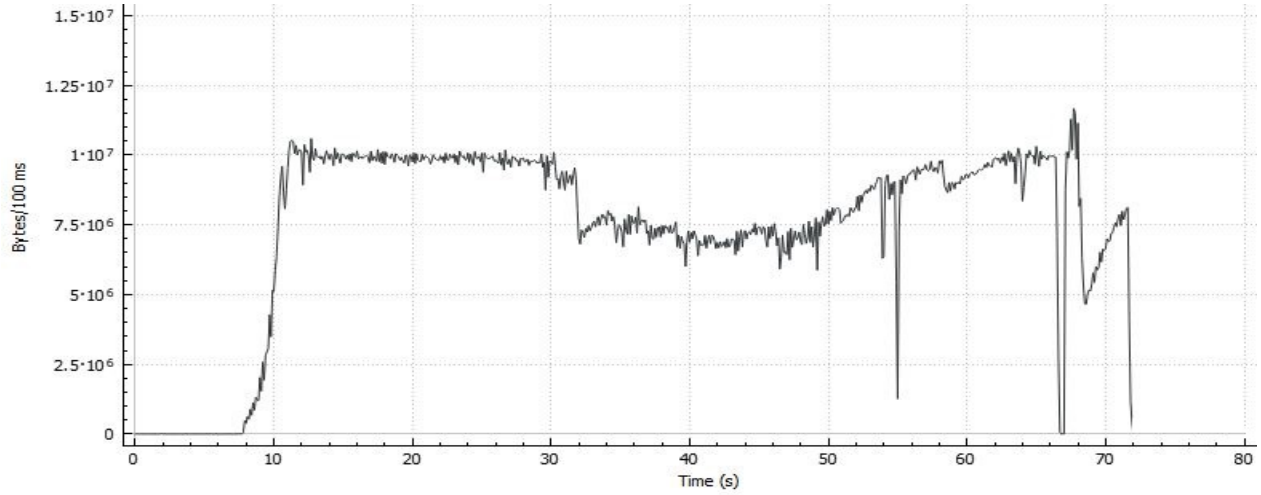
Figure 4.19: Sending Rate over Time for FASP - Eastcloud Setup



a) Sending Rate over Time for GridFTP (200 MB) - Replication 7



b) Sending Rate over Time for GridFTP (1 GB) - Replication 7



c) Sending Rate over Time for GridFTP (5 GB) - Replication 7

Figure 4.20: Sending Rate over Time for GridFTP - Eastcloud Setup

International Testbed Results for Single File Transfers

The goodput results in Figure 4.21 show that FASP achieves the highest aggregate goodput for the 200 MB file transfer compared to the other protocols. Also, FASP achieves similar goodput to GridFTP for the 1 GB and 5 GB file transfer. FASP performs well in long RTT networks and achieves higher goodput for the large file size, but it is a bit lower than Figure 4.4.

The goodput for GridFTP increases as the file size increased. This behaviour is caused by the TCP slow start mechanism, which occupies a larger proportion of transfer time as observed in Figure 4.20. This causes all the small files to be transferred before congestion avoidance is entered, preventing the protocol from operating at its maximum on the network for a substantial period of time. Moreover, the sender sends the first packet with a small congestion window size to the receiver and the receiver transmits a packet that has its congestion window size. The sender then slowly increases the window size of each packet until packet loss occurs or until the receiver's maximum window size is reached.

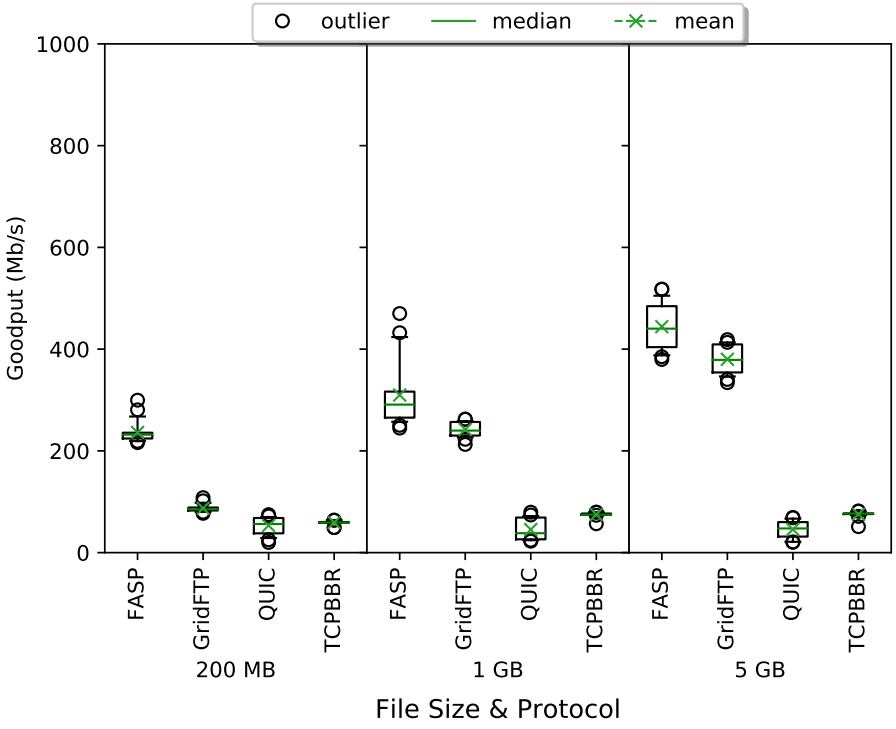


Figure 4.21: Aggregate Goodput for Each Protocol with Various File Sizes - Auckland Setup

On the other hand, the goodput of TCP BBR was below 100 Mbps, which shows that it is not able to reach a high sending rate on this long RTT path. This could be because it is delay-based and obtains a high round trip time value, which causes it to reduce its sending rate. Moreover, the performance of QUIC is still similar to the previous setups. Compared to TCP BBR, QUIC goodput results vary more between the runs, which is also more than the other setups. Most of the runs also had low goodput compared to the runs in local and waterloo.

All the protocols in this setup have achieved a low packet loss percentage, as shown in Figure 4.22. Unlike its previous performance in the other setups, FASP has shown a reduced amount of packet loss among all the protocols. This confirms that it is more stable and is less aggressive in long distance transfers of these sizes. The packet loss for the other protocols is still considered low, and does not influence goodput at this level.

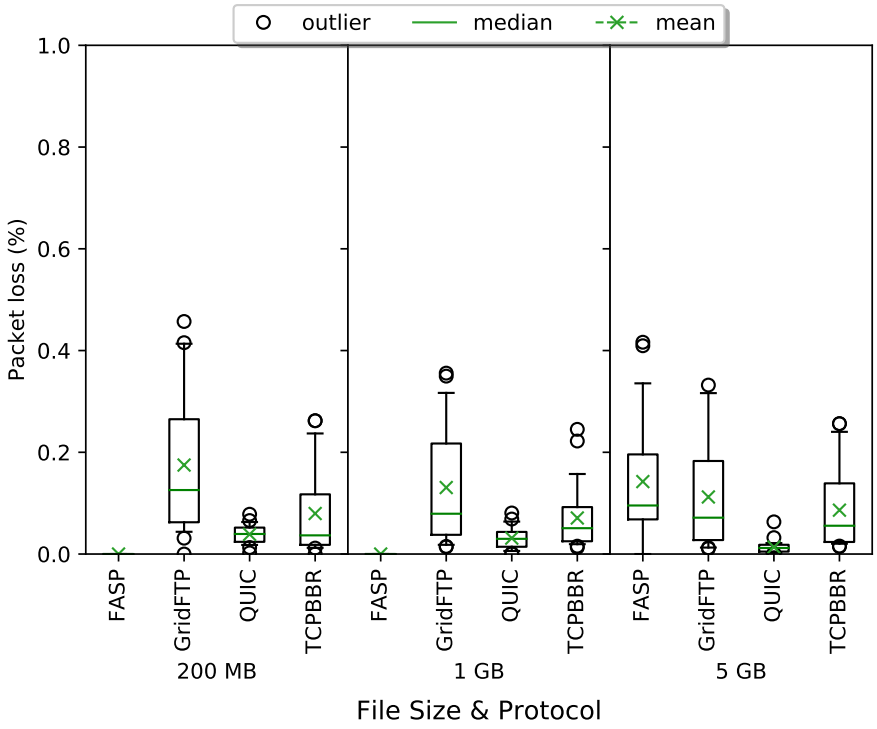
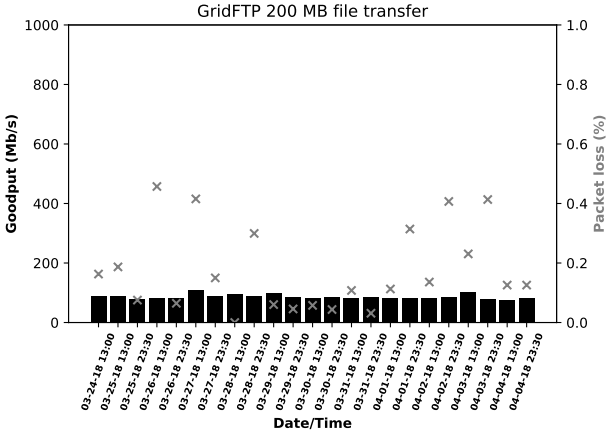


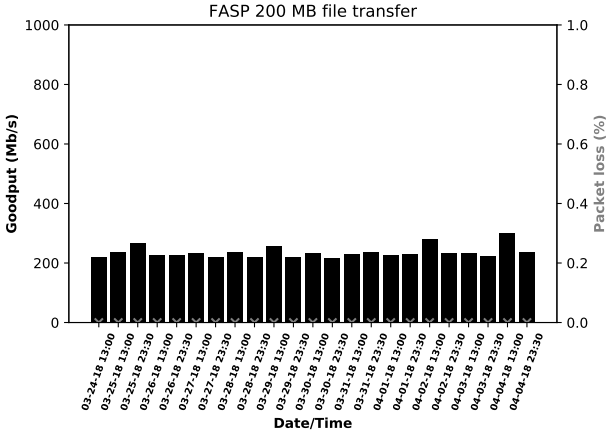
Figure 4.22: Aggregate Packet Loss for Each Protocol with Various File Sizes - Auckland Setup

In Figures 4.23 and 4.24, the goodput and packet loss results for every run of all the protocols is shown. In terms of consistency, FASP and GridFTP were the protocols with similar results for each run. Both of these protocols do not seem to be impacted much by the time of day. On the other hand, QUIC and TCP BBR achieve low goodput rates on the long distance link. For TCP BBR, the delay on this link is high and this reduces the sending rate, and correspondingly the goodput. For long distance transfers, FASP is quicker than GridFTP and TCP BBR in terms of reaching a high sending rate. However, FASP achieved a lower goodput for 1 GB compared to the 5 GB file transfer size.

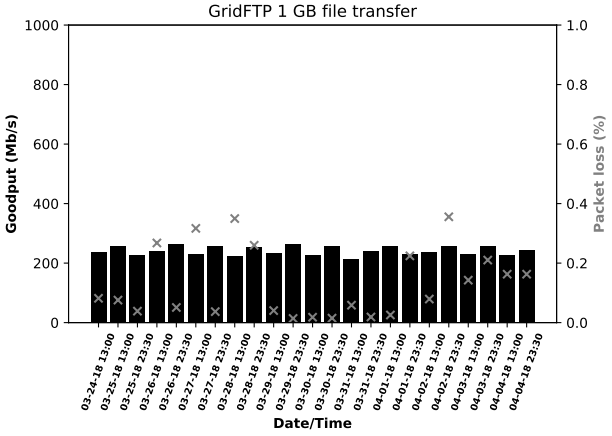
Figure 4.25 shows the sending rate over time for GridFTP for all the file sizes for a sample run. For the 200 MB file transfer, GridFTP does not get out of the slow start phase since the file gets transferred in a very short time, thus it is not able to reach high sending rates. For the 1 GB and the 5 GB file transfers, GridFTP has enough time to exit slow start and enter congestion avoidance and is able to reach higher sending rates. GridFTP also takes a very long time to ramp up as observed earlier in the Eastcloud setup. Additionally, the sending rate is so slow for a 200 MB file and that a file five times the size takes only 31% longer to transfer.



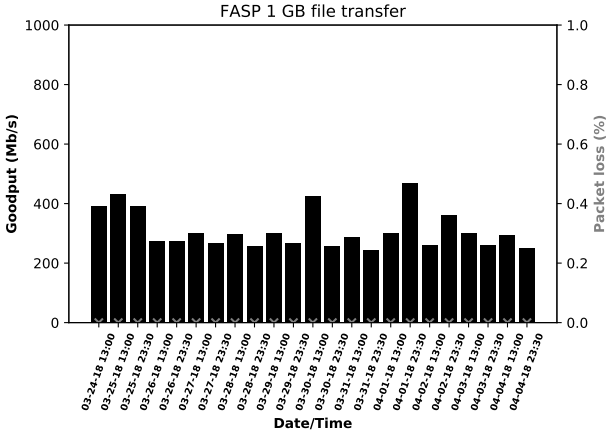
a) Goodput & Packet Loss for GridFTP (200 MB)



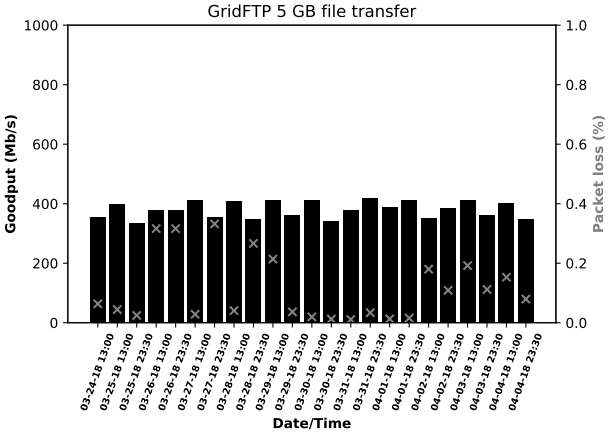
b) Goodput & Packet Loss for FASP (200 MB)



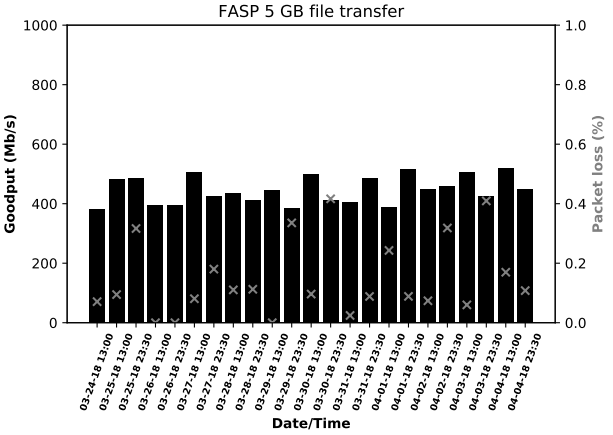
c) Goodput & Packet Loss for GridFTP (1 GB)



d) Goodput & Packet Loss for FASP (1 GB)

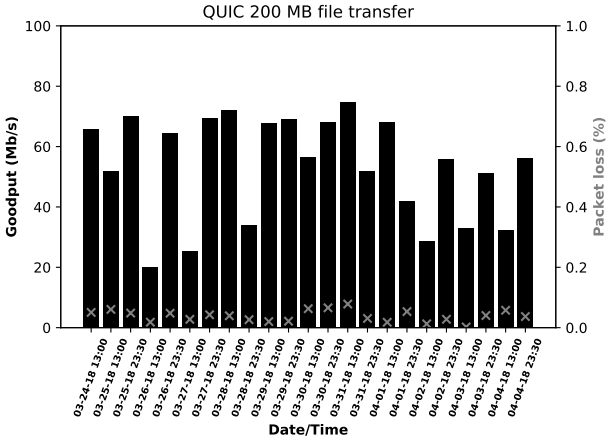


e) Goodput & Packet Loss for GridFTP (5 GB)

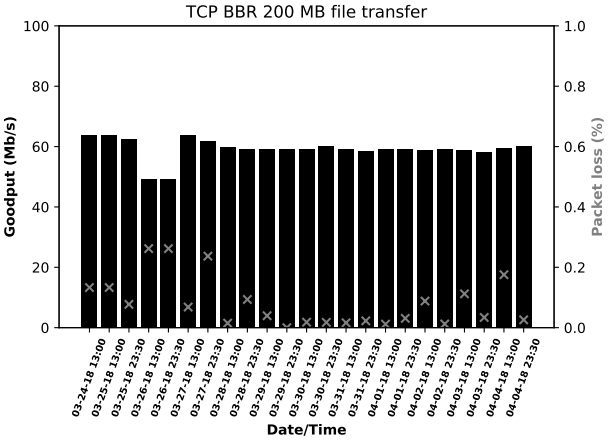


f) Goodput & Packet Loss for FASP (5 GB)

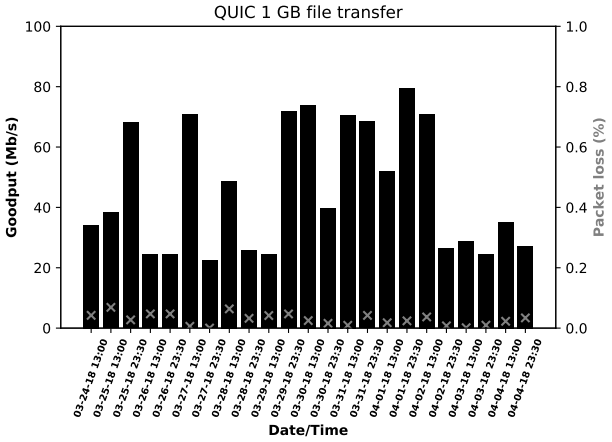
Figure 4.23: Goodput & Packet Loss for Protocols With Various File Sizes (1) - Auckland Setup



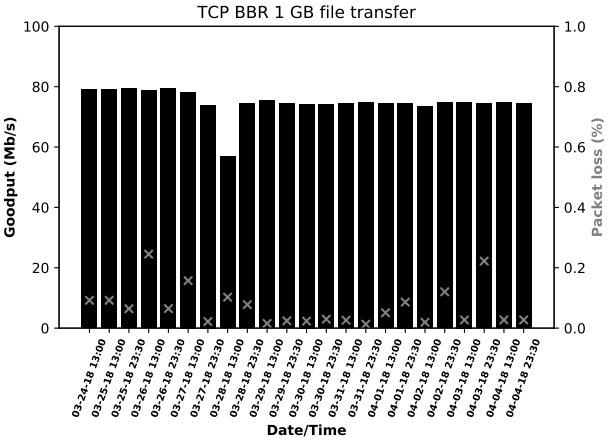
a) Goodput & Packet Loss for QUIC (200 MB)



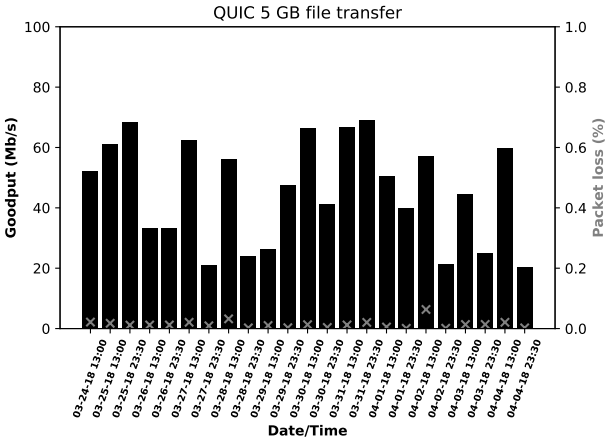
b) Goodput & Packet Loss for TCP BBR (200 MB)



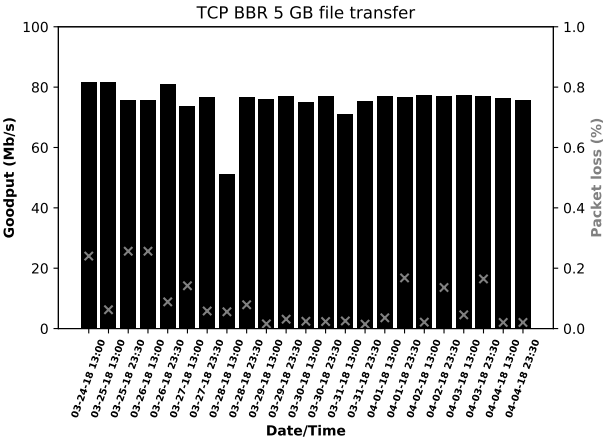
c) Goodput & Packet Loss for QUIC (1 GB)



d) Goodput & Packet Loss for TCP BBR (5 GB)

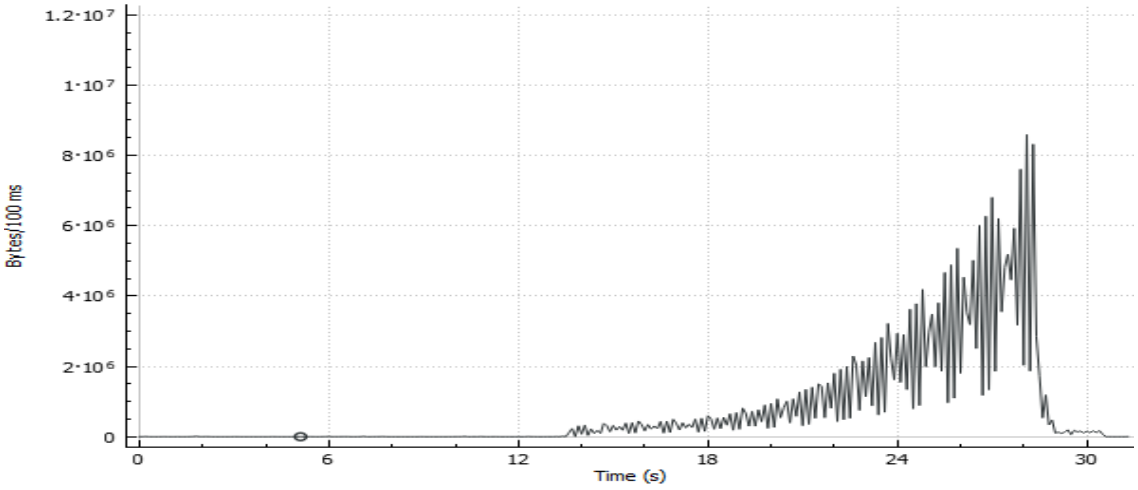


e) Goodput & Packet Loss for QUIC (5 GB)

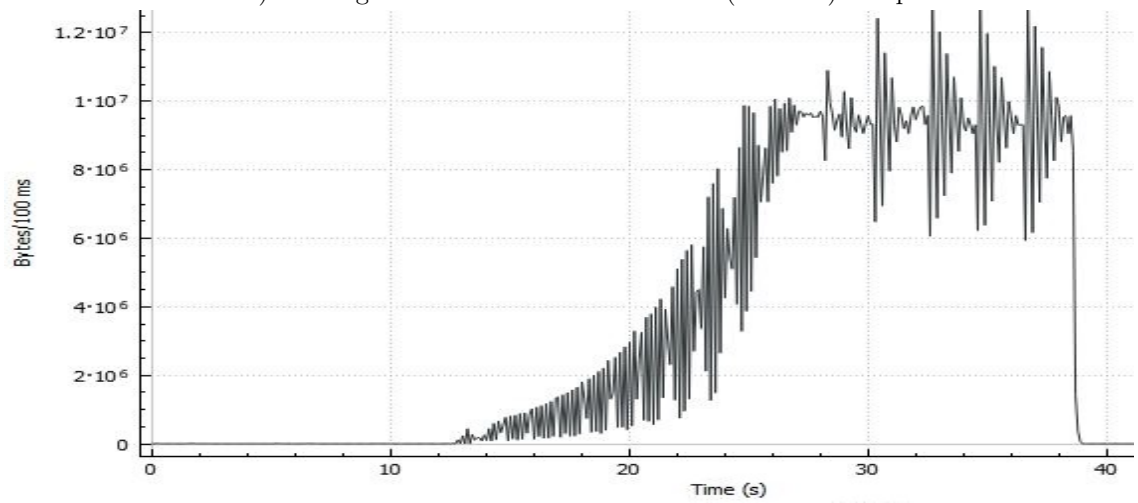


f) Goodput & Packet Loss for TCP BBR (5 GB)

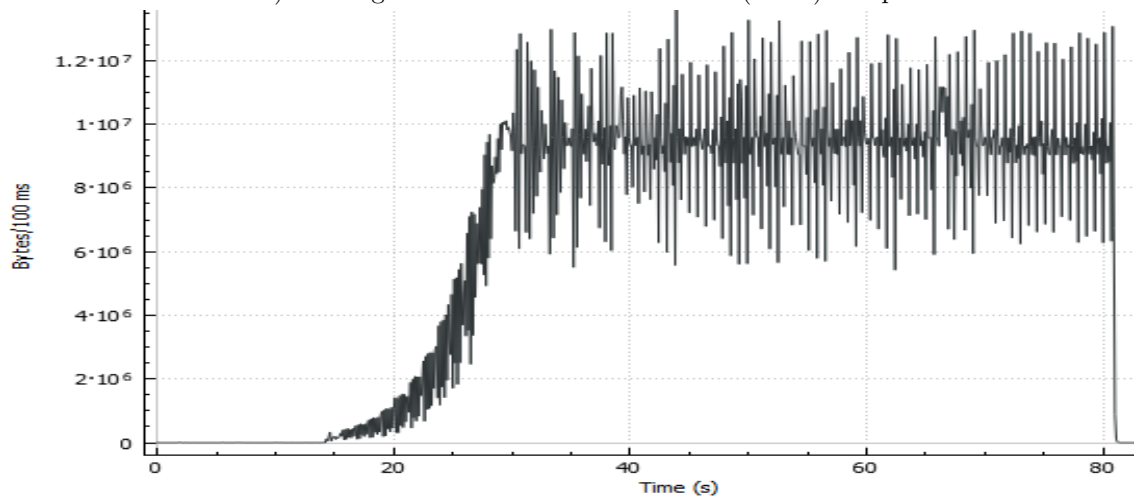
Figure 4.24: Goodput & Packet Loss for Protocols With Various File Sizes (2) - Auckland Setup



a) Sending Rate over Time for GridFTP (200 MB) - Replication 1



b) Sending Rate over Time for GridFTP (1 GB) - Replication 1



c) Sending Rate over Time for GridFTP (5 GB) - Replication 1

Figure 4.25: Sending Rate over Time for GridFTP - Auckland Setup

4.2 Results for Experiments with Background Traffic

4.2.1 Local Testbed Results for Transfers with Background Traffic

Figure 4.26 shows the aggregate goodput of each protocol for all the runs. Except for TCP BBR, there is no noticeable difference between the goodput values of the protocols with TCP and UDP background traffic in the local setup. The goodput of TCP BBR varied and was lower with UDP background traffic because background UDP traffic was aggressive in using the bandwidth.

However, it seems that the background traffic generated did impact the performance of the protocols by reducing their goodput differently compared to their performance in Figure 4.4. GridFTP is hardly affected and is now better than FASP. The behaviour of the protocols is similar to the one observed in Figure 4.4 with TCP BBR having the highest goodput and QUIC with the lowest goodput.

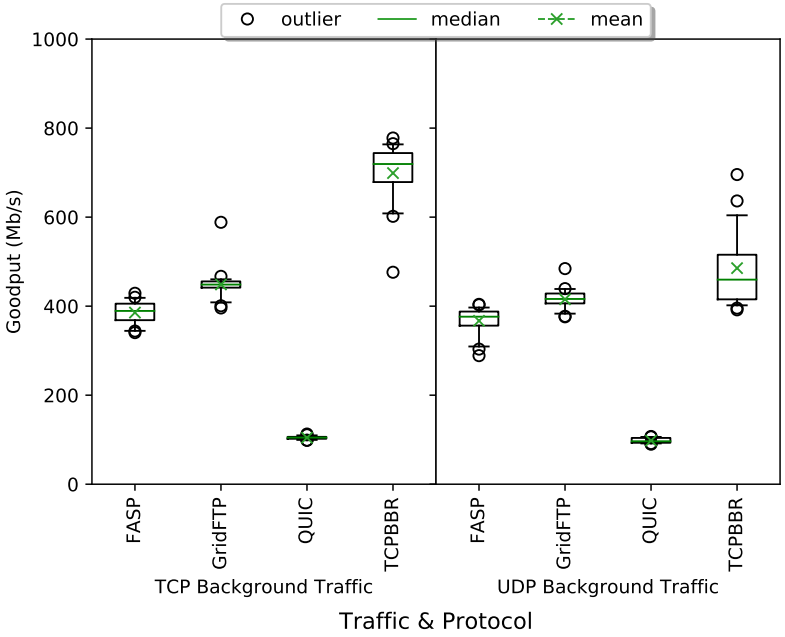


Figure 4.26: Aggregate Goodput for Each Protocol With Background Traffic - Local Setup

Figure 4.27 shows the aggregate packet loss results for each protocol with background traffic. The main observation here is that FASP had high packet loss for both types of background traffic. This is a similar behaviour to experiment 1 for the local setup. As explained earlier in Section 4.1.2, the behaviour of FASP could be caused by the activity on the destination host, preventing it from processing the FASP packets as fast as they arrive.

The other protocols showed similar packet loss results to the ones in the local setup for the first experiment. All the protocols seem to compete with the background traffic for bandwidth, which causes a reduction in the bytes sent when there is packet loss.

Table 4.2 shows the aggregate goodput for TCP and UDP background traffic and how the data transfer

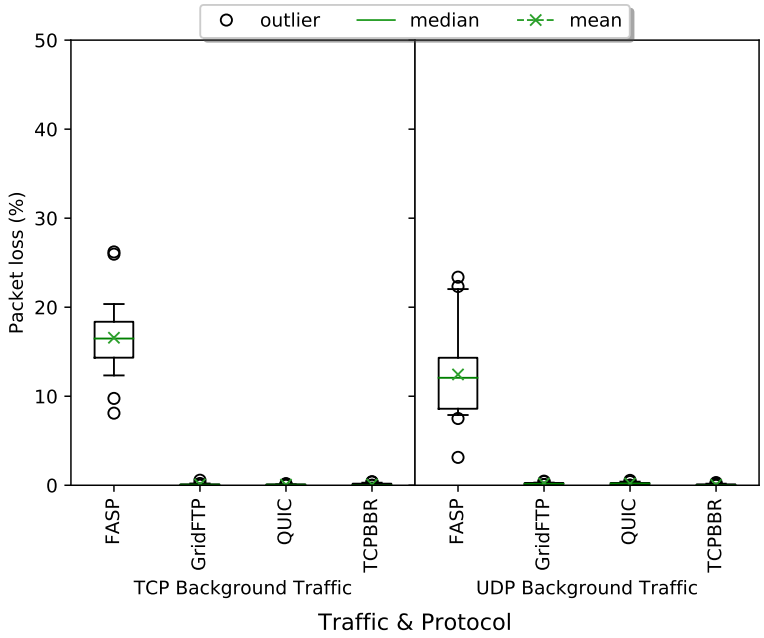


Figure 4.27: Aggregate Packet Loss for Each Protocol with Background Traffic - Local Setup

protocols affect the background traffic. From the results, the goodput of the background TCP traffic is not that different with each protocol used. This means that all the protocols share the bandwidth fairly with the TCP background traffic. On the other hand, the goodput results of the UDP background traffic were impacted by FASP. This is caused by FASP's aggressive behaviour, which tries to utilize more bandwidth on the link.

Table 4.2: Goodput (Mbps) for Background Traffic in Local Setup

Background Traffic		**FASP**	**GridFTP**	**QUIC**	**TCP BBR**
TCP Traffic	Mean	100.9	100.6	100.2	101
	Std. Dev.	2.3	0.2	0.04	0.8
UDP Traffic	Mean	175.8	201.5	201.1	201.8
	Std. Dev.	0.08	0.1	0.05	0.22

Table 4.3 shows the aggregate packet loss for both TCP and UDP background traffic. GridFTP, TCPBBR, and QUIC are more fair with TCP traffic than FASP since the packet loss for TCP traffic is lower in these cases. On the other hand, packet loss for UDP traffic has increased with all the protocols, but mainly with FASP and GridFTP. This could be caused by the aggressiveness of FASP and the use of multiple flows by GridFTP. With the increasing amount of UDP traffic, the packet loss tends to increase, which suggests that the goodput would also decrease.

Figure 4.29 shows the results for goodput and packet loss of each protocol for each run. The results were similar for each replication using FASP and GridFTP. A high variation in the packet loss results was

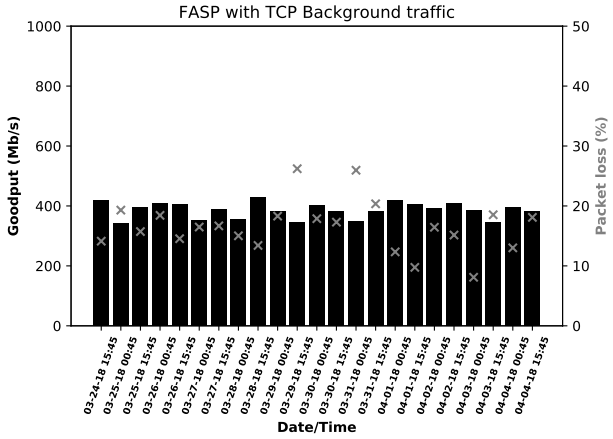
Table 4.3: Packet Loss (%) for Background Traffic in Local Setup

Background Traffic		FASP	GridFTP	QUIC	TCP BBR
TCP Traffic	Mean	0.34	0.11	0.007	0.20
	Std. Dev.	0.27	0.15	0.007	0.31
UDP Traffic	Mean	11.2	9.4	0.9	2.9
	Std. Dev.	7.6	3.2	0.7	1.3

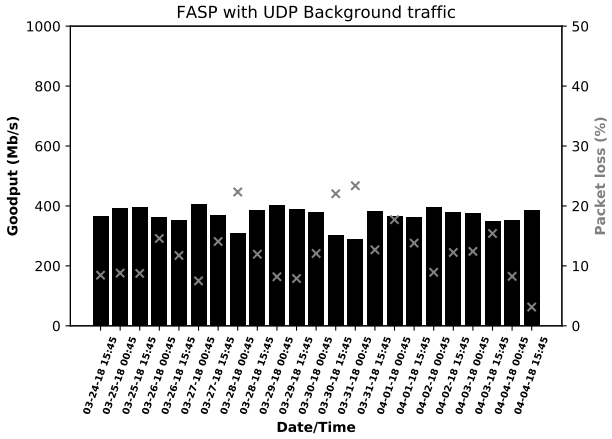
observed in the presence of UDP background traffic for most of the protocols while the goodput results varied more only for FASP and TCP BBR. The reason for BBR's low goodput with UDP traffic is because UDP is aggressive and tries to utilize more available bandwidth. This is potentially because TCP BBR is able to utilize more bandwidth with TCP background traffic since it TCP is fair to other traffic. Additionally, the performance of the protocols in the presence of TCP background traffic seems to be more stable since it uses a congestion control mechanism. Moreover, QUIC also showed similar results for each run, but it was not shown because there was no significant variation in its results.

The goodput of FASP is not affected by packet loss, as some replications would have a high packet loss but the same goodput as replications with low packet loss. This is because FASP uses UDP, which focuses on sending data as fast as possible rather than guaranteeing delivery like TCP. FASP also keeps sending packets and does not slow down when packet loss occurs.

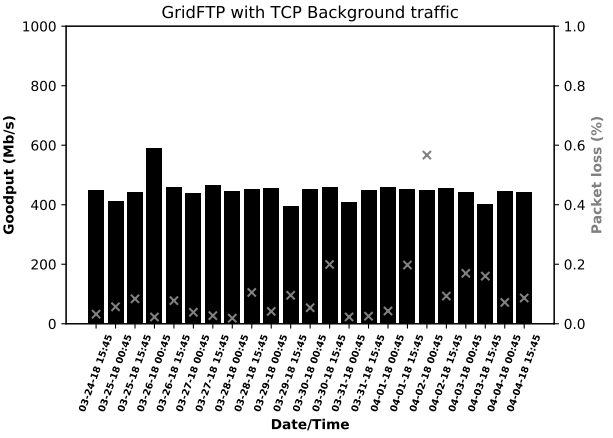
In terms of GridFTP's performance, the goodput results between the replications are consistent even though the packet loss results vary. Although the packet loss for some runs is higher than others, GridFTP is able to tolerate such packet loss and does not decrease its sending rate. GridFTP achieves similar goodput for both types of traffic. In some instances, packet loss is high, but it does not seem to affect the goodput. This is because some of the TCP streams used by GridFTP do not experience high packet loss, which allows GridFTP to still increase its sending rate for these streams. Figure 4.30 and 4.31 shows the throughput over time (replication 17) for each stream used by GridFTP. This confirms what was mentioned earlier that each stream has a different sending rate depending on the packet loss experienced by each stream.



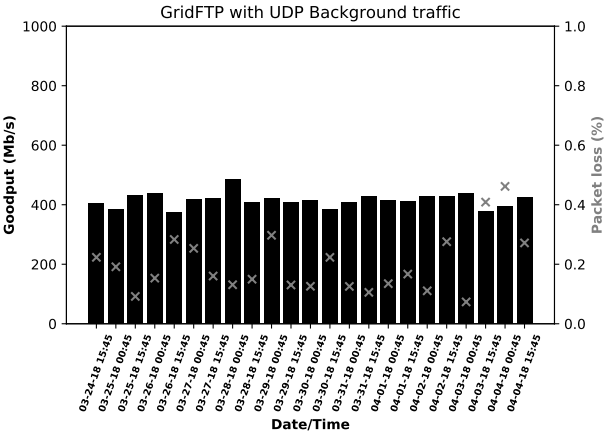
a) Goodput & Packet Loss for FASP with TCP Background Traffic



b) Goodput & Packet Loss for FASP with UDP Background Traffic

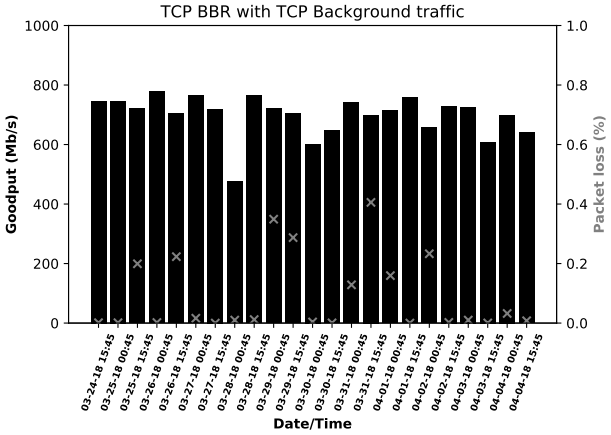


c) Goodput & Packet Loss for GridFTP with TCP Background Traffic

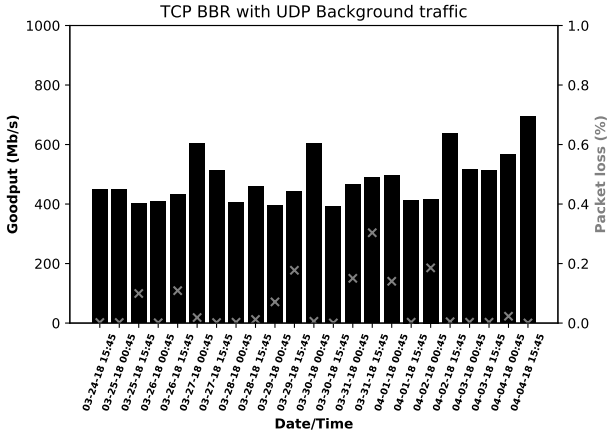


d) Goodput & Packet Loss for GridFTP with UDP Background traffic

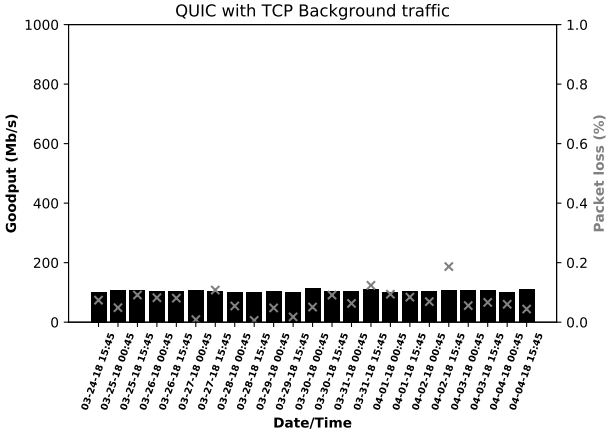
Figure 4.28: Goodput & Packet Loss for protocols with background traffic (1) - Local Setup



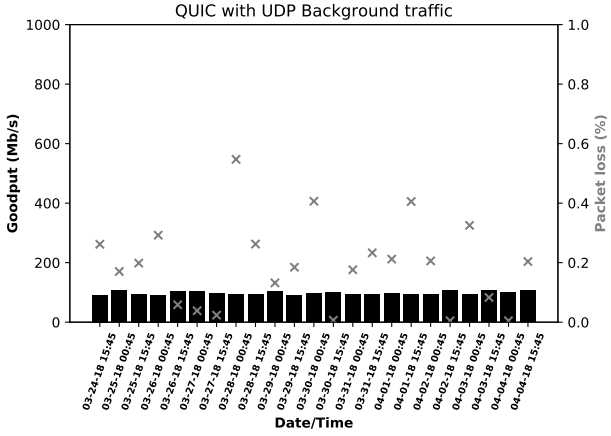
e) Goodput & Packet Loss for TCP BBR with TCP Background traffic



f) Goodput & Packet Loss for TCP BBR with UDP Background traffic

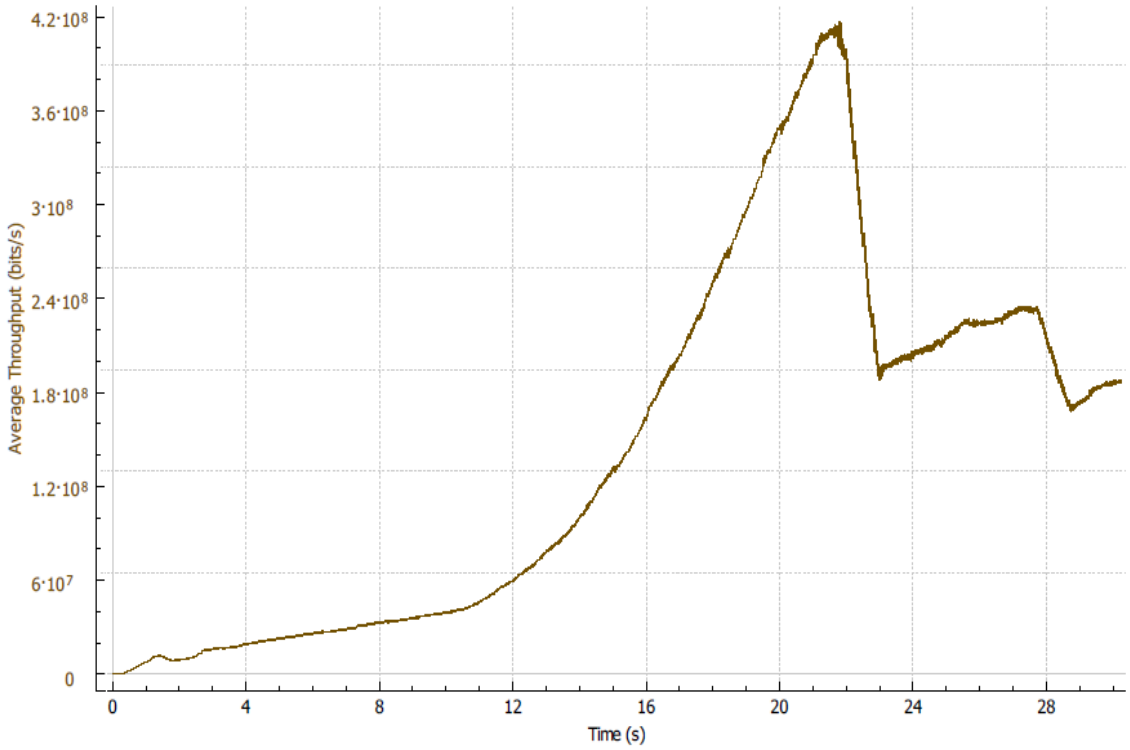


g) Goodput & Packet Loss for QUIC with TCP Background traffic

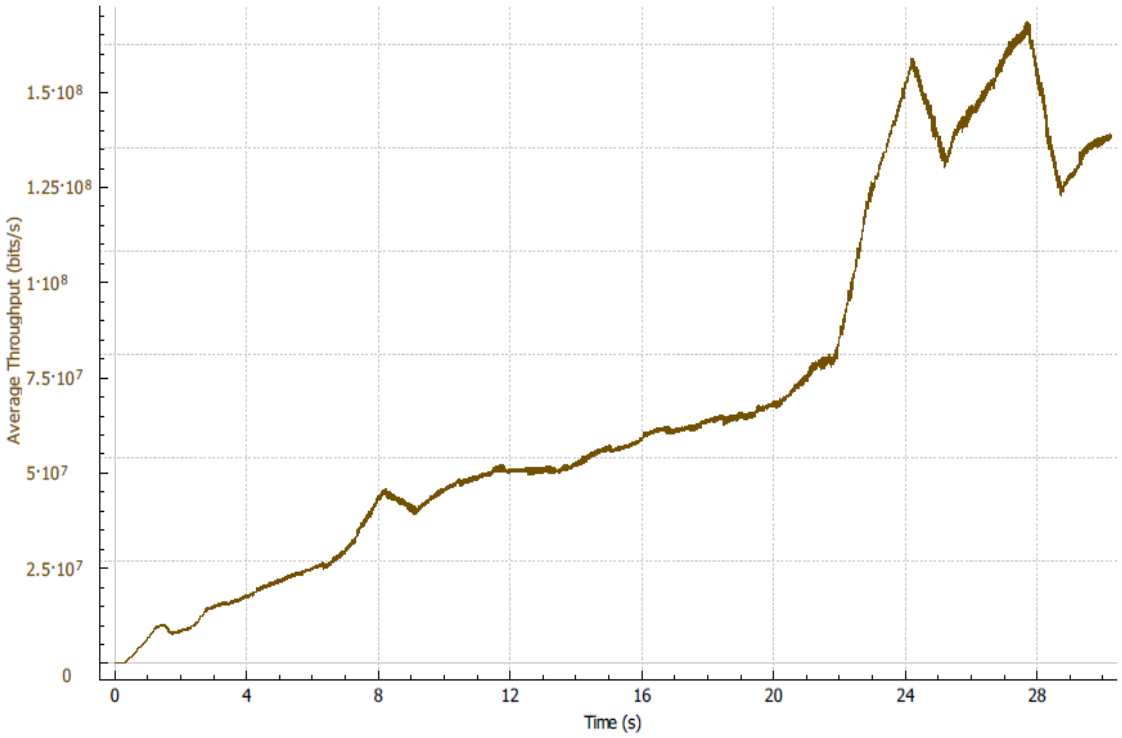


h) Goodput & Packet Loss for QUIC with UDP Background traffic

Figure 4.29: Goodput & Packet Loss for protocols with background traffic (2) - Local Setup

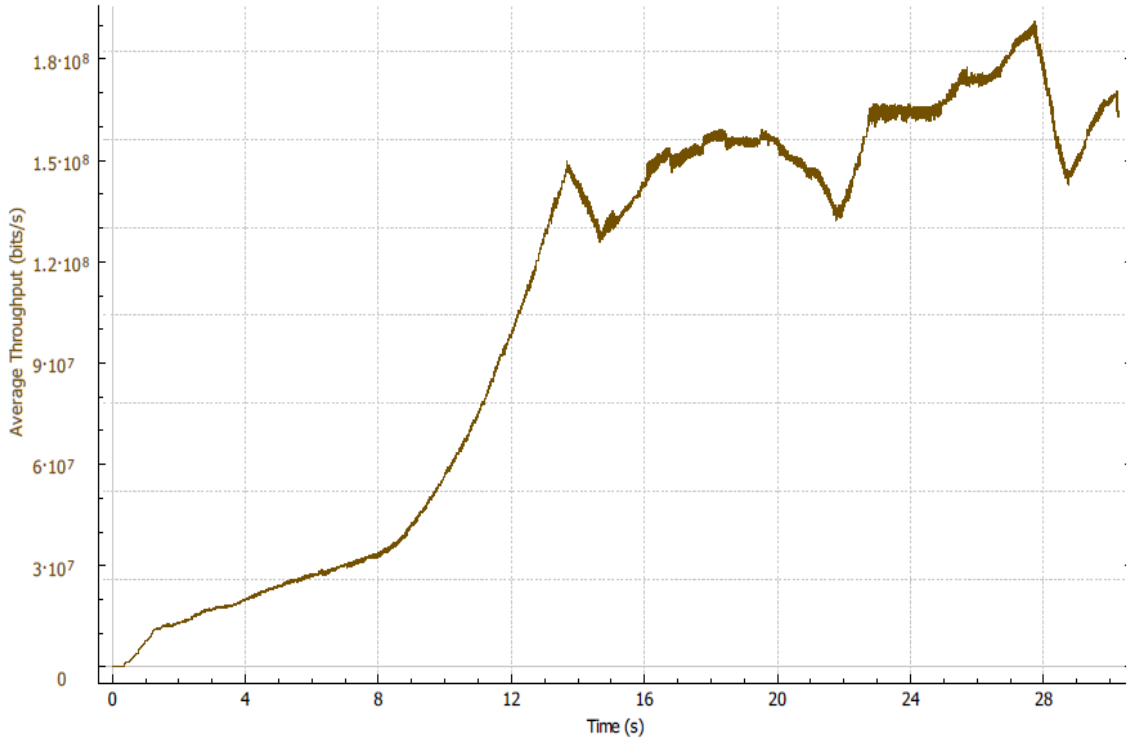


a) Sending Rate Over Time for Stream 1

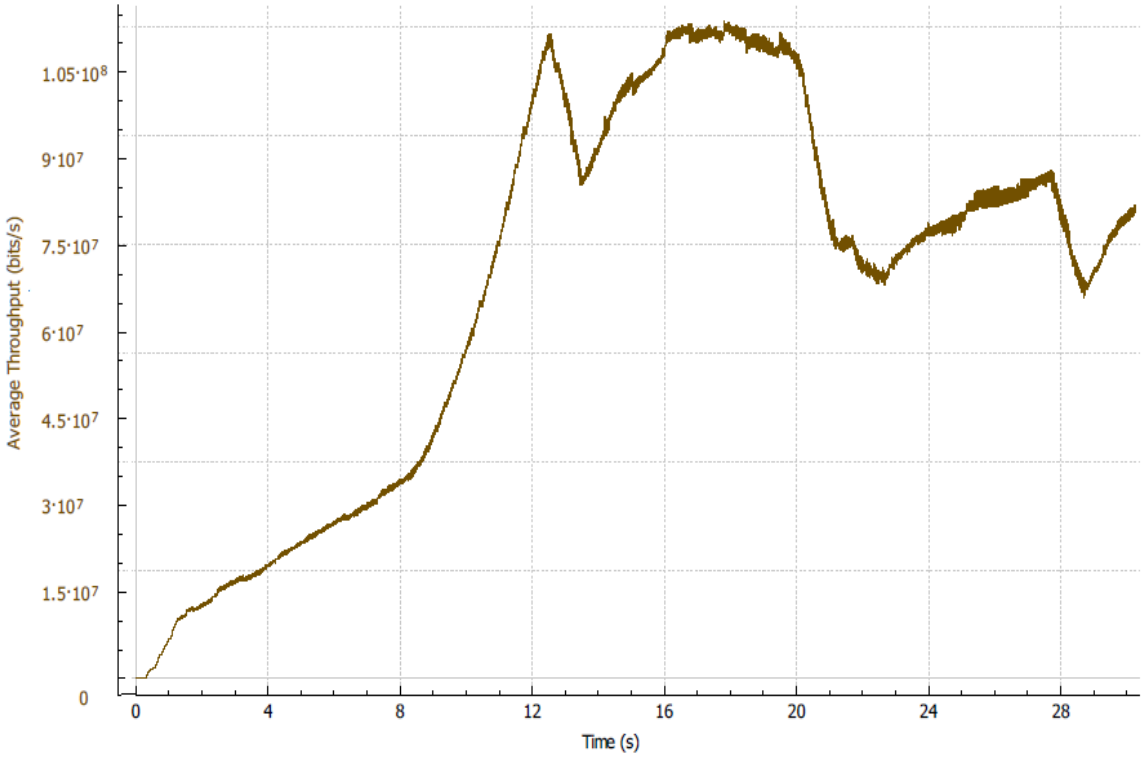


b) Sending Rate Over Time for Stream 2

Figure 4.30: Sending Rate Over Time for All GridFTP Streams - Local Setup -Replication 17 (1)



a) Sending Rate Over Time for Stream 3



b) Sending Rate Over Time for Stream 4

Figure 4.31: Sending Rate Over Time for All GridFTP Streams - Local Setup -Replication 17 (2)

4.2.2 Waterloo Testbed Results for Transfers with Background Traffic

Figure 4.32 shows the aggregate goodput for each protocol in Waterloo. The goodput achieved by all the protocols is again lower than the one observed in the local setup. The behaviour of the protocols in this experiment is similar to that in Section 4.1.2. The aggregate goodput for FASP and GridFTP is similar to each other in this setup, but their goodput is lower than the local setup. Also, the background traffic seems to affect the performance of the protocols, as they compete for bandwidth. The aggregate goodput for FASP and GridFTP is similar with both types of background traffic. The background traffic does not affect the goodput of TCP BBR as much as the other protocols. This is because BBR's sending rate changes depending on the delay and not packet loss. However, TCP BBR still has a low goodput with background UDP traffic similar to the local setup. The performance of QUIC is also similar to that of Figure 4.7.

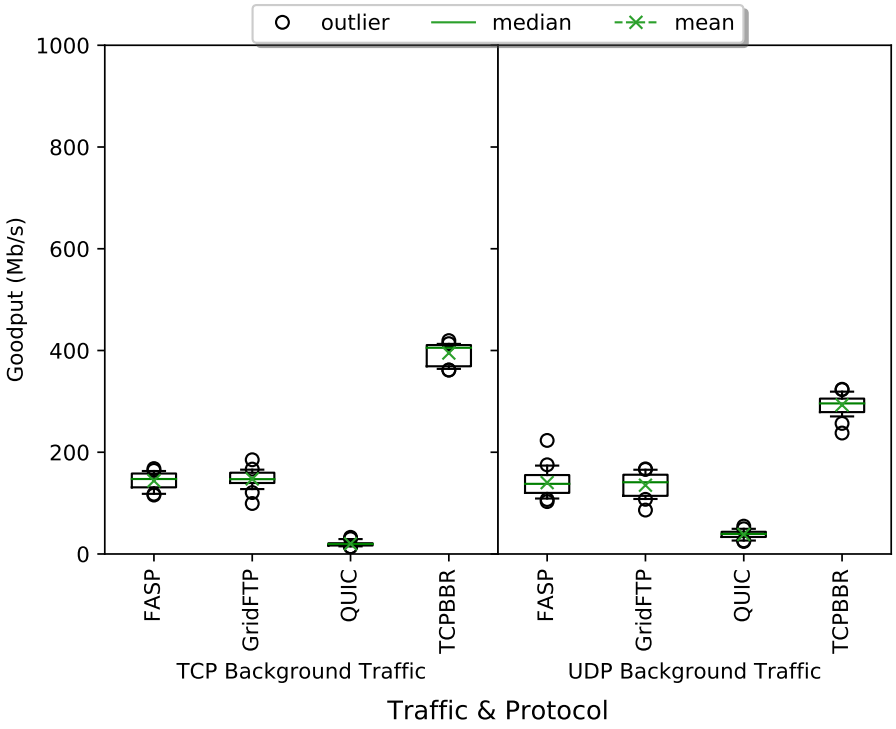


Figure 4.32: Aggregate Goodput for Each Protocol with Background Traffic - Waterloo Setup

Figure 4.33 shows the aggregate packet loss results for each protocol with background traffic. The packet loss of FASP is high with both types of background traffic. The high packet loss percentage in FASP's runs is caused by the firewall in the destination's network, which degrades the goodput of FASP compared to the other setups (see Figure 4.10). The other protocols are also affected by background traffic, especially with background UDP traffic. The background UDP traffic affects the protocols more than TCP, because it tries to utilize more bandwidth, thus unfair to other protocols. In addition, the results for the TCP-based protocols is better than in Figure 4.10.

Table 4.4 shows the aggregate goodput for background TCP and UDP traffic in this setup. The TCP

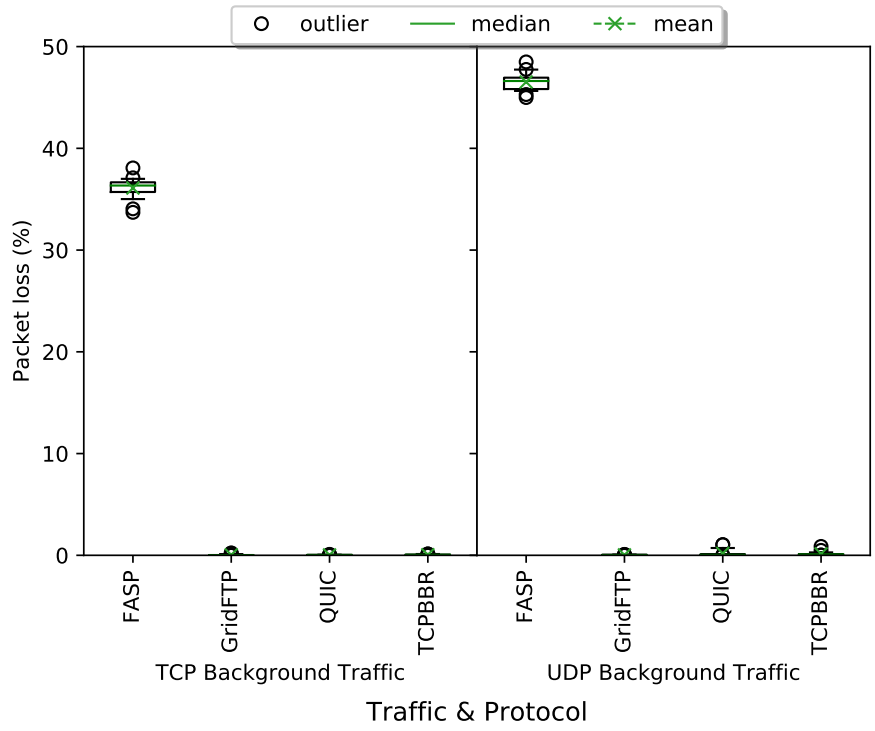


Figure 4.33: Aggregate Packet loss for Each Protocol with Background Traffic - Waterloo Setup

traffic goodput results are very similar with each protocol used. Also, the goodput results of the UDP background traffic do not seem to be impacted by any of the protocols. This means that UDP gets all of its packets through at the expense of the other protocols. The overall results of the background traffic are similar to the local setup.

Table 4.4: Goodput (Mbps) for Background Traffic in Waterloo Setup

Background Traffic		**FASP**	**GridFTP**	**QUIC**	**TCP BBR**
TCP Traffic	Mean	100.2	100.2	100.1	100.2
	Std. Dev.	0.08	0.07	0.05	0.25
UDP Traffic	Mean	201.2	201.2	201.1	200.8
	Std. Dev.	0.04	0.05	0.02	0.07

Table 4.5 shows the aggregate packet loss for both TCP and UDP background traffic. All the protocols used do not cause the TCP background traffic to have high packet loss. On the other hand, packet loss for the UDP background traffic has increased, mainly when using FASP. The reason is how the protocols try to compete for available bandwidth. This causes congestion, and therefore packet loss to the UDP background traffic.

Figures 4.34 and 4.35 show the goodput and packet loss results of each protocol for every individual run. The results are similar between FASP and GridFTP in terms of the goodput achieved. However, FASP has

Table 4.5: Packet Loss (%) for Background Traffic in Waterloo Setup

Background Traffic		FASP	GridFTP	QUIC	TCP BBR
TCP Traffic	Mean	0.03	0.04	0.05	0.07
	Std. Dev.	0.06	0.08	0.12	0.12
UDP Traffic	Mean	6.0	1.5	0.87	1.6
	Std. Dev.	5.96	4.4	1.0	1.3

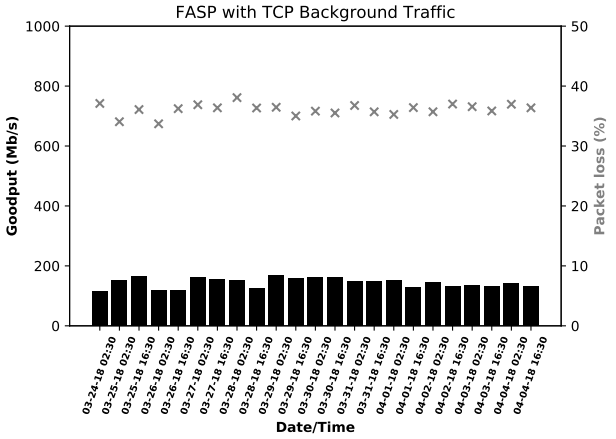
extremely high packet loss for all of the runs compared to the other protocols. This is caused by the firewall, background traffic and the inefficiency of the destination machine.

As observed in Figure 4.31, the performance of the GridFTP and TCP BBR is affected by UDP background traffic more than with TCP background traffic, which is caused by the aggressive nature of UDP. However, TCP BBR is still able to achieve a high goodput compared to the other protocols. The other protocols had a varying goodput for each run while TCP BBR had similar results for each run.

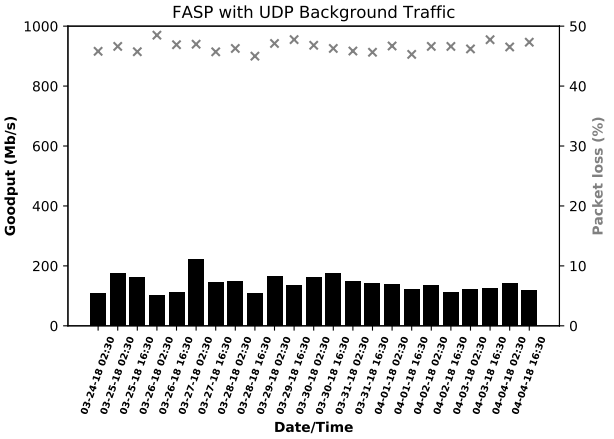
GridFTP showed more stable performance with background TCP traffic than with background UDP traffic. It is not caused by high packet loss, rather it is caused by the destination machine being heavily loaded at the time of the experiment, which would cause TCP to not be able to send more data.

The performance of QUIC for most of the runs varies for both types of traffic, but QUIC is affected more by the TCP background traffic. QUIC might be reducing its goodput with TCP, because it uses CUBIC, which tries to be fair to other TCP flows by reducing QUIC's sending rate. However, it does not reduce its rate with UDP traffic.

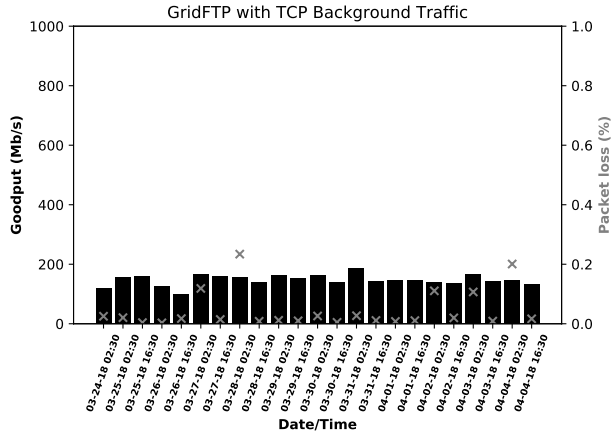
Figure 4.36 shows the sending rate over time for TCP BBR with both types of traffic. As seen in the figure, TCP BBR reaches a high sending rate with the TCP Background traffic, but not with UDP background traffic. This is because TCP BBR is halving its congestion window more often with UDP and this causes more frequent drops in the sending rate. It is also possible that the UDP traffic is causing congestion on the network, which causes more delay and this causes the performance of TCP BBR to degrade.



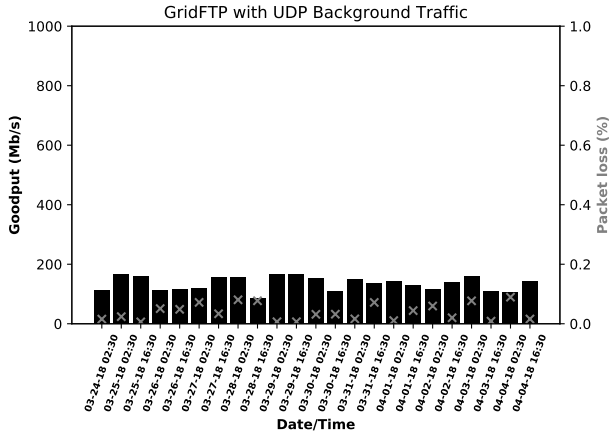
a) Goodput & Packet Loss for FASP with TCP Background Traffic



b) Goodput & Packet Loss for FASP with UDP Background Traffic

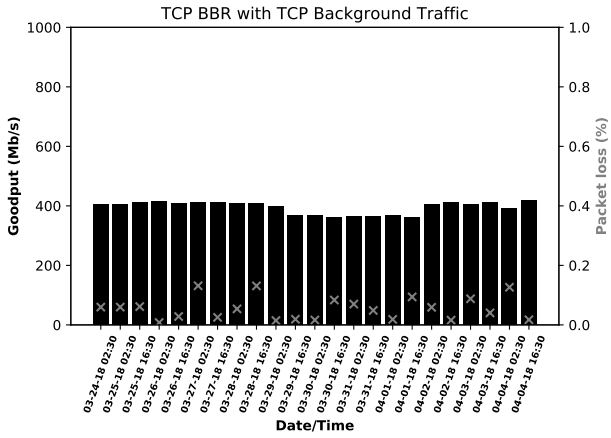


c) Goodput & Packet Loss for GridFTP with TCP Background Traffic

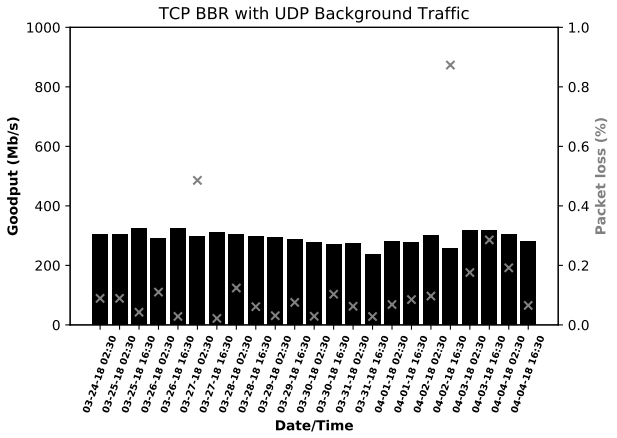


d) Goodput & Packet Loss for GridFTP with UDP Background Traffic

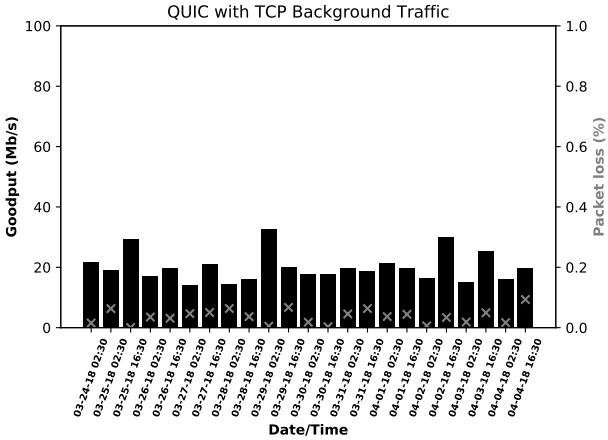
Figure 4.34: Goodput & Packet Loss for FASP with Background Traffic - Waterloo Setup



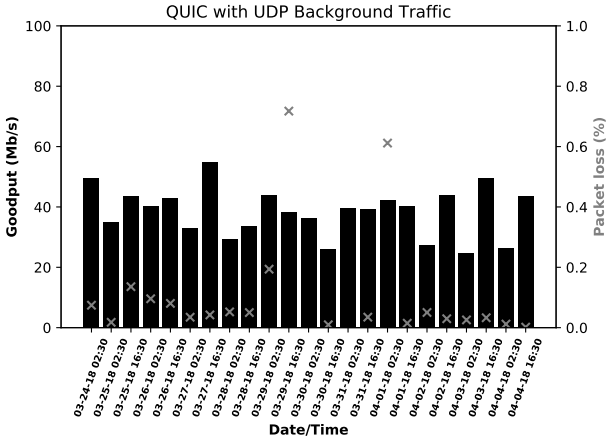
e) Goodput & Packet Loss for TCP BBR with TCP Background Traffic



f) Goodput & Packet Loss for TCP BBR with UDP Background Traffic

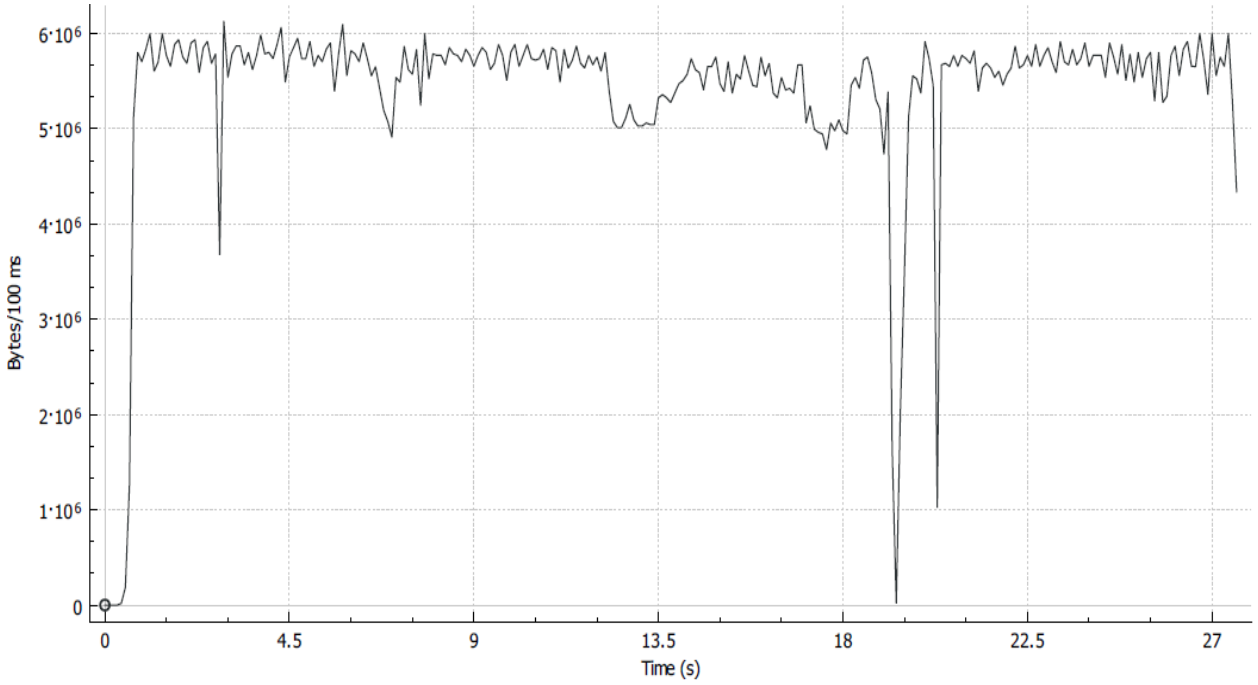


g) Goodput & Packet Loss for QUIC with TCP Background Traffic

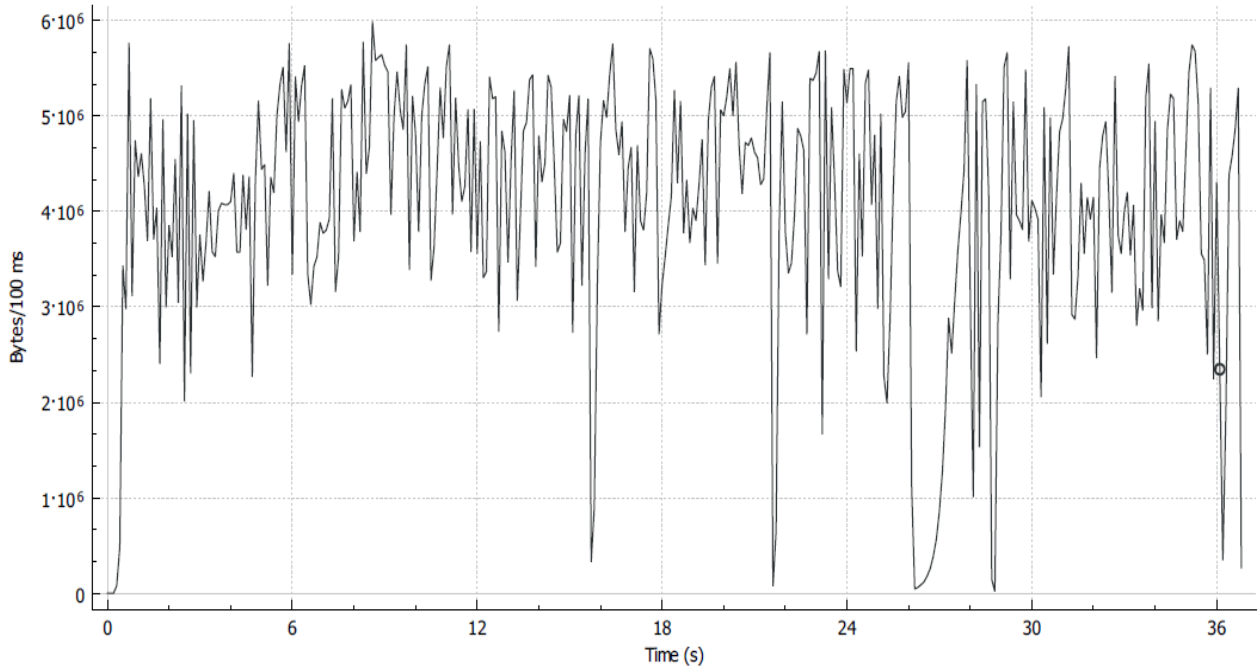


h) Goodput& Packet Loss for QUIC with UDP Background Traffic

Figure 4.35: Goodput & Packet Loss for Protocols with Background Traffic - Waterloo Setup



a) Sending Rate over Time for TCP BBR with TCP Background Traffic - Replication 2



b) Sending Rate over Time for TCP BBR with UDP Background Traffic - Replication 2

Figure 4.36: Sending Rate over Time for TCP BBR with Background Traffic - Waterloo Setup

4.2.3 Eastcloud Testbed Results for Transfers with Background Traffic

Figure 4.37 shows the aggregate goodput for each protocol in this setup. The goodput achieved by all the protocols is lower than in the local setup compared to 4.14. Since there is a longer RTT than the RTT in the local setup, the goodput of the protocols decreases. In addition, the background traffic seems to affect the performance of the protocols. The aggregate goodput for FASP and GridFTP is similar with both types of background traffic. Both of these protocols achieve a goodput more than 300 Mbps, which is higher than in the Waterloo setup. Both FASP and GridFTP had different results for each run, which may be caused by packet loss. FASP and GridFTP with UDP background traffic are identical, from a statistical point of view. The performance of QUIC also degrades with background traffic compared to the Waterloo setup. Since there are different hops in this setup and a higher RTT than in the Waterloo setup, this can cause a difference in the performance of some of the protocols.

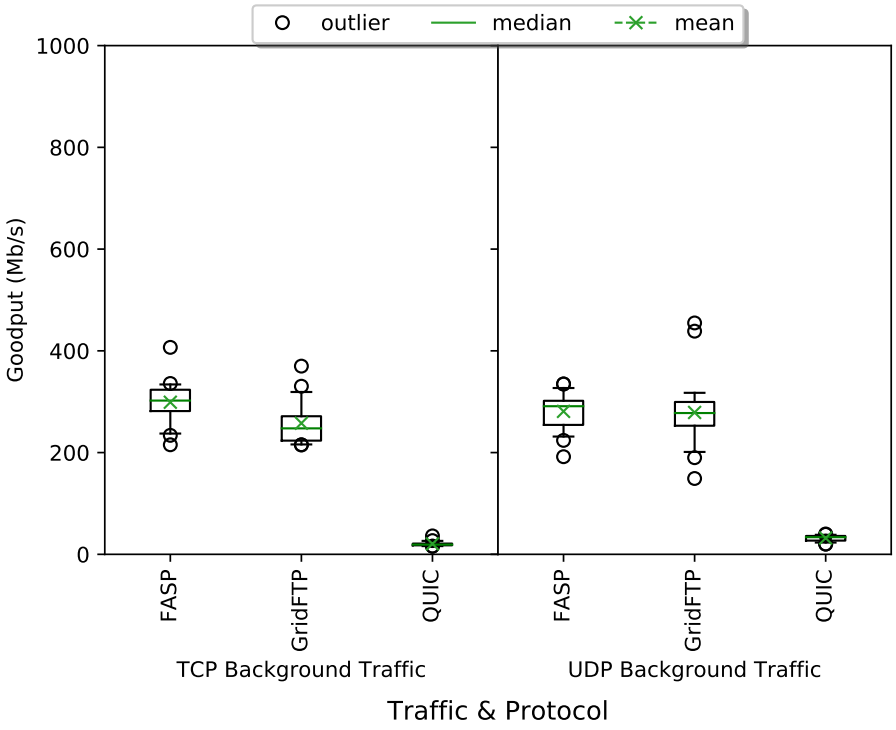


Figure 4.37: Aggregate Goodput for Each Protocol with Background Traffic - Eastcloud Setup

Figure 4.38 shows the aggregate packet loss results for each protocol with background traffic. The packet loss is the highest for GridFTP and is similar with both types of background traffic. This could be caused by GridFTP's multiple flows, which could be aggressive and may cause further congestion. Also, GridFTP had a higher packet loss with TCP background traffic compared to the UDP background traffic. FASP showed some packet loss, but it was low compared to GridFTP. FASP is more resilient to packet loss in the presence of background traffic on high RTT links and is able to utilize a high sending rate. FASP also had a higher packet loss with UDP background traffic than with TCP background traffic, which could be also caused by

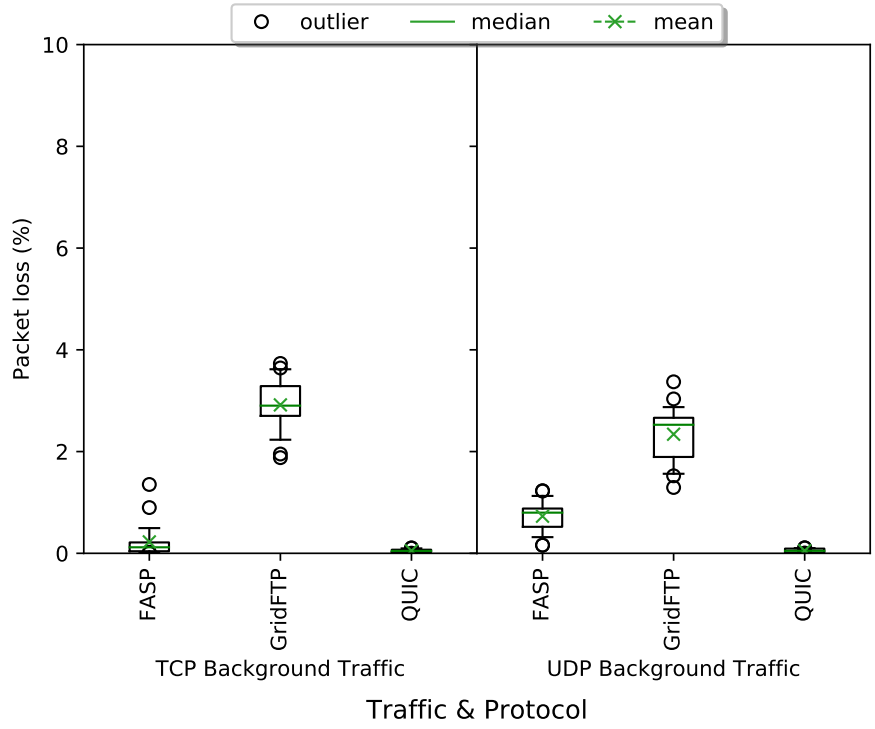


Figure 4.38: Aggregate Packet Loss for Each Protocol with Background Traffic - Eastcloud Setup

congestion induced from the UDP background traffic. Furthermore, QUIC showed a similar behaviour to previous setups.

Table 4.6 shows the aggregate goodput for TCP and UDP background traffic, respectively. The goodput of the TCP background traffic has decreased when FASP was used. This means that FASP was being more aggressive than the TCP background traffic on the link. On the contrary, the aggregate goodput of the background UDP traffic does not seem to be impacted by any of the protocols' traffic in this setup. It ignores packet loss, but does not seem to have experienced much. Moreover, the TCP background traffic has lower goodput than that of UDP traffic, and this is lower than the previous setups.

Table 4.6: Goodput (Mbps) for Background Traffic in Eastcloud Setup

Background Traffic		**FASP**	**GridFTP**	**QUIC**
TCP Traffic	Mean	64.9	98.7	103
	Std. Dev.	21.8	12.2	2.3
UDP Traffic	Mean	201.4	201.3	201.1
	Std. Dev.	0.10	19.3	0.01

Table 4.7 shows the aggregate packet loss for both TCP and UDP background traffic in this setup. The packet loss for both types of traffic is not high. However, the packet loss for TCP traffic is higher than UDP traffic, which also explains why the goodput for TCP was lower than the goodput of the UDP traffic.

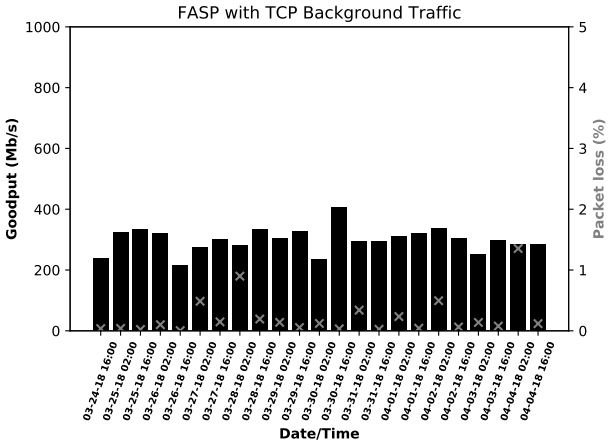
Table 4.7: Packet Loss (%) for Background Traffic in Eastcloud Setup

Background Traffic		FASP	GridFTP	QUIC
TCP Traffic	Mean	2.1	1.52	2.34
	Std. Dev.	0.36	1.09	0.63
UDP Traffic	Mean	1.50	1.62	0.32
	Std. Dev.	0.63	0.82	0.41

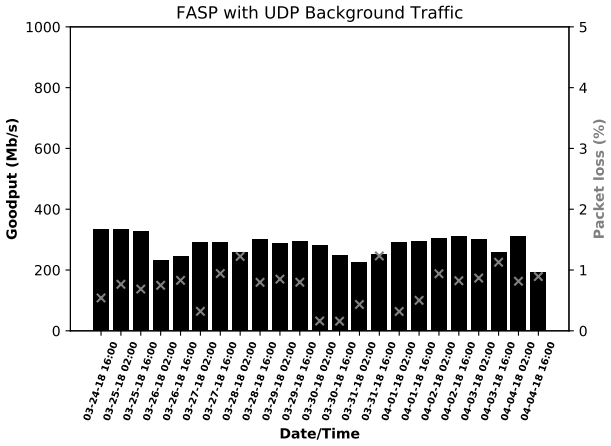
This is not the case with GridFTP, as the packet loss for UDP traffic is slightly greater than that of the TCP traffic. The background UDP traffic might be aggressively trying to utilize more bandwidth, thus causes more congestion on the link.

Figure 4.39 shows the goodput and packet loss results of each protocol for every individual run. For most of the runs, the goodput and packet loss results are not as similar as the results the local setup. The protocols are affected by different network characteristics such as RTT and available bandwidth on the link. The results are similar between FASP and GridFTP, but FASP has marginally higher goodput for most of the runs than GridFTP. In addition, GridFTP has higher packet loss for most of the runs than FASP and QUIC. As explained in the local setup, the goodput of GridFTP can decrease because the destination machine could be heavily overloaded with traffic at the time of the transfer. Additionally, the goodput of GridFTP can increase even if there is high packet loss, because not all the TCP streams used by GridFTP decrease their sending rate. This shows that not all the TCP streams experience high packet loss. This is observed in Figure 4.24 in the local setup.

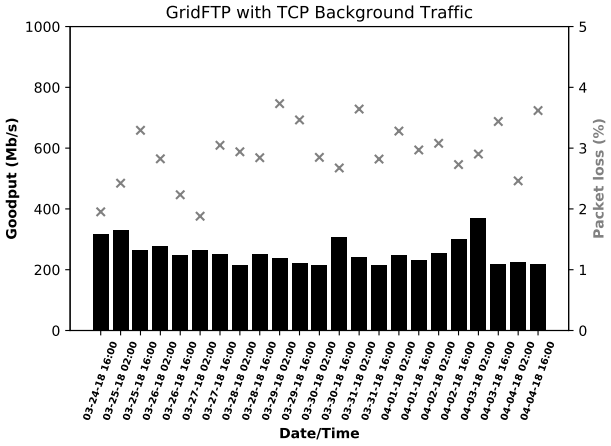
FASP does not have high packet loss in this setup compared to the local and Waterloo setups. The destination machine is able to keep up with packets sent by FASP. Moreover, QUIC also showed different results for each run, but the results were similar for each run with TCP background traffic. For UDP background traffic, QUIC results were different for each run, but had high goodput results compared to that with background TCP traffic. In addition, QUIC's performance varies more with UDP traffic, because of the varying network conditions and UDP's aggressive behaviour.



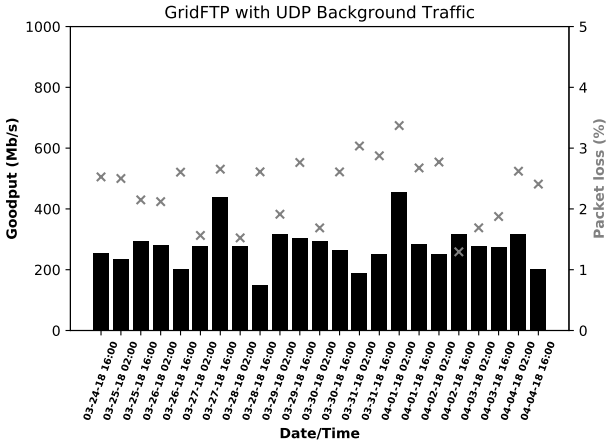
a) Goodput & Packet Loss for FASP with TCP Background Traffic



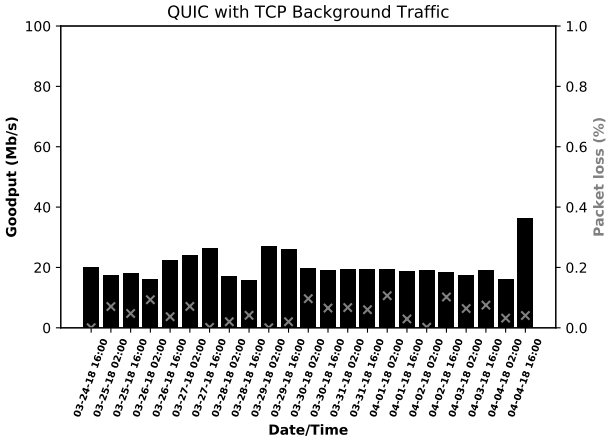
b) Goodput & Packet Loss for FASP with UDP Background Traffic



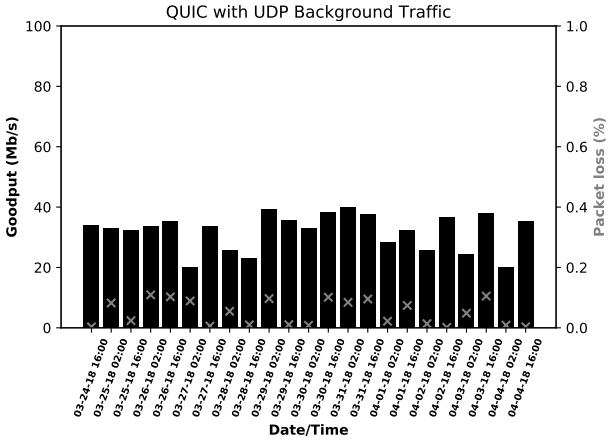
c) Goodput & Packet Loss for GridFTP with TCP Background Traffic



d) Goodput & Packet Loss for GridFTP with UDP Background Traffic



e) Goodput & Packet Loss for QUIC with TCP Background Traffic



f) Goodput & Packet Loss for QUIC with UDP Background Traffic

Figure 4.39: Goodput & Packet Loss for Protocols with Background Traffic - Eastcloud Setup

4.2.4 International Testbed Results for Transfers with Background Traffic

Figure 4.40 shows the aggregate goodput for each protocol in this setup. The aggregate goodput for the all the protocols is similar with both types of background traffic, except for FASP. FASP had the highest aggregate goodput and the highest variability for the all runs among the other protocols. FASP performs well and is able to utilize more bandwidth on this link, especially with TCP background traffic. However, the goodput results of FASP with TCP background traffic has varied substantially between the runs, which suggests that not all the runs are able to achieve a high sending rate. The goodput achieved by GridFTP was similar to the Waterloo setup and lower than the other setups. Compared to Figure 4.20, the goodput of the protocols has slightly reduced, but is still similar for most of the runs to the results in Figure 4.20. On the other hand, the goodput of TCP BBR and QUIC was below 100 Mbps; they are affected by delay and packet loss on the long RTT path.

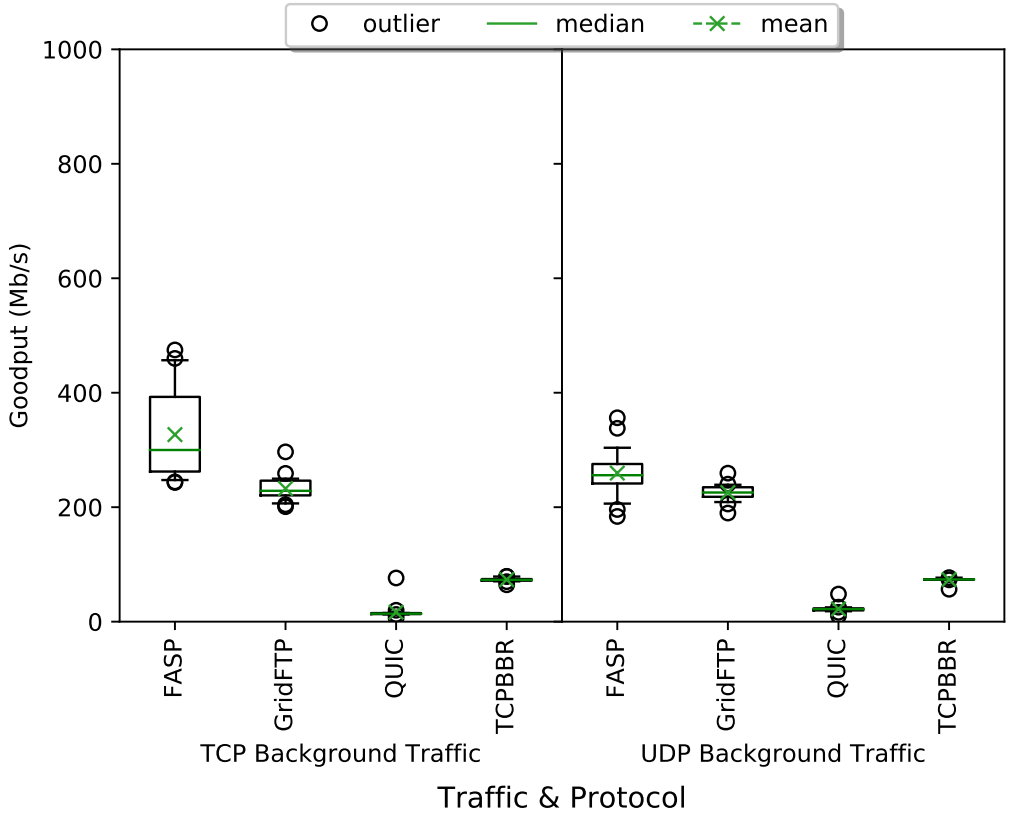


Figure 4.40: Aggregate Goodput for Each Protocol with Background Traffic - Auckland Setup

Figure 4.41 shows the aggregate packet loss results for each protocol with background traffic in this setup. All the protocols have similar aggregate packet loss values with GridFTP having the highest packet loss. As observed earlier in Eastcloud, the goodput decreased in some cases, even though packet loss was low. This is a similar behaviour and is potentially caused by the high delay. Another reason for this behaviour could be the longer route (higher propagation delay) to destination. All the protocols seem to have similar

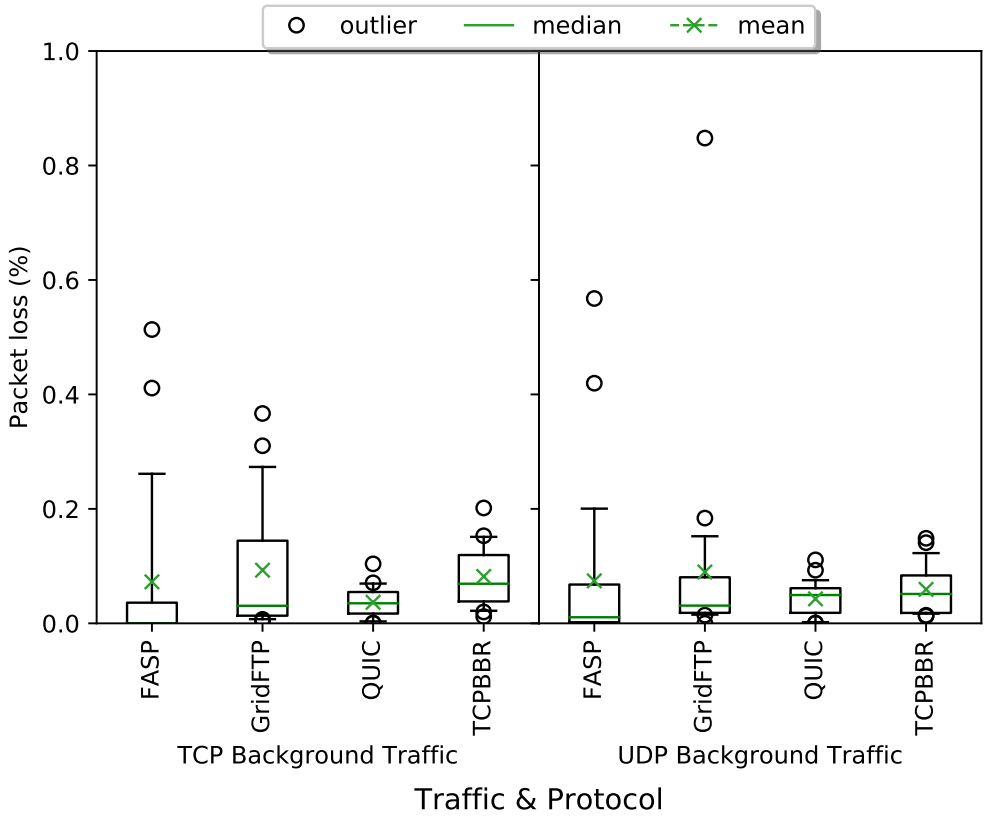


Figure 4.41: Aggregate Packet loss for Each Protocol with Background Traffic - Auckland Setup

packet loss values for both types of the background traffic. Furthermore, the packet loss for all the protocols did not exceed 1%, which is very low.

Table 4.8 shows the aggregate goodput for background TCP and UDP traffic in this setup. The background TCP and UDP traffic goodput results are similar with each protocol used. This shows that all background traffic gets to the destination machine, but at the expense of the other protocol's goodput. The background traffic is able to fully achieve the target transfer rate.

Table 4.8: Goodput (Mbps) for Background Traffic in Auckland Setup

Background Traffic		**FASP**	**GridFTP**	**QUIC**	**TCP BBR**
TCP Traffic	Mean	100.9	100.6	98.9	100.5
	Std. Dev.	3.02	5.6	4.8	0.2
UDP Traffic	Mean	201.4	201.3	201.1	200.7
	Std. Dev.	0.07	0.03	0.003	0.01

Table 4.9 shows the aggregate packet loss for both TCP and UDP background traffic. The packet loss for the TCP background traffic is almost zero when each protocol is used as competing traffic. On the other hand, packet loss for the UDP background traffic is the highest with GridFTP. The reason for that is possibly

Table 4.9: Packet Loss (%) for Background Traffic in Auckland Setup

Background Traffic		FASP	GridFTP	QUIC	TCP BBR
TCP Traffic	Mean	0.20	0.20	0.23	0.08
	Std. Dev.	0.47	0.40	0.59	0.05
UDP Traffic	Mean	0.59	5.08	0.71	1.5
	Std. Dev.	0.94	2.33	2.25	0.15

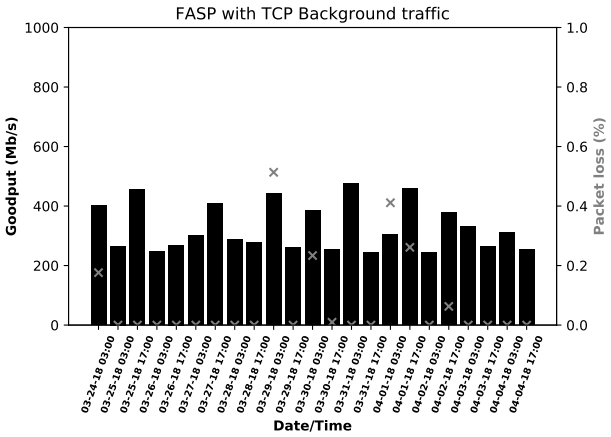
related to GridFTP being aggressive. GridFTP is may be congestion by utilizing a high sending rate and thus more UDP packets are lost.

Figures 4.42 and 4.43 show the goodput and packet loss results of each protocol for every individual run. As can be seen, the performance of FASP and GridFTP over several days varied more with TCP background traffic compared with UDP background traffic, but the performance was more consistent with UDP background traffic. For UDP background traffic, GridFTP also showed similar results between all the runs.

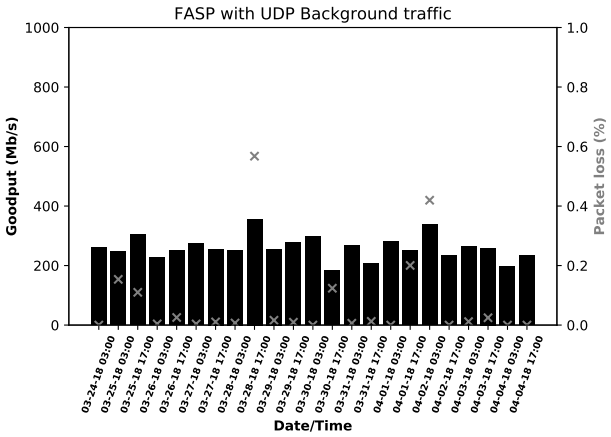
FASP is able to reach a high goodput for several runs with TCP background traffic. However, the goodput of FASP decreased with background UDP traffic. In terms of packet loss, FASP achieves low packet loss in this setup. This shows that it is more resilient to packet loss on the high RTT links compared to previous setups, even if background traffic exists.

The performance of QUIC is poor on high RTT links, which could be caused by the high amount of delay on this link. Since it is also exposed to more cross traffic on this international link, it will try to be fair to other TCP flows, and this might affect its performance even further. QUIC shows consistent results for packet loss.

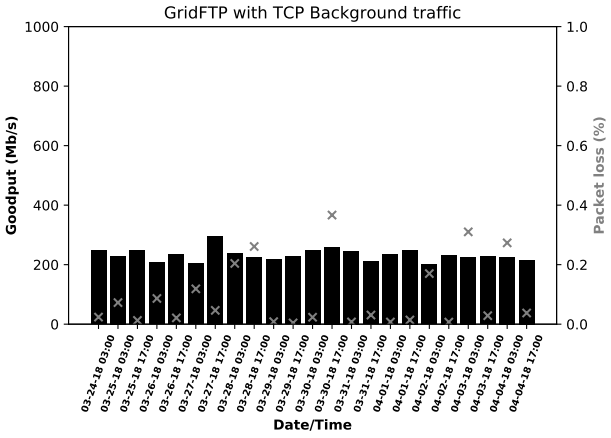
As explained earlier in Section 4.2.4, TCP BBR reduced its sending rate in the presence of both types of background traffic and is affected by the high delay on this link. This is similar to its behaviour for experiment 1 in the Auckland setup. On the other hand, TCP BBR showed very consistent results for goodput with both types of background traffic.



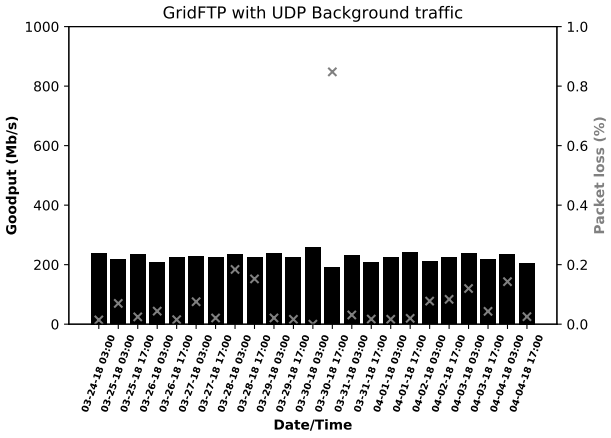
a) Goodput & Packet Loss for FASP with TCP Background Traffic



b) Goodput & Packet Loss for FASP with UDP Background Traffic

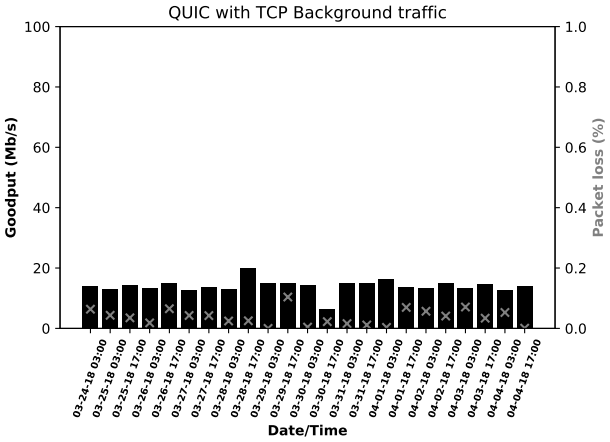


c) Goodput & Packet Loss for GridFTP with TCP Background Traffic

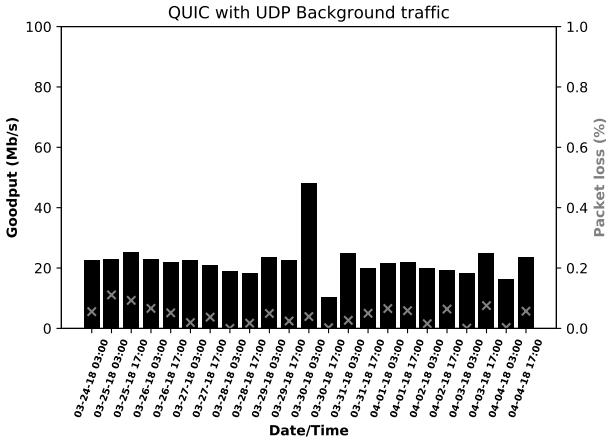


d) Goodput & Packet Loss for GridFTP with UDP Background Traffic

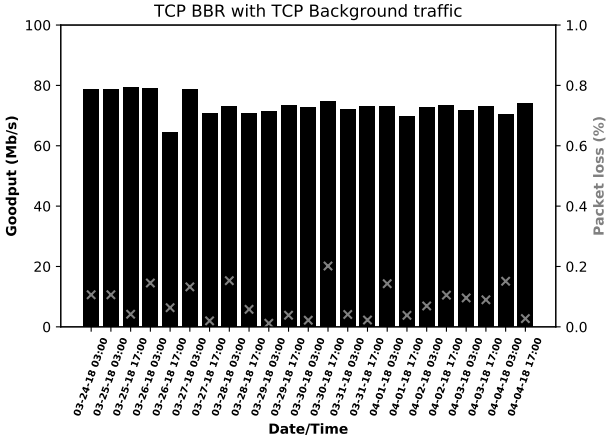
Figure 4.42: Goodput & Packet Loss for protocols with Background Traffic (1) - Auckland Setup



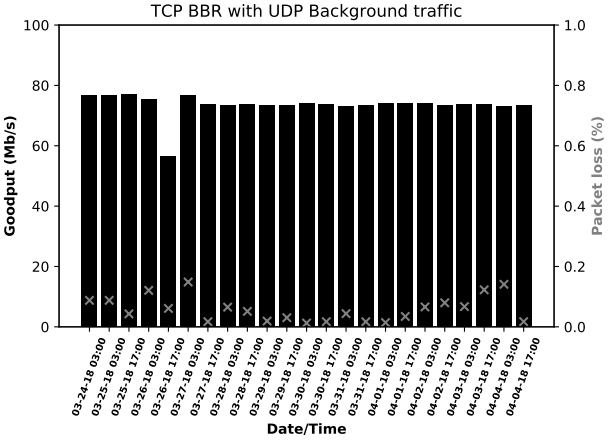
c) Goodput & Packet Loss for QUIC with TCP Background Traffic



d) Goodput & Packet Loss for QUIC with UDP Background Traffic



e) Goodput & Packet Loss for TCP BBR with TCP Background Traffic



f) Goodput & Packet Loss for TCP BBR with UDP Background Traffic

Figure 4.43: Goodput & Packet Loss for protocols with Background Traffic (2) - Auckland Setup

4.3 Results for Experiments with Multiple Flows

4.3.1 Local Testbed Results for Multiple file Transfers

Figure 4.44 shows aggregate goodput of each protocol using 2, 4, and 6 transfer flows. The goodput for all the protocols increases as the number of flows increases. As the number of flows increases, the congestion window for each TCP-based flow would be the same, which increases the aggregate sending rate and so the goodput. Therefore, the goodput of GridFTP and TCP BBR is high. GridFTP was able to achieve a goodput of 800 Mbps for both 4 and 6 flows. This behaviour was also observed in Figure 4.25.

TCP BBR achieved a goodput above 800 Mbps for 2 flows and above 900 Mbps for both 4 and 6 flows, which is close to the maximum bottleneck sending rate. The aggregate goodput achieved by TCP BBR is higher than single flow in the same setup.

The goodput for FASP increases slightly as the number of flows increases. The sending rate utilized by each flow varied between 160 Mbps and 163 Mbps for each flow. This adds to nearly the full link bandwidth using 6 flows. As observed in Figure 4.18, FASP achieves the target sending rate after 5 seconds, which is the same in this experiment as each flow transfers a 1 GB file with at the highest rate possible for most of the transfer. Therefore, FASP shows that it is able to achieve similar results for each number of flows used. In addition, FASP does probe for more bandwidth with multiple flows, but it is unable to utilize more bandwidth so packet loss keeps increasing, and the goodput decreases.

With multiple transfer flows, the goodput of QUIC reached an aggregate goodput between 80 and 90 Mbps, which is lower than that of the other protocols. This is because QUIC takes more time to initiate its transfer and to send the data in chunks.

Figure 4.45 shows the aggregate packet loss of each protocol using multiple transfer flows. With multiple flows, FASP had the highest packet loss and this is possibly caused by the destination host could have been overloaded was not able to process FASP packets fast enough. This high traffic causes congestion to occur on the receiver's network interface card. Also, each flow of FASP tried to achieve the target transfer rate of 600 Mbps, but was aggressive and caused more packet loss.

For TCP-based flows such as GridFTP and TCP BBR, a single flow also had lower packet loss than multiple flows. This is because the flows compete with each other and experience congestion-related packet loss at the same time. Thus, multiple flows take a long time to recover congestion window compared to a single flow. The presence of multiple flows increases congestion as the overall sending rate increases, therefore the available bandwidth also gets heavily used. In some cases, packet loss can also occur when multiple flows increase their congestion window at the same time, and thus experience correlated packet loss. This is observed in Figure 4.46, where two GridFTP flows experience packet loss at the same time, which causes each flow to reduce its sending rate. By limiting the congestion window, TCP can reduce the amount of congestion on the link produced by multiple flows. The performance of QUIC does not seem to be affected

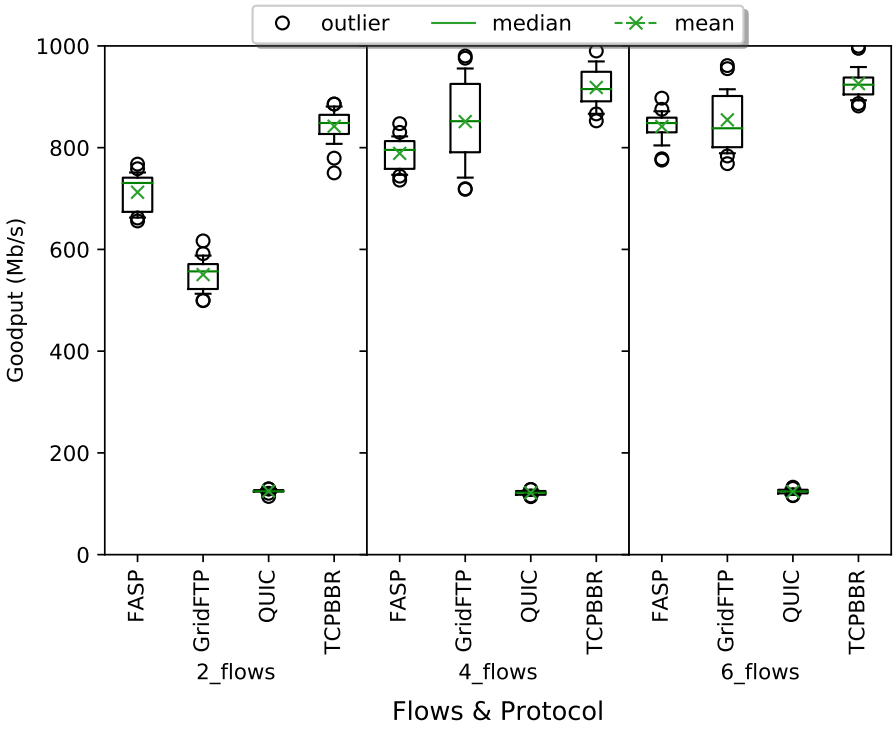


Figure 4.44: Aggregate Goodput for Each Protocol with Multiple Flows - Local Setup

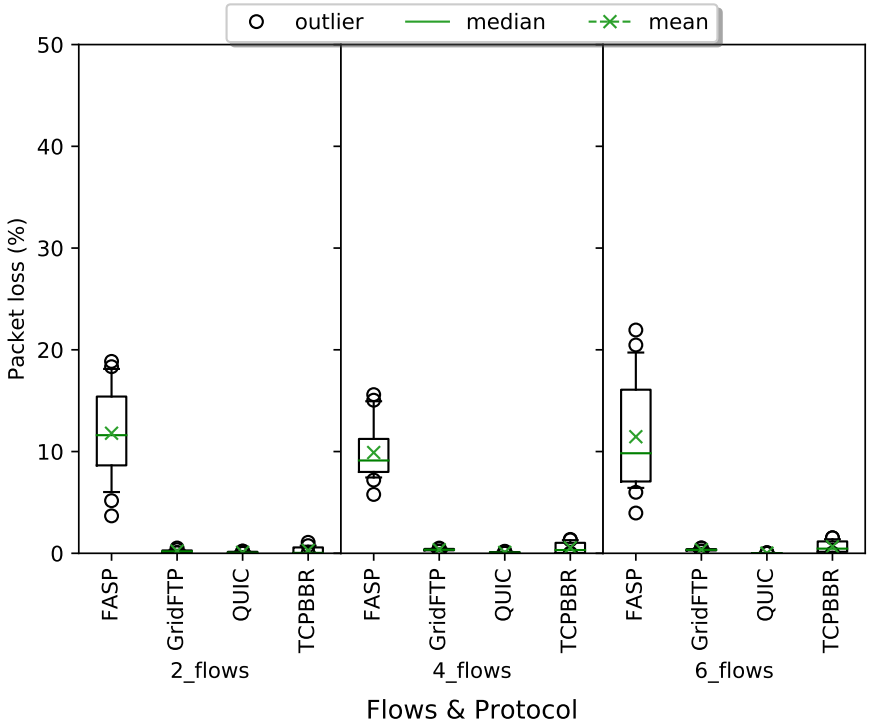


Figure 4.45: Aggregate Packet Loss for Each Protocol with Multiple Flows - Local Setup

by multiple flows, therefore no high packet loss is observed.

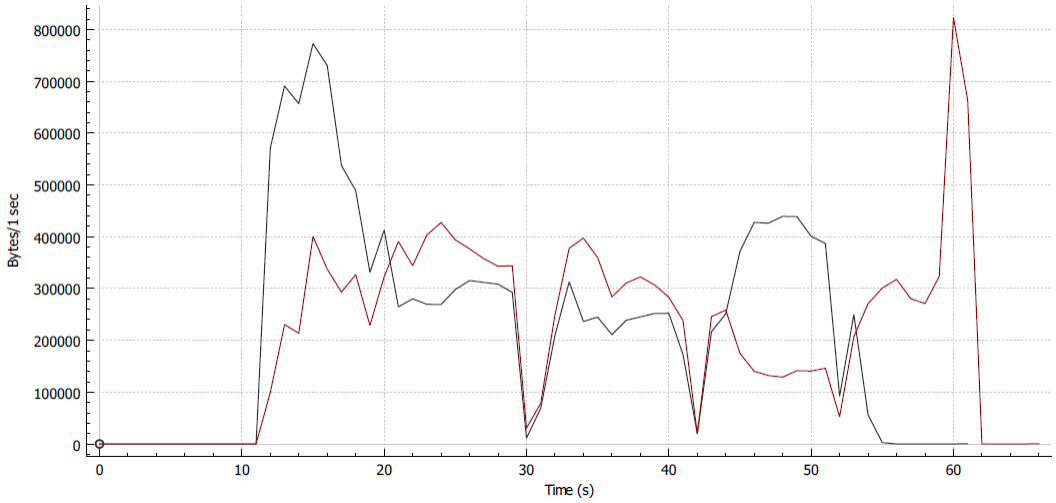
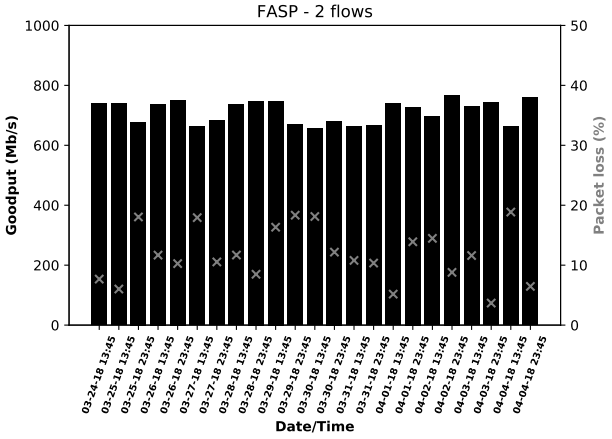


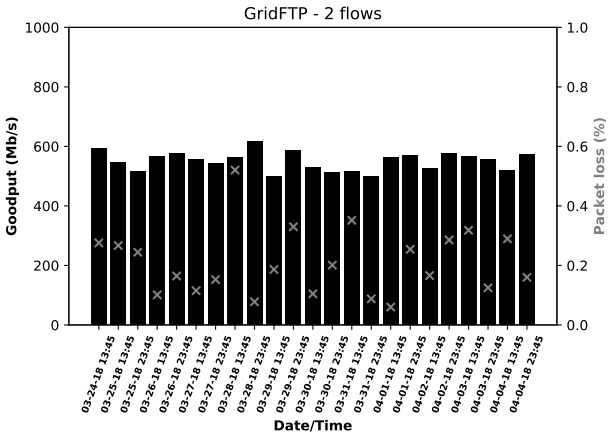
Figure 4.46: Sending Rate Over Time for 2 flows of GridFTP

Figure 4.47 and 4.48 show the goodput and packet loss results for every run of each protocol in this experiment. In terms of consistency, the goodput results for FASP, QUIC, and TCP BBR were similar for most of the runs. They are able to produce similar goodput results for every replication and they are more stable with multiple flows in the local setup. In terms of packet loss, all the protocols show different packet loss values for every run, which shows that different types of congestion are observed for the different replications. FASP had the highest packet loss and it varies with different types of flows as shown in Figure 4.48. The goodput results for FASP are similar between the runs, which is a sign of stable performance by FASP.

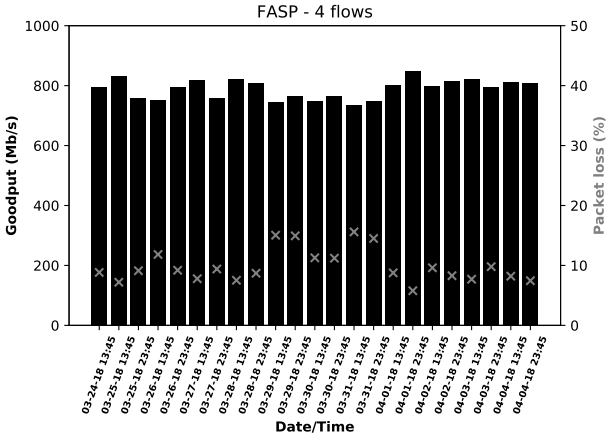
Moreover, the goodput results of GridFTP varied from one run to another and was the least stable among all the protocols, especially with 4 flows and 6 flows. This might be caused by each flow achieving different sending rates and their response to the varying congestion and available bandwidth for each replication. In terms of packet loss, GridFTP showed low packet loss for all the replications, which does not heavily impact the goodput. The goodput of QUIC was consistent among all the replications and the packet loss was very low, which shows that QUIC is stable in this setup.



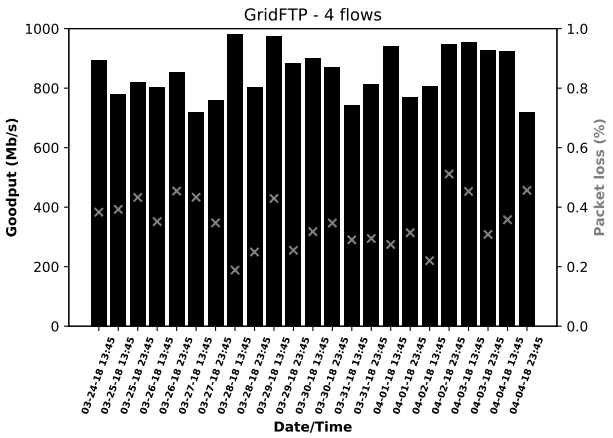
a) Goodput & Packet Loss for FASP with 2 Flows



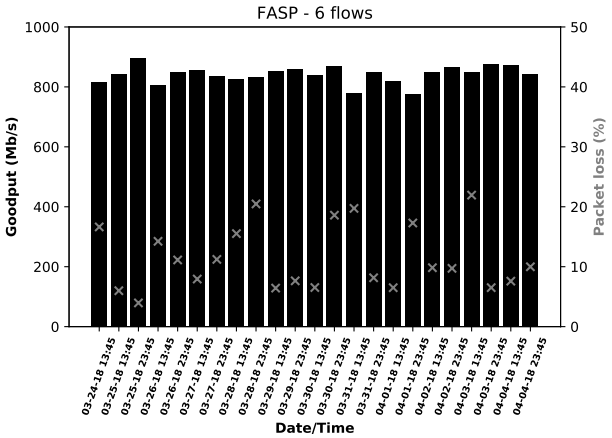
b) Goodput & Packet Loss for GridFTP with 2 Flows



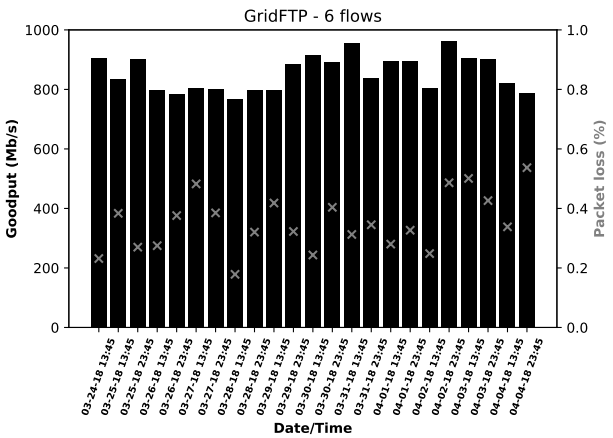
c) Goodput & Packet Loss for FASP with 4 Flows



d) Goodput & Packet Loss for GridFTP with 4 Flows

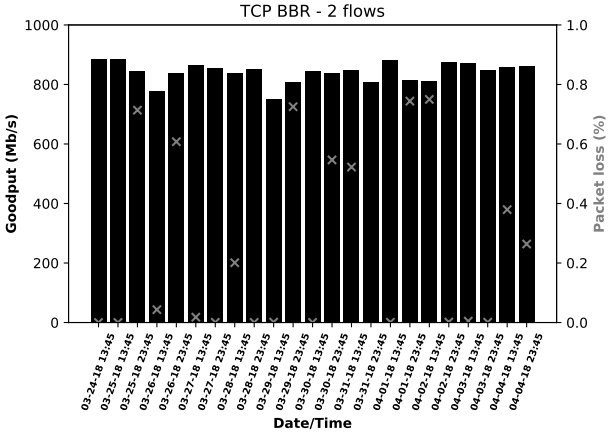


e) Goodput & Packet Loss for FASP with 6 Flows

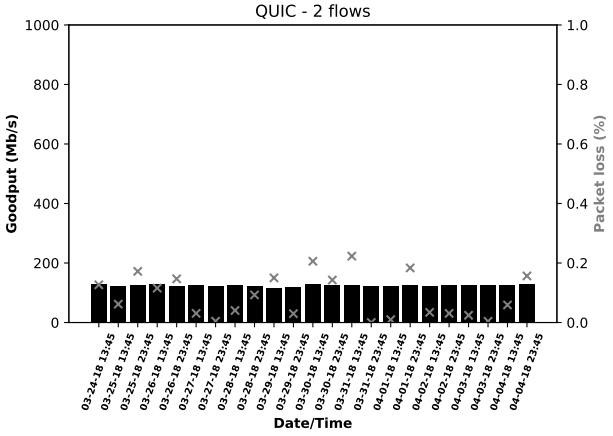


f) Goodput & Packet Loss for GridFTP with 6 Flows

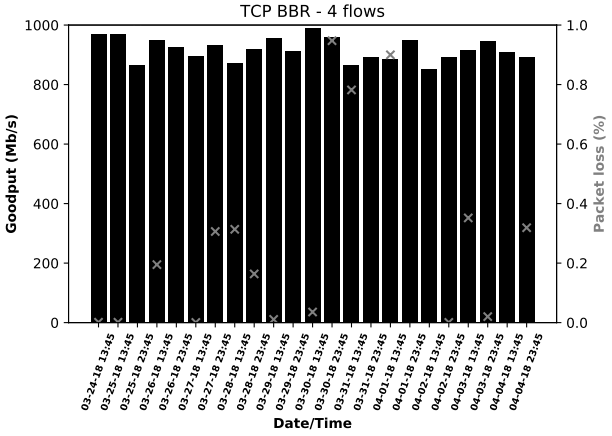
Figure 4.47: Goodput & Packet Loss for Protocols with Multiple Flows (1) - Local Setup



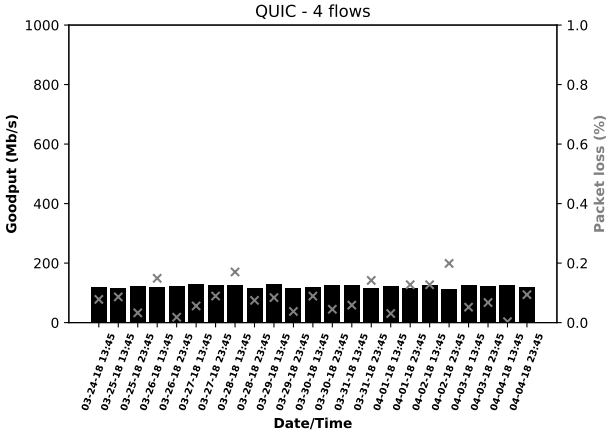
a) Goodput & Packet Loss for TCP BBR with 2 Flows



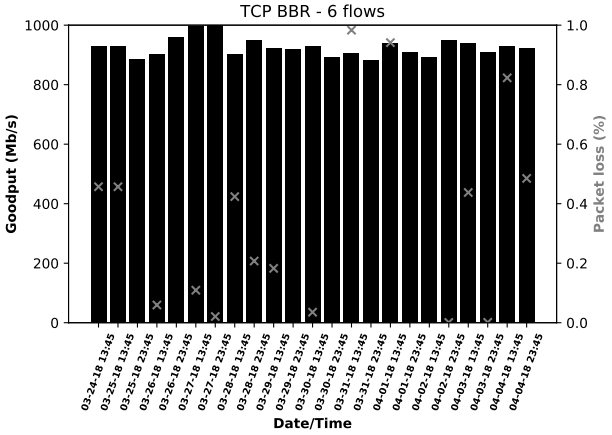
b) Goodput & Packet Loss for QUIC with 2 Flows



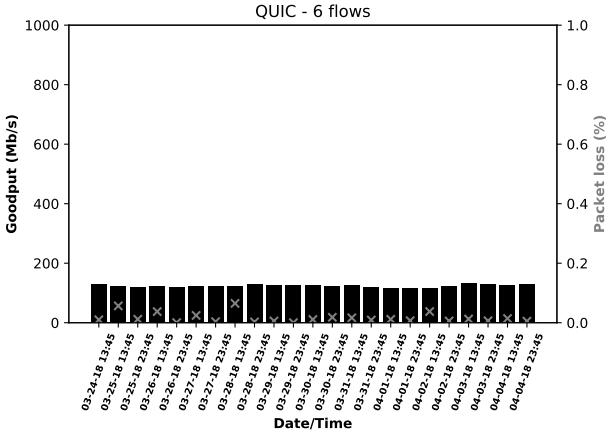
c) Goodput & Packet Loss for TCPBBR with 4 Flows



d) Goodput & Packet Loss for QUIC with 4 Flows



e) Goodput & Packet Loss for TCPBBR with 6 Flows



f) Goodput & Packet Loss for QUIC with 6 Flows

Figure 4.48: Goodput & Packet Loss for Protocols with Multiple Flows (2) - Local Setup

4.3.2 Waterloo Testbed Results for Multiple File Transfers

Figure 4.49 shows aggregate goodput of each protocol using multiple transfer flows. Compared to the single flow experiments, using multiple flows improves the performance of all the protocols. Similar to the local results, the goodput of GridFTP was the highest, especially for 6 flows. This is possibly because the aggregate congestion window growth rate has increased with the use of more flows, which also increased the sending rate, thus the aggregate goodput has increased. TCP BBR also had a high goodput in this setup, which shows it is because it also experiences low packet loss as shown in Figure 4.50 and uses a high sending rate.

The goodput for FASP does not seem to improve as the number of flows increases in this setup. This could be caused by the high packet loss, which affects its goodput. The same goes for the other protocols as they achieve reduced goodput rates compared to the local setups, which is also related to the firewall and the inefficient destination machine.

The goodput results of QUIC improve with the use of multiple flows. This also shows that QUIC aggregate sending rate increases as the number of flows increases. This is a similar behaviour to the local setup for experiment 1.

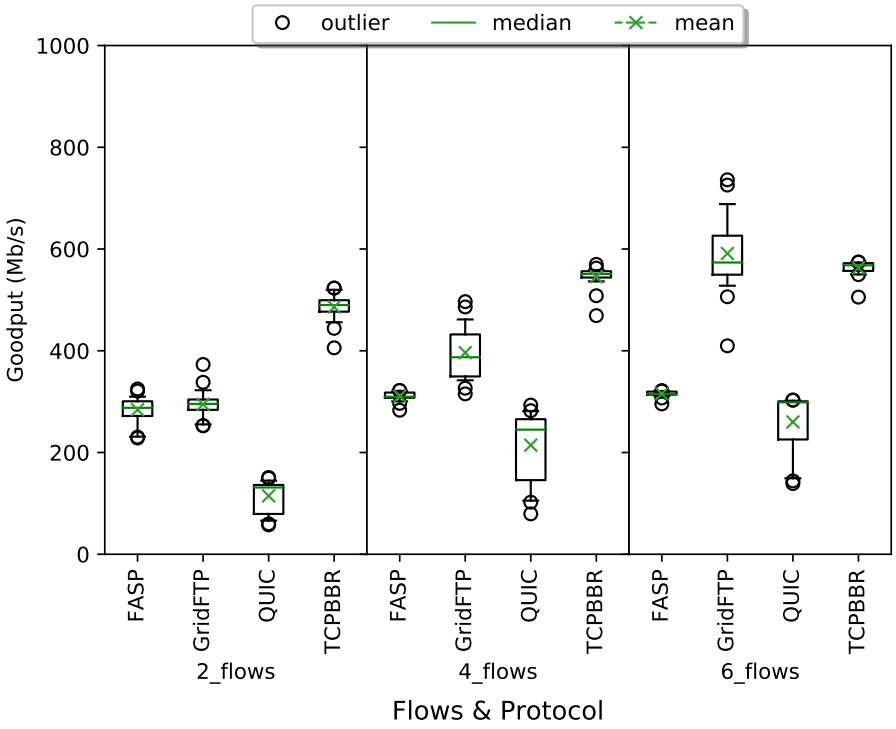


Figure 4.49: Aggregate Goodput for Each Protocol with Multiple Flows - Waterloo Setup

Figure 4.50 shows the aggregate packet loss for each protocol using multiple transfer flows. As explained earlier in Section 4.1.2, the high packet loss experienced by FASP is caused by the firewall at the destination, but using multiple flows increases packet loss. This is because each flow is trying to probe for more bandwidth, but each flow can't obtain the sufficient bandwidth, thus the aggregate packet loss increases. High packet

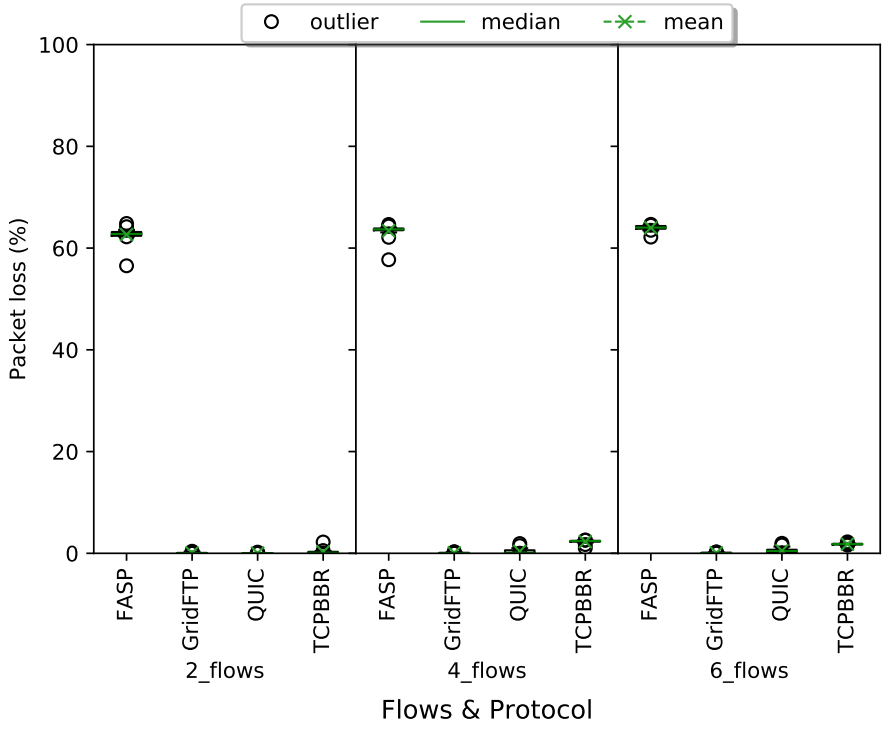
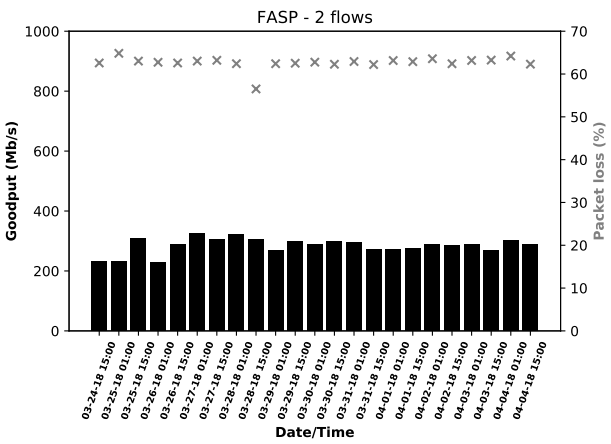


Figure 4.50: Aggregate Packet Loss for Each Protocol with Multiple Flows - Waterloo Setup

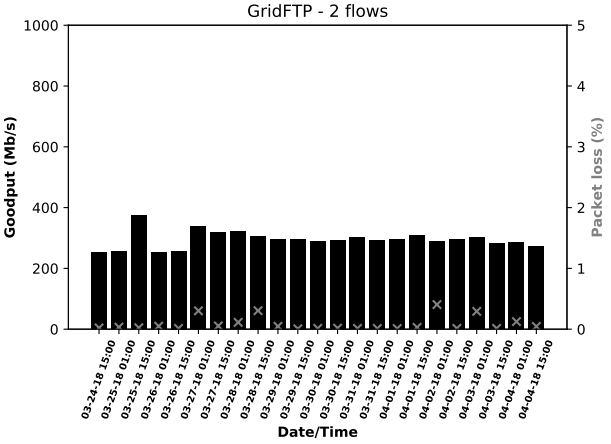
loss was expected since multiple flows were sharing the same outgoing link. The performance of FASP could have been optimized to prevent more packet loss by setting a lower target transfer rate for each FASP flow used. This was not possible as the FASP software license had expired before further experiments could be performed with a different target rate. The packet loss of GridFTP is lower compared to packet loss of the single flow experiments, which means that GridFTP is more reliable when multiple flows are used. Furthermore, TCP BBR and QUIC have some packet loss and it increases as the number of flows increases.

Figure 4.51 and 4.52 show the goodput and packet loss results for every run of each protocol. In terms of consistency, the goodput results for FASP and TCP BRR were similar for each run, which shows that they both can achieve a stable performance. The behaviour of TCP BBR is still similar to that using single flow. In addition, GridFTP and QUIC achieved varying goodput results between the replications with QUIC having a higher variation. This could be caused by the congestion control mechanism (CUBIC), which both use in responding to packet loss, thus the goodput varies from a replication to another. GridFTP is still able to achieve high goodput, especially for 6 flows. In terms of packet loss, FASP had the highest packet loss followed by TCP BBR, and they both seem to be aggressive in using the available bandwidth. The packet loss of GridFTP is low compared to single flow and it replicates the same behaviour for the various runs. QUIC starts to have more packet loss as the number of flows increase, which potentially implies that the flows are not fairly sharing bandwidth.

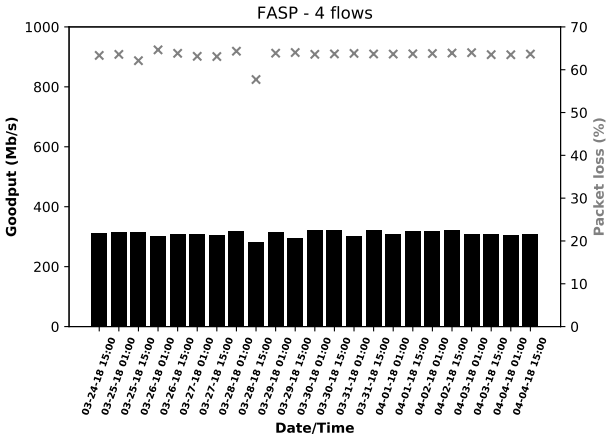
Figure 4.53 shows the sending rate over time for QUIC (2 flows) for a high goodput replication and a



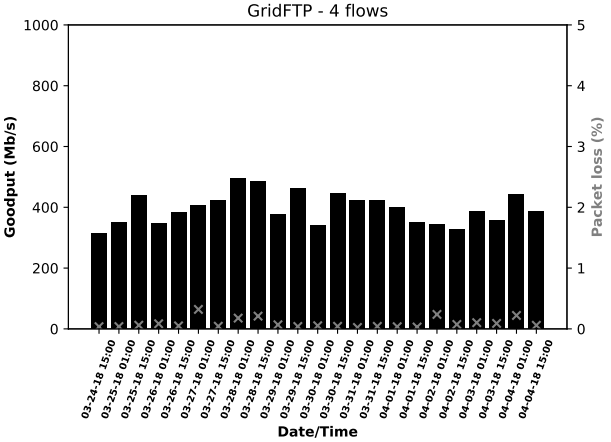
a) Goodput & Packet Loss for FASP with 2 Flows



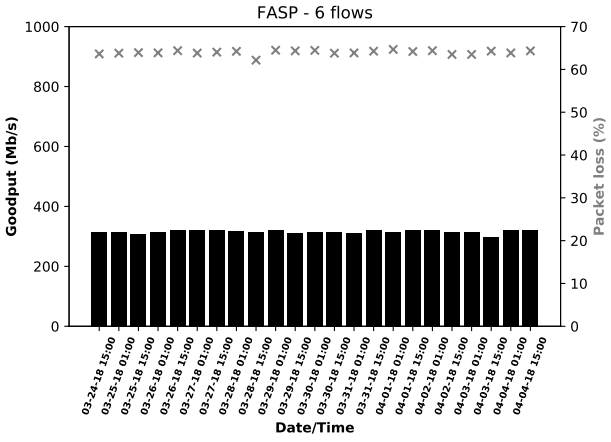
b) Goodput & Packet Loss for GridFTP with 2 Flows



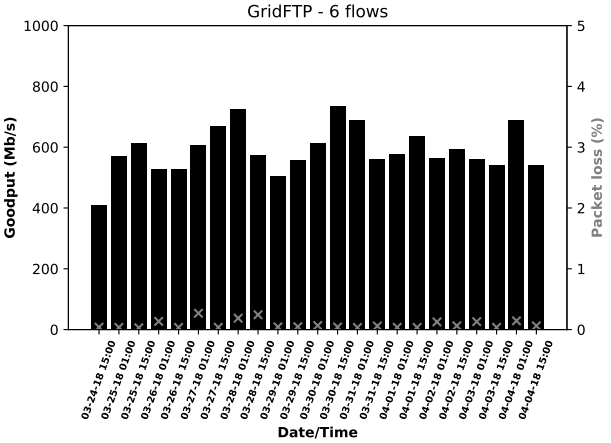
c) Goodput & Packet Loss for FASP with 4 Flows



d) Goodput & Packet Loss for GridFTP with 4 Flows

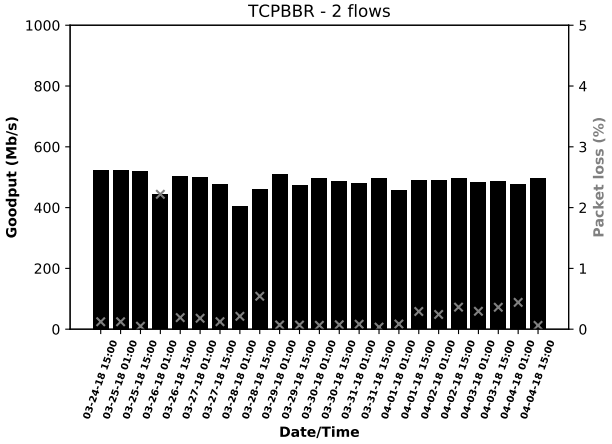


e) Goodput & Packet Loss for FASP with 6 Flows

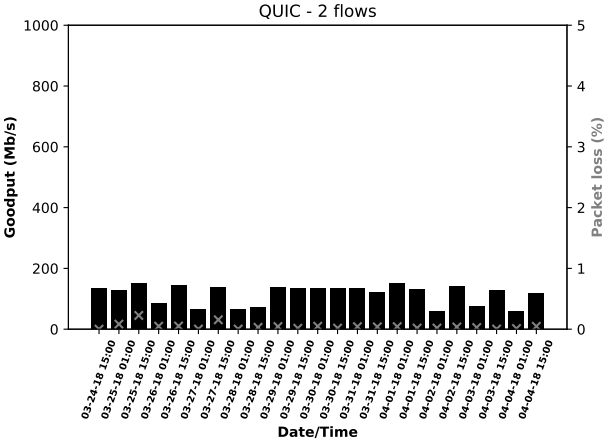


f) Goodput & Packet Loss for GridFTP with 6 Flows

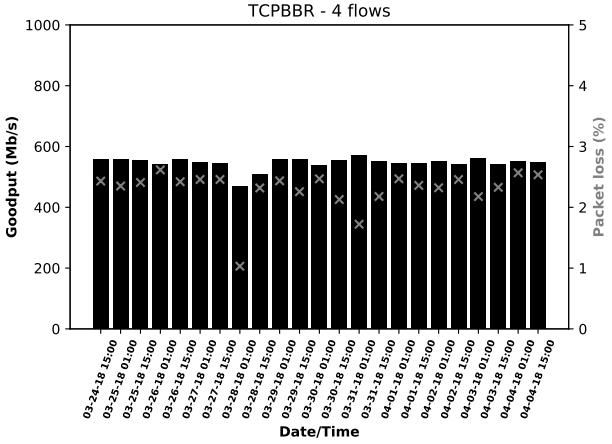
Figure 4.51: Goodput & Packet Loss for Protocols with Multiple Flows (1) - Waterloo Setup



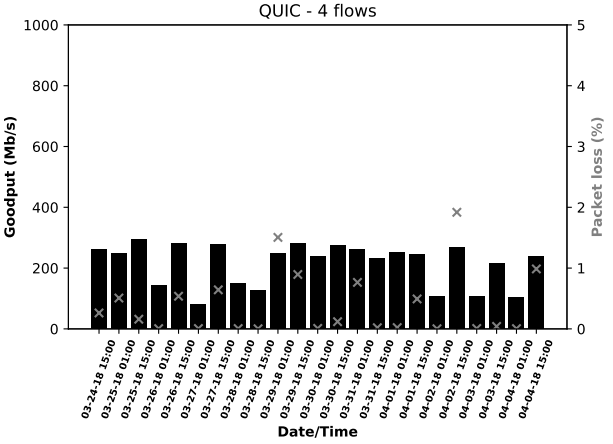
a) Goodput & Packet Loss for TCP BBR with 2 Flows



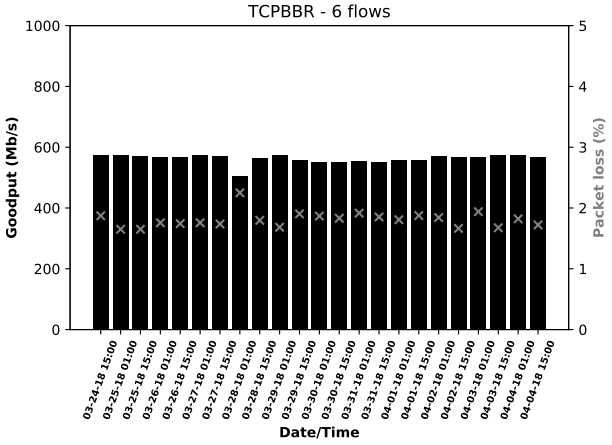
b) Goodput & Packet Loss for QUIC with 2 Flows



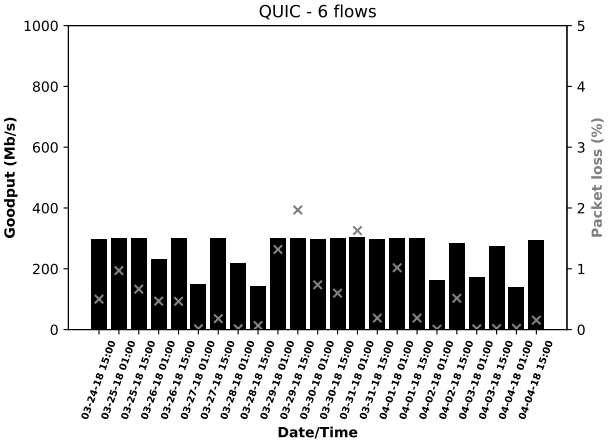
c) Goodput & Packet Loss for TCPBBR with 4 Flows



d) Goodput & Packet Loss for QUIC with 4 Flows



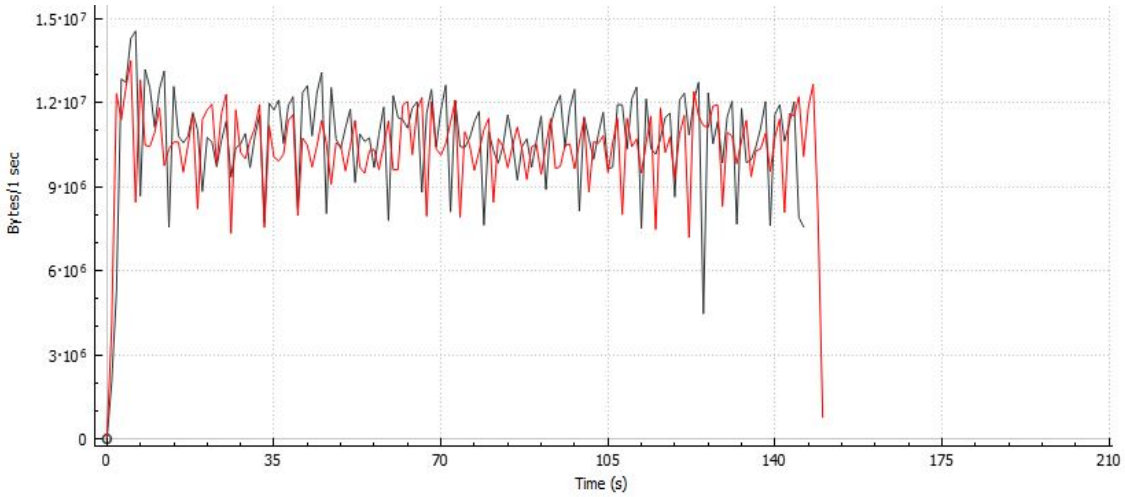
e) Goodput & Packet Loss for TCPBBR with 6 Flows



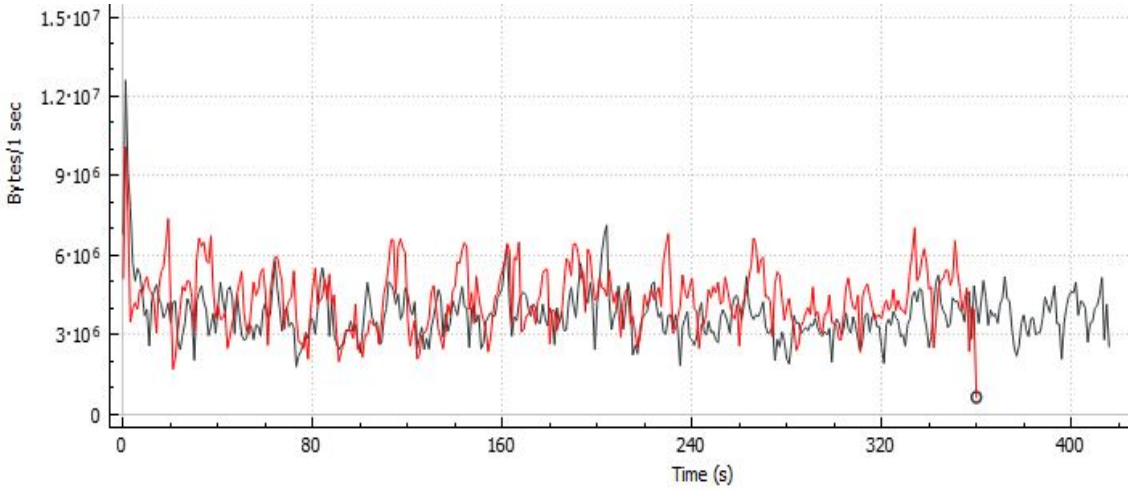
f) Goodput & Packet Loss for QUIC with 6 Flows

Figure 4.52: Goodput & Packet Loss for Protocols with Multiple Flows (2)- Waterloo Setup

low goodput replication. This shows that for replication 16 QUIC is able to achieve a high sending rate, thus takes a shorter time to complete the transfer. However, QUIC has a substantially low sending rate for replication 18 and takes longer to complete the transfer. This affects the goodput of each flow used by QUIC. Each flow of QUIC also has a different pattern and one flow completes the transfer before the other. This is because one flow achieves a higher sending rate than the other. Both flows also experience different drops in their sending rate, which is either a response to packet loss or the number of chunks being sent. Whenever QUIC prepares a new chunk, it reduces its sending rate for approximately 50 ms and then sends that chunk. This behaviour of QUIC shows that it is unstable when using multiple flows on high RTT links.



a) Sending Rate over Time for QUIC -2 flows - Replication 16



b) Sending Rate over Time for QUIC -2 flows - Replication 18

Figure 4.53: Sending Rate over Time for QUIC - 2 flows - Waterloo Setup

4.3.3 Eastcloud Testbed Results for Multiple File Transfers

Figure 4.54 shows the aggregate goodput of each protocol using multiple flows. Using multiple flows still improves the performance of the protocols in this setup compared to the Waterloo setup. The goodput of GridFTP and FASP is similar using 2 flows, but as the number of flows increases, GridFTP has a higher goodput than FASP. Additionally, GridFTP has higher goodput than in the Waterloo setup. It is possible that this setup has more available bandwidth and a more efficient destination machine.

The high goodput achieved by FASP in this setup implies that the packet loss experienced by FASP in this setup is low as seen in Figure 4.55. FASP is also able to utilize more bandwidth in setup compared to that in the Waterloo setup. The goodput does not increase when the 6 flows are used and it is somewhat similar to the goodput of the 4 flows. This is caused by the target sending rate set to 600 Mbps for each flow, and not all the flows would be able to fully utilize the 600 Mbps since the link is limited to 1 Gbps and each flow would have a similar sending rate. Therefore, there is no noticeable difference observed between the goodput of the 4 flows and the 6 flows.

The goodput of QUIC has increased compared to the goodput it achieved in the local setup. This is caused by the use of multiple flows, which tends to increase its sending rate, thus the aggregate goodput. However, this goodput is lower than in the Waterloo setup, which might be related to the different hops and the increased RTT compared to the Waterloo setup.

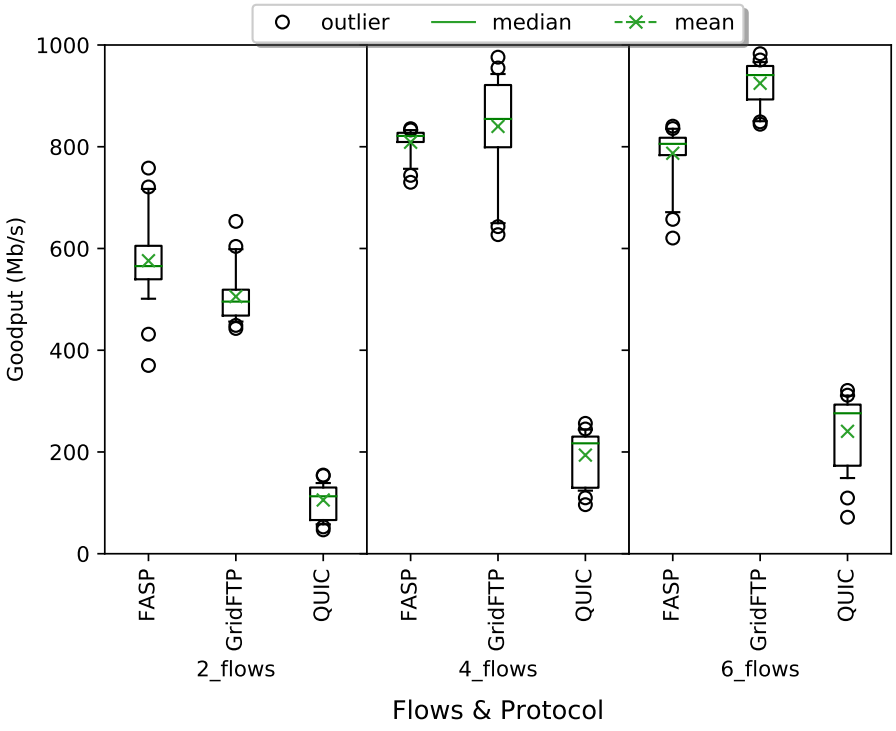


Figure 4.54: Aggregate Goodput for Each Protocol with Multiple Flows - Eastcloud Setup

Figure 4.55 shows the aggregate packet loss of each protocol using multiple transfer flows. The presence

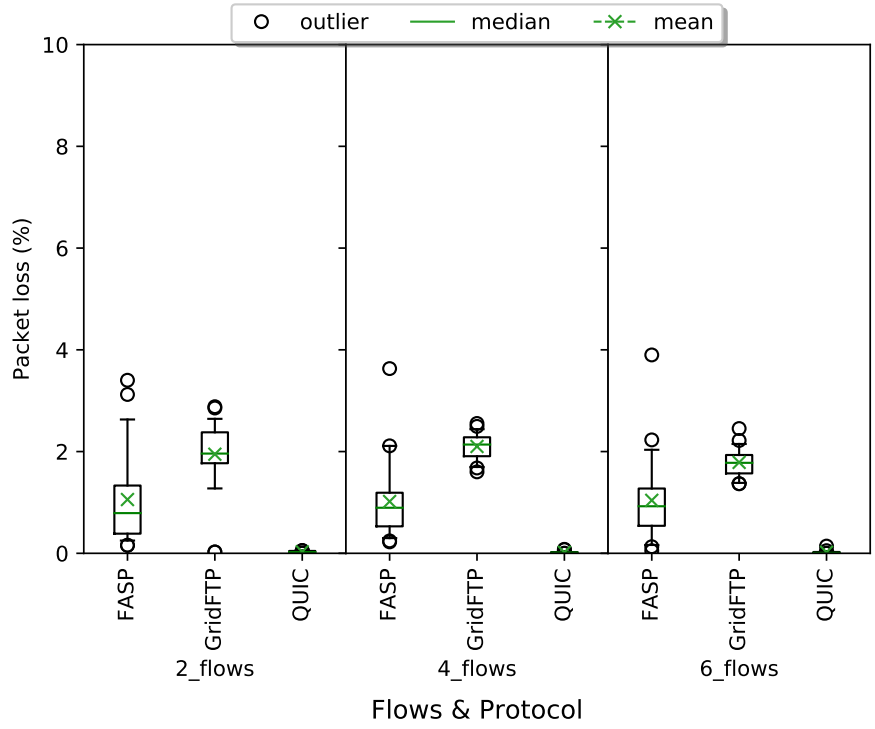


Figure 4.55: Aggregate Packet Loss for Each Protocol with Multiple Flows - Eastcloud Setup

of multiple flows causes more packet loss to occur compared to the local setup, because the sending rate increases and congestion occurs on the link, which causes the consumption of all the available bandwidth. With multiple flows, GridFTP had the highest packet loss followed by FASP. This is potentially caused by the congestion induced by the multiple flows. Each flow is tries to utilize more bandwidth, thus causes more congestion on the link. Compared to a single flow, the packet loss of FASP went above 1% for multiple flows while in the single flow (experiment 1) it was below 1%. Compared to the local setup, the packet loss of FASP has substantially decreased. FASP also achieved a lower packet loss than GridFTP as the number of flows increase for most of the replications, which shows that it shares bandwidth more fairly between its flows.

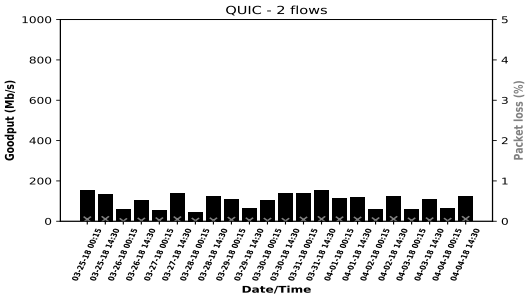
GridFTP achieved a higher packet loss rate in this setup compared to the local and Waterloo setup. For TCP-based flows such as GridFTP, a single flow had lower packet loss than multiple flows. As explained earlier in Section 4.3.1, packet loss can affect each flow, thus multiple flows can decrease their sending rate at the same time or at different times. The increasing number of packets increases congestion and more packets can get dropped at routers or at the destination host. Consequently, multiple flows that have a high sending rate will likely lose more packets as they can cause more congestion if they don't fairly share the bandwidth. QUIC does not seem to be affected by multiple flows, therefore high packet loss is not observed. This also shows that QUIC shares bandwidth fairly between all its flows.

Figures 4.56 and 4.57 display the goodput and packet loss results each protocol for the various replica-

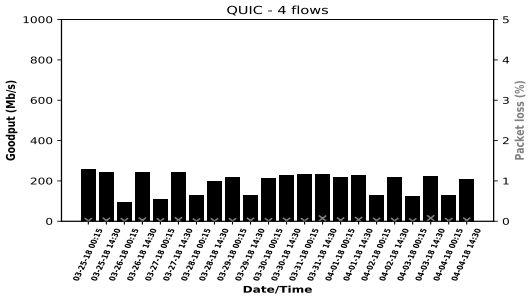
tions. In terms of consistency, the goodput results for FASP were similar for each run. This shows that it is able to produce similar results for every replication and it is more stable with multiple flows. Even though FASP experiences high packet loss in some of the runs, it still maintains the same goodput as the other runs with lower packet loss.

The results of GridFTP and QUIC varied from one run to another. Since both use CUBIC, they tend to respond to any packet loss that occurs as a sign of congestion and both reduce their sending rate in order to prevent any further congestion. GridFTP is able to achieve high goodput for most of the runs, especially using 6 flows, which shows that it is able to utilize more bandwidth in total. This is a better performance than observed using a single flow.

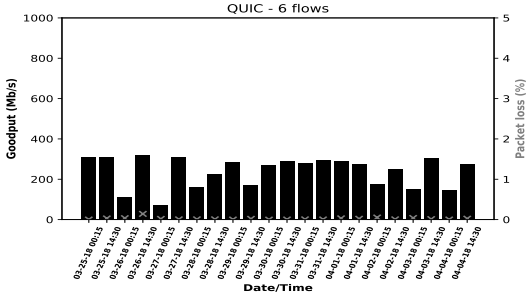
QUIC also has a better performance using multiple flows compared to using a single flow. The aggregate goodput also increases as the number of flows increases for QUIC overall, since there is more available bandwidth to utilize by QUIC. The goodput results for each replication varies for QUIC depending on the amount of traffic on the network and available bandwidth that varies for each replication. The packet loss achieved by QUIC is low compared to the previous setups, which shows this setup has a more stable link. All the protocols showed different packet loss values for every run, which shows that different amounts of congestion are observed in every replication.



c) Goodput & Packet Loss for QUIC with 2 Flows

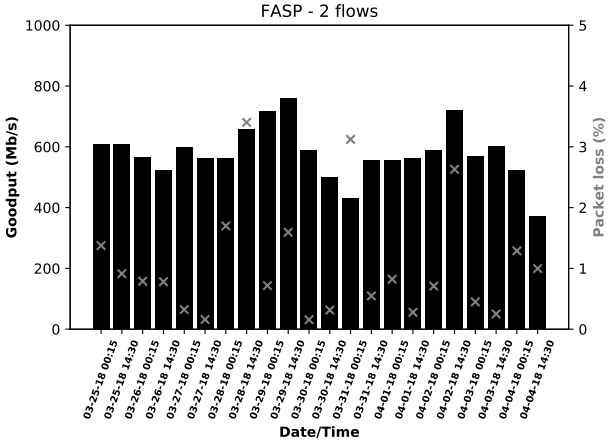


f) Goodput & Packet Loss for QUIC with 4 Flows

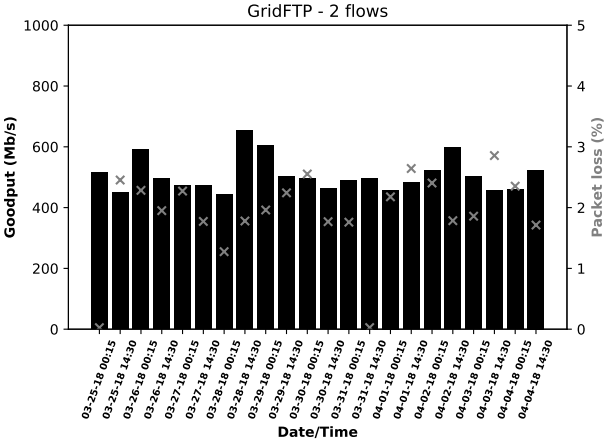


i) Goodput & Packet Loss for QUIC with 6 Flows

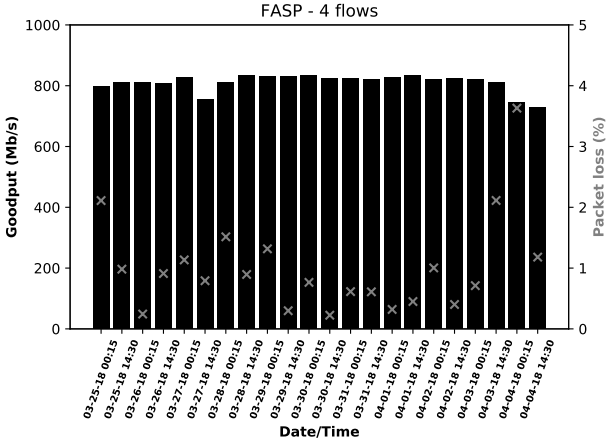
Figure 4.56: Goodput & Packet Loss for Protocols with Multiple Flows (1) - Eastcloud Setup



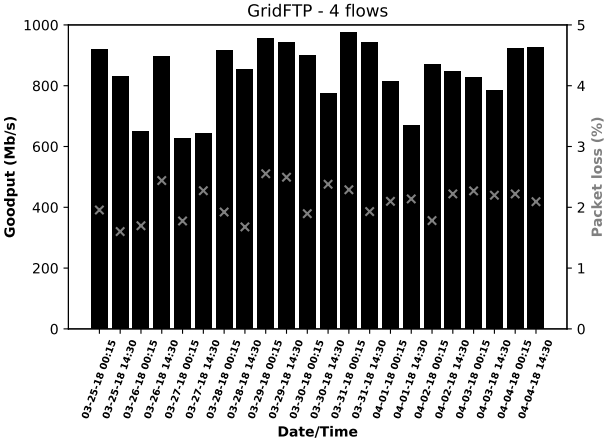
a) Goodput & Packet Loss for FASP with 2 Flows



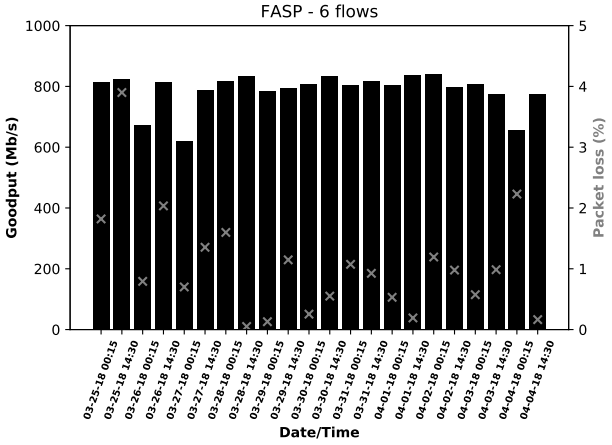
b) Goodput & Packet Loss for GridFTP with 2 Flows



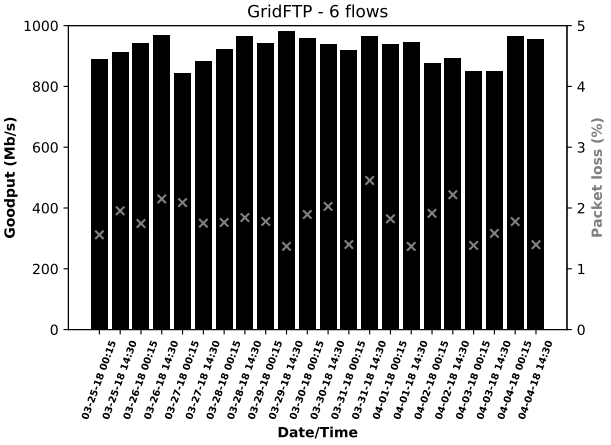
d) Goodput & Packet Loss for FASP with 4 Flows



e) Goodput & Packet Loss for GridFTP with 4 Flows



g) Goodput & Packet Loss for FASP with 6 Flows



h) Goodput & Packet Loss for GridFTP with 6 Flows

Figure 4.57: Goodput & Packet Loss for Protocols with Multiple Flows (2) - Eastcloud Setup

4.3.4 International Testbed Results for Multiple File Transfers

Figure 4.58 shows aggregate goodput of each protocol using multiple transfer flows. FASP and GridFTP had the highest goodput among the other protocols and better goodput compared to the single flow experiments. This shows that FASP and GridFTP are able to utilize more bandwidth. FASP increases its sending rate on long RTT links and reaches the target sending rate that was preset in its configuration for 4 flows and 6 flows. Each flow uses approximately a sending rate of 160 Mbps, which nearly utilizes the full link bandwidth when using 6 flows.

As the number of flows increases, GridFTP is able to achieve higher aggregate goodput than FASP, which means that all the flows of GridFTP are using a high sending rate. The goodput for TCP BBR and QUIC has decreased compared to the Waterloo setup. This is possibly caused by the increased amount of cross traffic and the delay on this link.

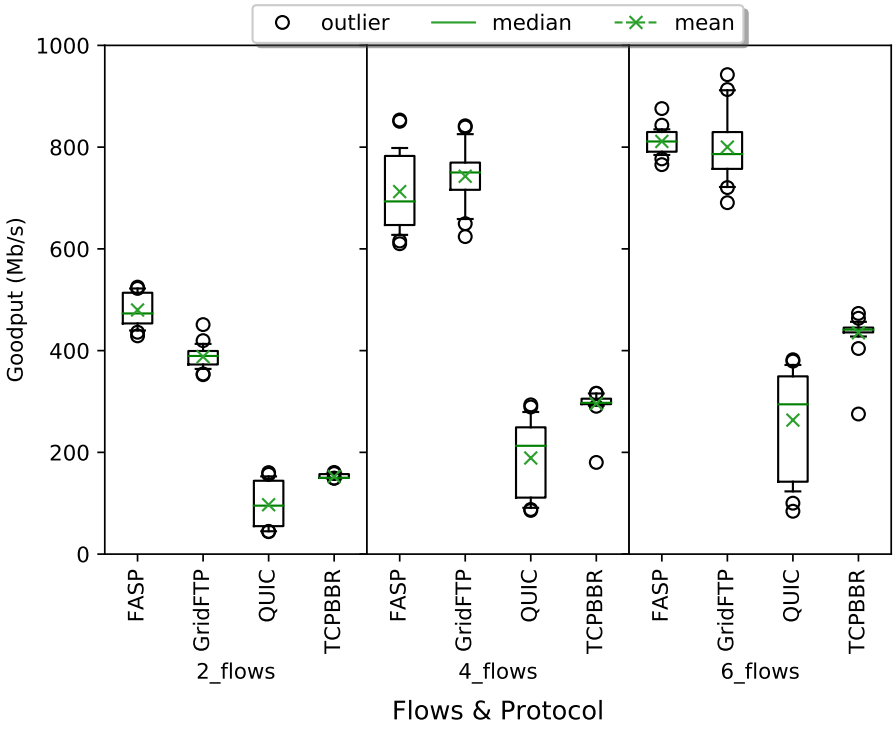


Figure 4.58: Aggregate Goodput for Each Protocol with Multiple Flows - Auckland Setup

Figure 4.59 shows the aggregate packet loss of each protocol using multiple transfer flows. All the protocols seem to achieve a packet loss below 1% for all types the flows. This is a similar behaviour to the single flow experiments for the same setup.

With multiple flows, FASP had low packet loss with two flows, but the aggregate packet loss increased as the number of flows increased. This is because as the number of flows increases, more flows are sending traffic and each flow could experience packet loss. Therefore any additional flow can contribute to increasing the aggregate packet loss, which affects all the protocols. With that being said, the packet loss experienced

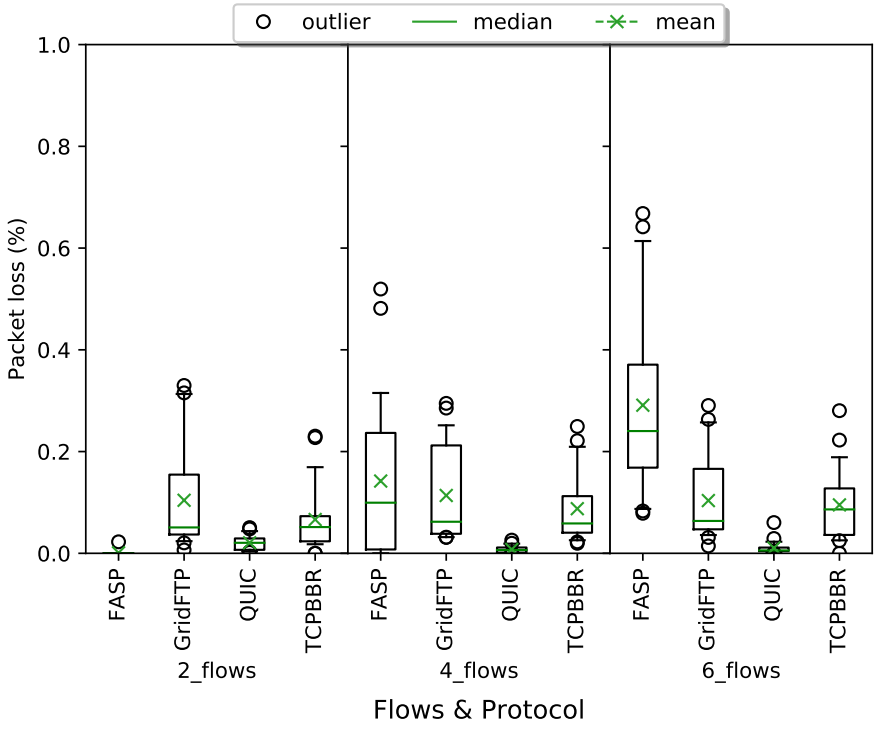
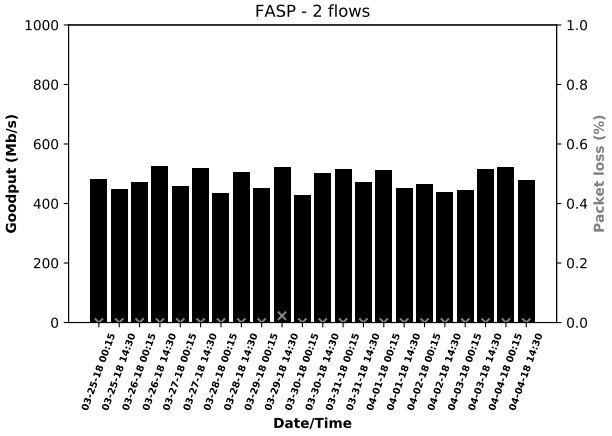


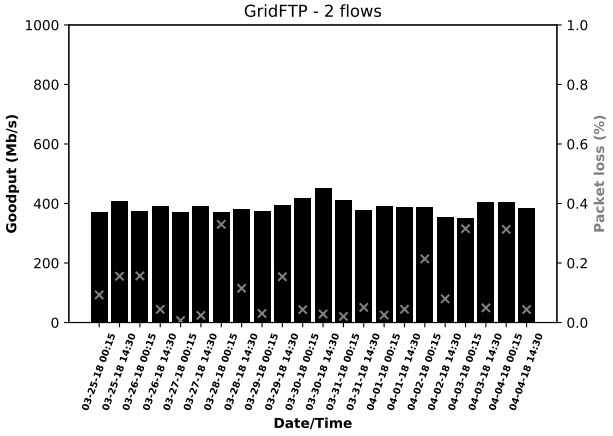
Figure 4.59: Aggregate Packet Loss for Each Protocol with Multiple Flows - Auckland Setup

by FASP with different types of flows is still low compared to the previous setups. Similar to previous setups, QUIC achieved a very low packet loss.

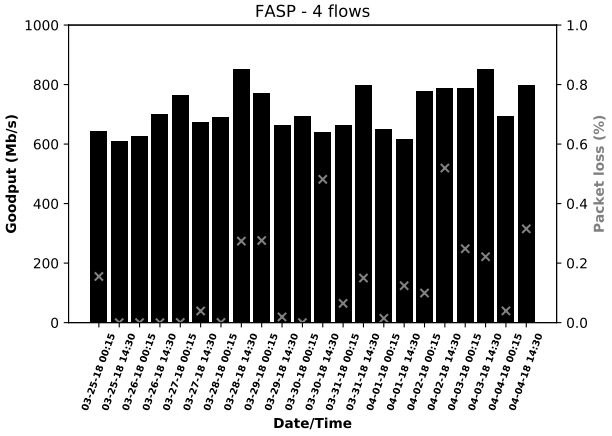
Figure 4.60 shows the goodput and packet loss results for each flow of each protocol with various runs. The goodput results for FASP, GridFTP, and TCP BRR are similar for most of the replications. These protocols are able to reproduce similar goodput results for every run, which shows that they are more stable with multiple flows in this setup compared to previous setups. In terms of packet loss, all the protocols had varying packet loss values for each replication, but had a packet loss below 0.5 %. This low packet loss and should not affect the performance of the protocols severely, especially GridFTP and QUIC. Also, as the number of flows increases, the packet loss increases and varies more between the replication for FASP. This is because as the number of flows increases, the aggregate packet loss would increase if each flow experiences packet loss. However, GridFTP with 4 flows showed higher packet loss than for 6 flows while TCP BBR showed similar packet loss for both 2 and 4 flows. This depends on the level of congestion induced by traffic, which varied for each replication. Moreover, the results of GridFTP and QUIC varied from one run to another. Again, this behaviour was observed in the Waterloo setup, but with a higher RTT in this setup. The behaviour of the protocols depends on the varying network conditions for every replications.



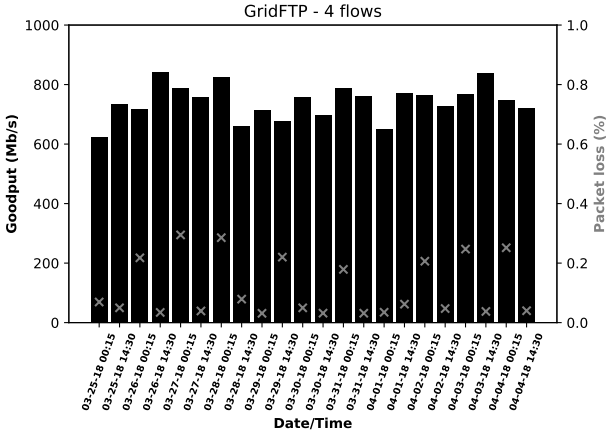
a) Goodput & Packet Loss for FASP with 2 Flows



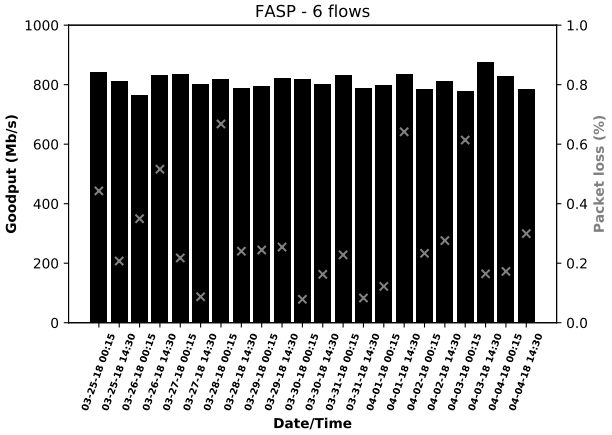
b) Goodput & Packet Loss for GridFTP with 2 Flows



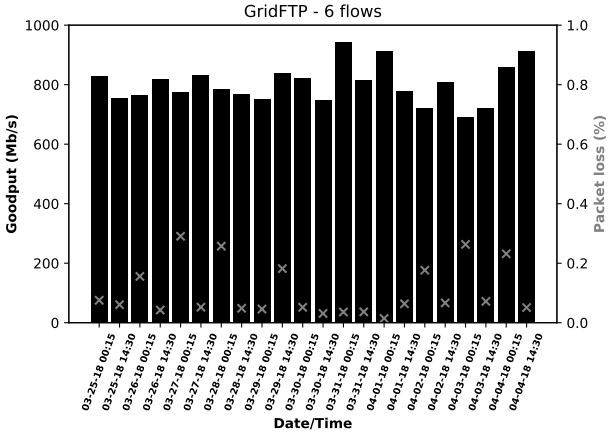
c) Goodput & Packet Loss for FASP with 4 Flows



d) Goodput & Packet Loss for GridFTP with 4 Flows

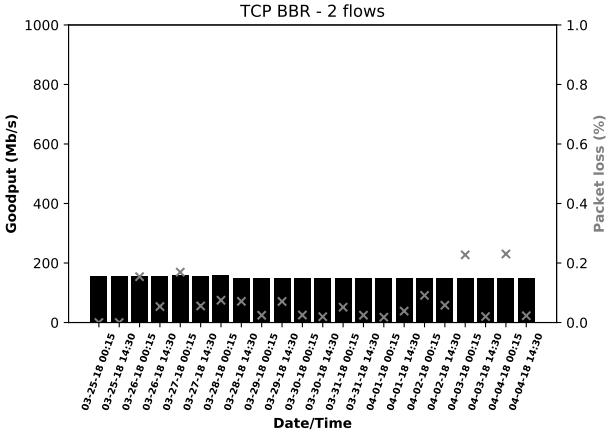


e) Goodput & Packet Loss for FASP with 6 Flows

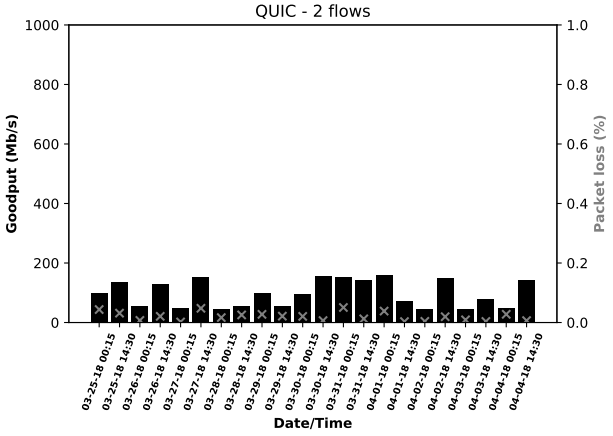


f) Goodput & Packet Loss for GridFTP with 6 Flows

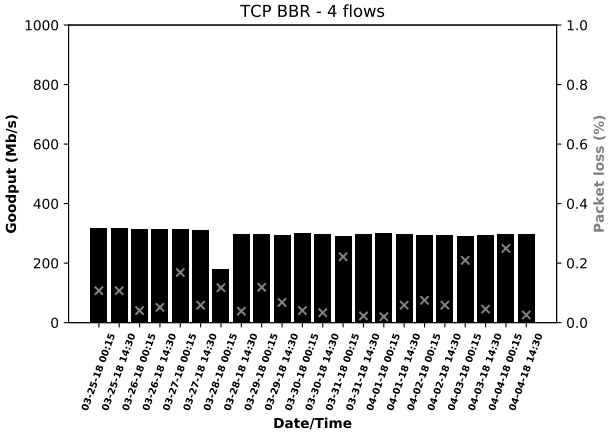
Figure 4.60: Goodput & Packet Loss for Protocols with Multiple Flows (1) - Auckland Setup



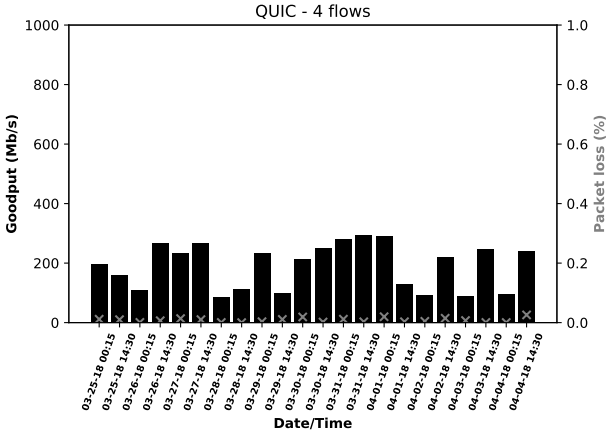
a) Goodput & Packet Loss for TCP BBR with 2 Flows



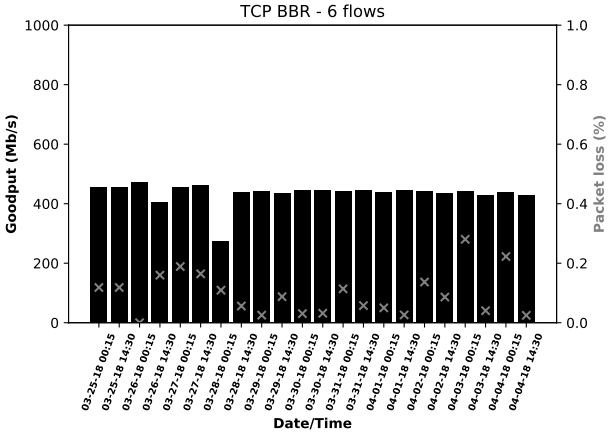
b) Goodput & Packet Loss for QUIC with 2 Flows



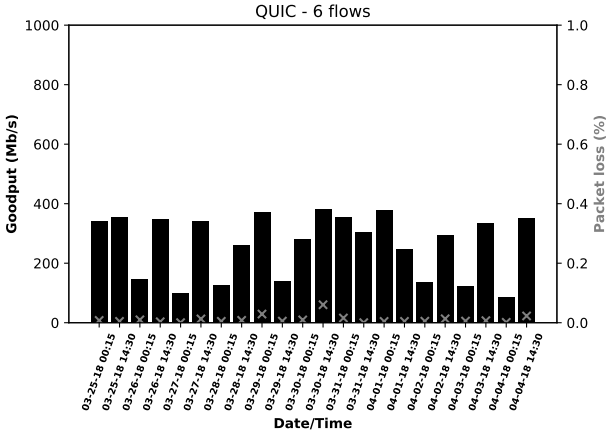
c) Goodput & Packet Loss for TCPBBR with 4 Flows



d) Goodput & Packet Loss for QUIC with 4 Flows



e) Goodput & Packet Loss for TCPBBR with 6 Flows



f) Goodput & Packet Loss for QUIC with 6 Flows

Figure 4.61: Goodput & Packet Loss for Protocols with Multiple Flows (2) - Auckland Setup

4.4 Results for Experiments with Competing Data Transfer Traffic

4.4.1 Local Testbed Results for Transfers with Competing Traffic

Figure 4.62 shows the aggregate goodput for each protocol with competing traffic from other file transfers. The goodput results of FASP and GridFTP are similar and they nearly utilize the same bandwidth when competing with other protocols. The highest goodput achieved by FASP and GridFTP was with QUIC and TCP BBR traffic, but was similar when they were competing against each other. This shows that both FASP and GridFTP aggressively compete for more bandwidth, even if there is competing traffic. When they compete against each other, they achieve similar goodput, because their design aims at utilizing the highest sending rate possible. The aggregate goodput of FASP and GridFTP decreased compared to the observed aggregate goodput in experiment 1 for the local setup.

The goodput of TCP BBR decreases more when FASP and GridFTP are competing with it. This shows FASP and GridFTP are utilizing more bandwidth than TCP BBR and thus they are not sharing the bandwidth fairly with TCP BBR. This is a low goodput compared to the goodput observed in experiment 1 for the local setup. TCP BBR was able to reach a goodput above 800 Mbps for single flow while in this experiment it is only able to reach a goodput below 600 Mbps.

For QUIC, there is no significant effect from competing traffic observed since it does not utilize high bandwidth, therefore the goodput is low in all cases. However, this goodput is lower than the goodput observed in experiment 1 for the local setup. With QUIC's low bandwidth utilization, all the other protocols can utilize a large share of bandwidth and increase their sending rate as per the available capacity.

In terms of packet loss, Figure 4.63 shows that FASP has the highest packet loss in this setup. This is a similar behaviour as in the previous experiments and for the same reason mentioned earlier. The rest of the protocols achieved a packet loss that did not exceed 0.7 percent, which is also similar to the packet loss observed in the previous local experiments. The packet loss of the protocols does not get affected by competing traffic in this setup, except for FASP which is for the same reason explained in the previous local experiments.

Figures 4.64 and 4.65 show the goodput and packet loss results of each protocol for every individual run. As can be seen, the performance of FASP and GridFTP over several replications varies when they compete against each other. They both try to utilize high bandwidth rates, therefore their goodput is lower than the goodput they have when competing with other protocols. The goodput of FASP with QUIC and TCP BBR traffic was higher and showed more consistent results for each run. This also implies that FASP is aggressive and does not fairly share bandwidth with the other protocols. In addition, QUIC also had similar results for each run, but with low goodput rate. The goodput results of GridFTP with QUIC and TCP BBR is high for most of the runs. This shows that GridFTP is able to utilize more bandwidth than the competing protocols. The performance of TCP BBR seems to be similar with FASP and GridFTP traffic, but it is not able to reach

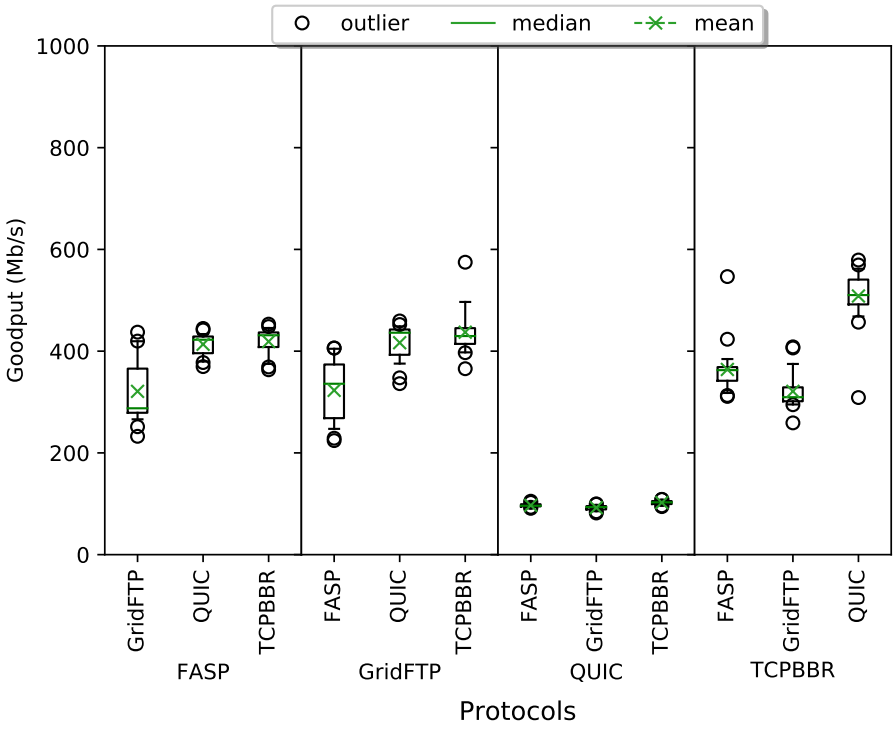


Figure 4.62: Aggregate Goodput for Each Protocol with Competing Traffic - Local Setup

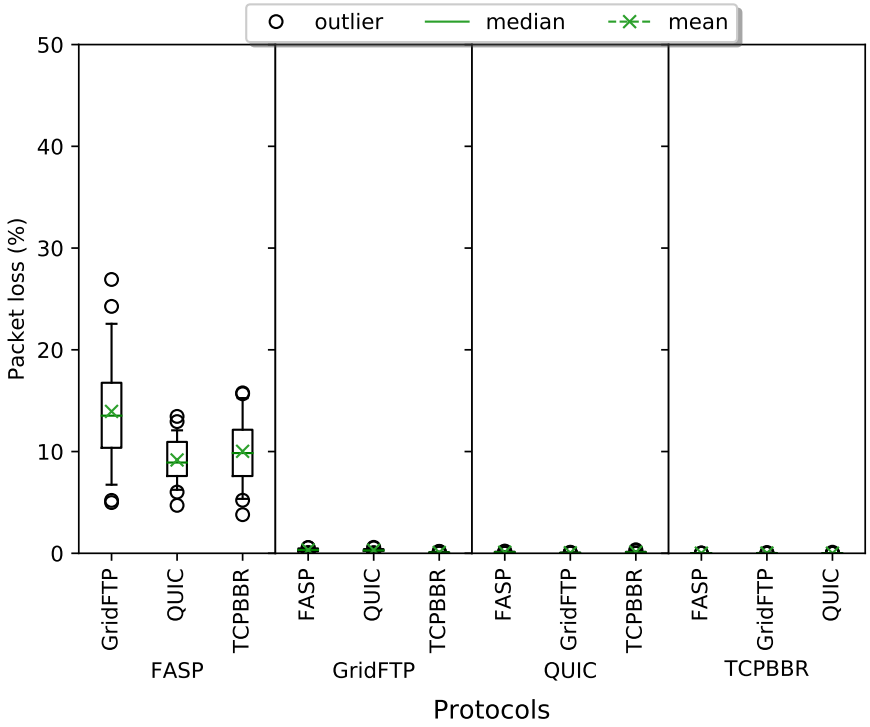
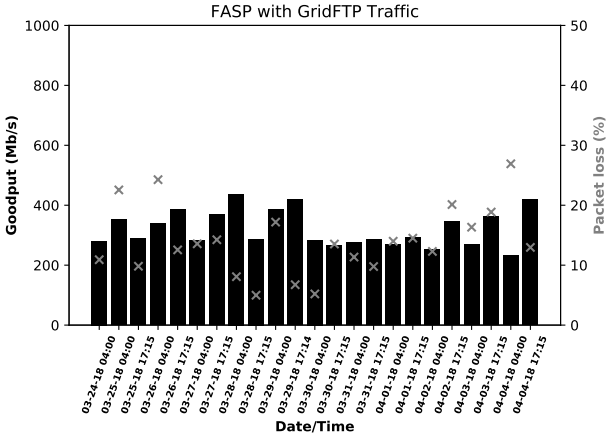
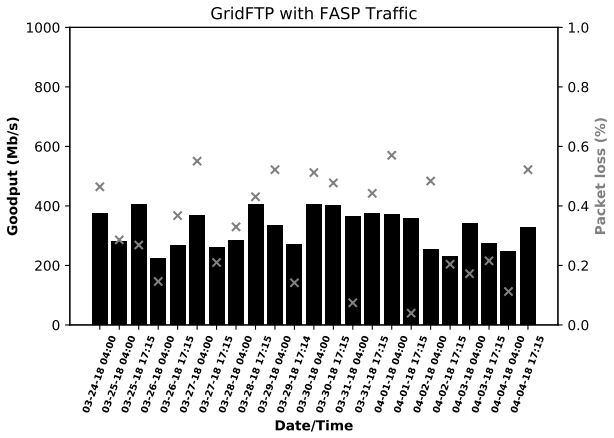


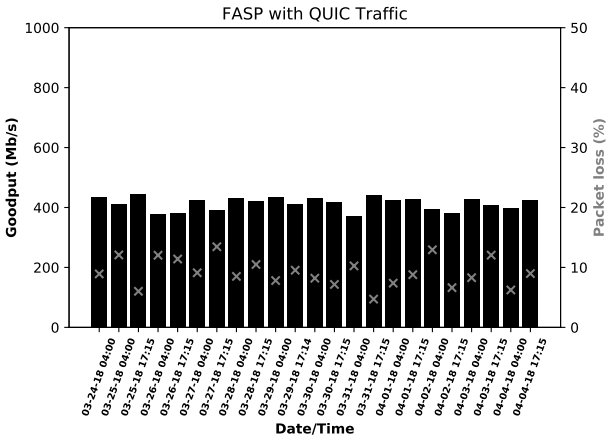
Figure 4.63: Aggregate Packet Loss for Each Protocol with Competing Traffic - Local Setup



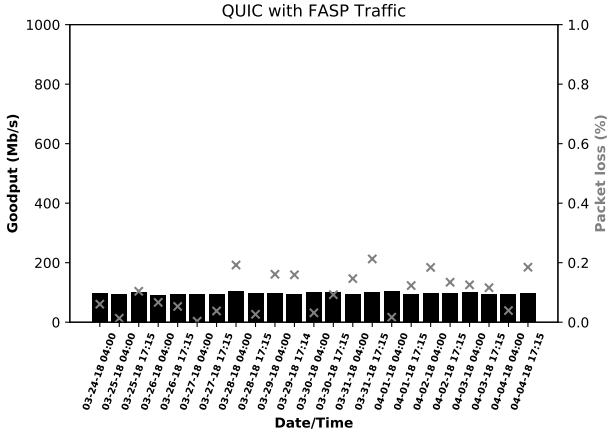
a) Goodput & Packet Loss for FASP with GridFTP Traffic



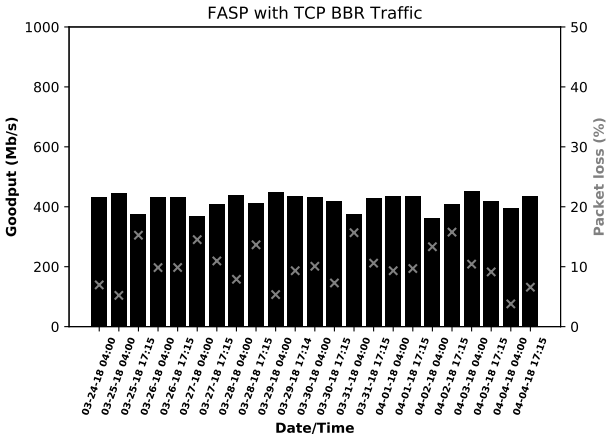
b) Goodput & Packet Loss for GridFTP with FASP Traffic



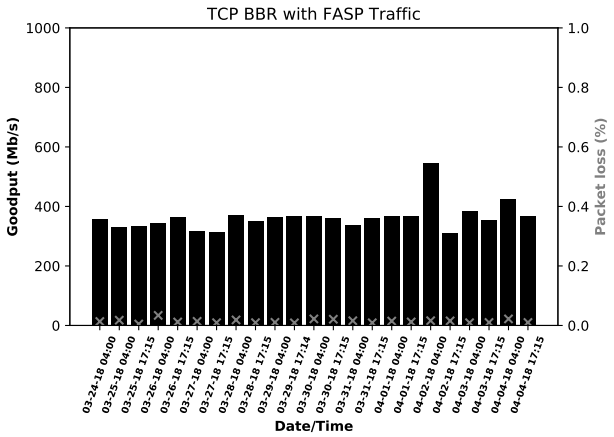
c) Goodput & Packet Loss for FASP with QUIC Traffic



d) Goodput & Packet Loss for QUIC with FASP Traffic

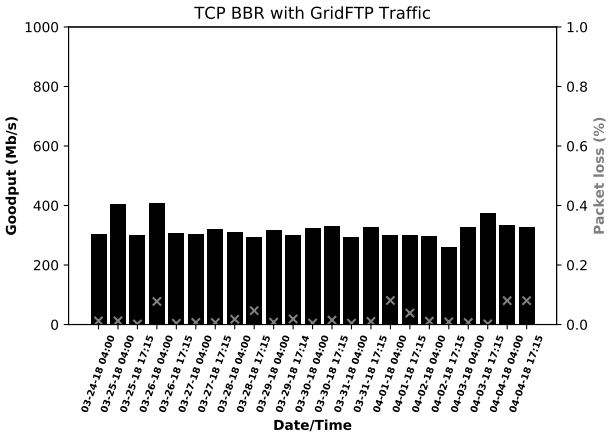


e) Goodput & Packet Loss for FASP with TCP BBR Traffic

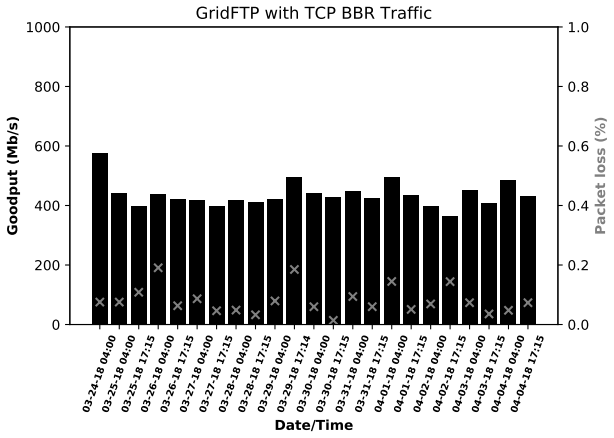


f) Goodput & Packet Loss for TCP BBR with FASP Traffic

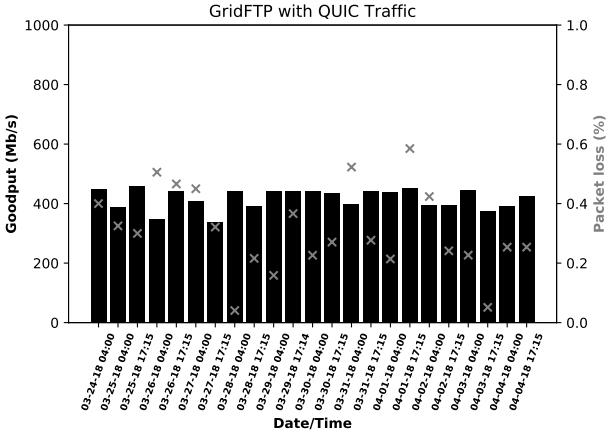
Figure 4.64: Goodput & Packet Loss for Protocols with Competing Traffic (1) - Local Setup



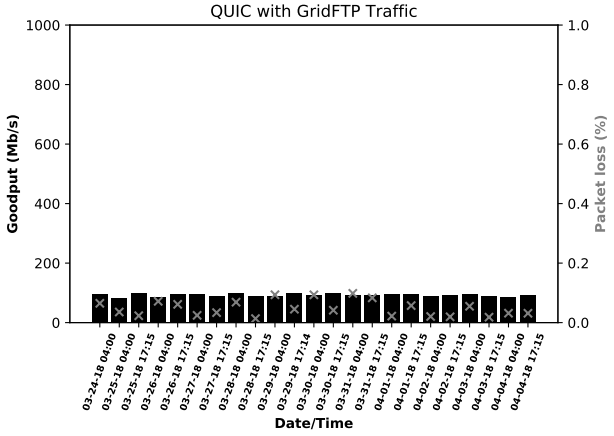
a) Goodput & Packet Loss for TCP BBR with GridFTP Traffic



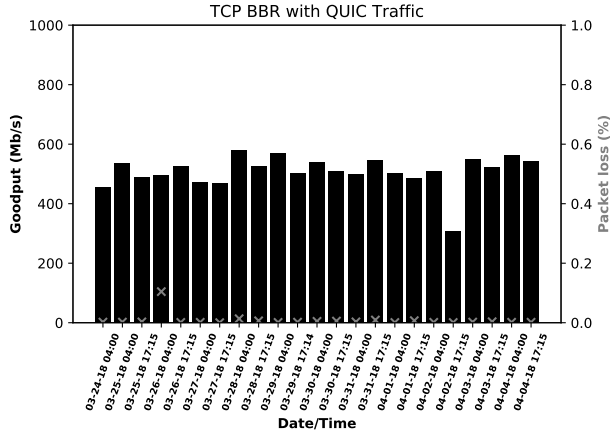
b) Goodput & Packet Loss for GridFTP with TCP BBR Traffic



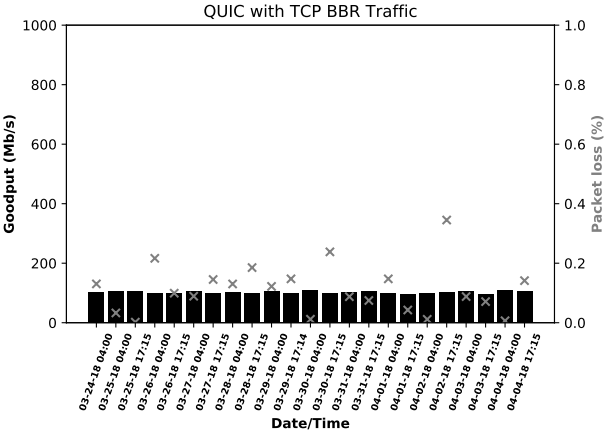
c) Goodput & Packet Loss for GridFTP with QUIC Traffic



d) Goodput & Packet Loss for QUIC with GridFTP Traffic



e) Goodput & Packet Loss for TCP BBR with QUIC Traffic



f) Goodput & Packet Loss for QUIC with TCP BBR Traffic

Figure 4.65: Goodput & Packet Loss for Protocols with Competing Traffic (2) - Local Setup

a high goodput rates as the previous set of experiments. This is because both FASP and GridFTP don't fairly share bandwidth with TCP BBR. Overall, FASP and GridFTP compete aggressively for bandwidth and with other protocols. In terms of packet loss, FASP and GridFTP have varying packet loss, which means that they experience different levels of congestion for each run. The congestion is not necessarily induced by the unknown traffic on the network, but it could be induced by the competing protocols when they are run in parallel.

In some instances like in Figure 4.66, FASP can experience high packet loss, and the sending rate decreases, but it still tries to reach a high sending rate again. This behaviour shows unpredictability and inconsistency in the performance of FASP. Moreover, the available capacity on the link can change when other traffic such as GridFTP tries to utilize more bandwidth, which would also affect the sending rate of FASP.

The packet loss for TCP BBR was very low and it did not affect its goodput rate. This is because TCP BBR responds to delay as a measure for congestion and not packet loss, thus it does not reduce its sending rate when there is packet loss. The packet loss for QUIC was also very low and did not vary substantially between all the runs.

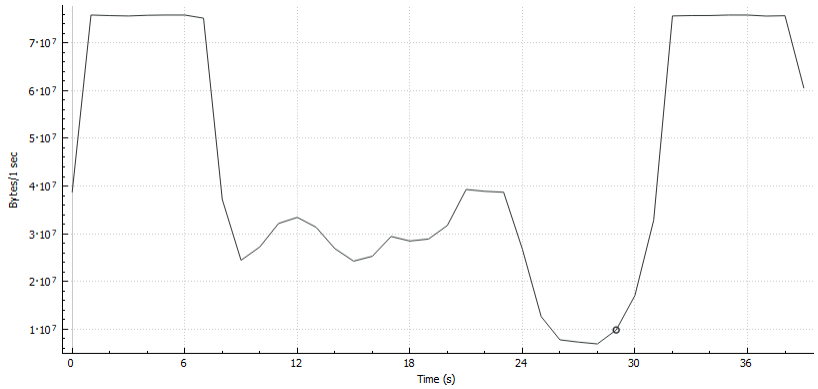


Figure 4.66: Sending Rate Over Time - FASP with GridFTP traffic - Local Setup (Replication 10)

4.4.2 Waterloo Testbed Results for Transfers with Competing Traffic

Figure 4.67 shows the aggregate goodput for each protocol in this setup with competing traffic. The goodput achieved by all the protocols is lower than the observed goodput in the local setup, except for TCP BBR. Compared to the single flow in experiment 1 for the same setup, the all the protocols achieve similar goodput results. Since the receiver machine showed some inefficiency with processing packets, the goodput of the protocols is still the same as the previous experiments for the Waterloo setup. Surprisingly, TCP BBR still achieves a high goodput rate as it utilizes more bandwidth. Also, the file transfer might have completed before the appropriate rate is identified due to delay. Both FASP and GridFTP for this setup had similar goodput results and their goodput for most of the runs does not reach more than 250 Mbps, which shows

that they are limited by the processing power of the machine at the receiver side. In addition, the competing traffic affects the performance of the protocols, as they compete for bandwidth. The performance of QUIC is also similar to the single flow experiments.

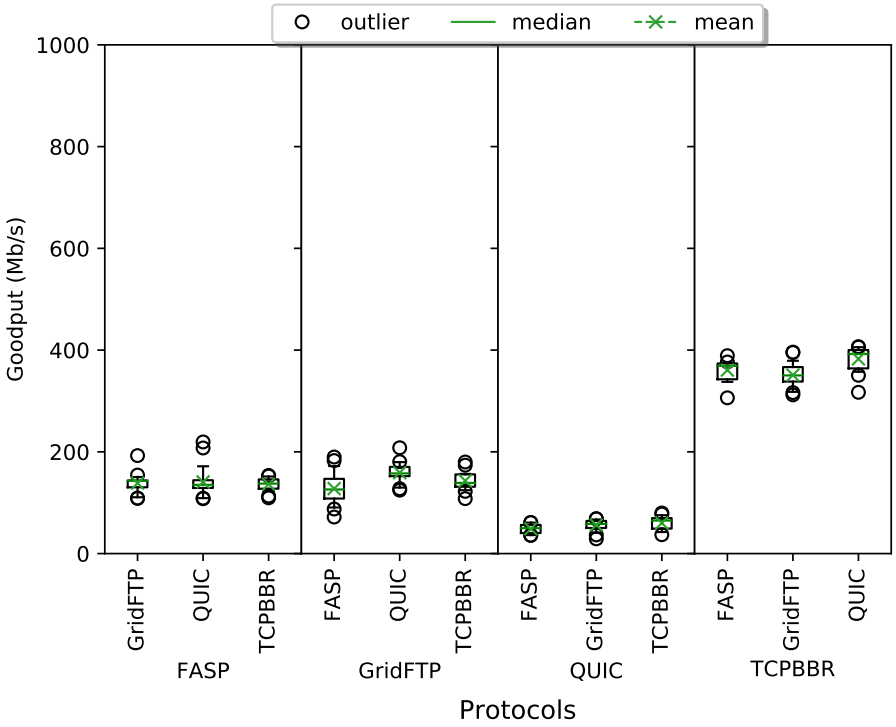


Figure 4.67: Aggregate Goodput for Each Protocol with Competing Traffic - Waterloo Setup

Figure 4.68 shows the aggregate packet loss results for each protocol with competing traffic in this setup. The packet loss is the highest for FASP and it is affected by all types of competing traffic. The high packet loss percentage in FASP's runs is caused by the firewall at the destination network. This was also observed in the previous experiments for the Waterloo setup. As explained earlier 4.1.2, FASP is aggressive with using bandwidth, which could also cause congestion on the link.

Figure 4.69 and 4.70 shows the goodput and packet loss results of each protocol for every individual run in this setup. The results show that FASP, GridFTP, and QUIC vary the most in terms goodput and packet loss between the runs. FASP is not able to achieve high goodput for most of the runs since it has high packet loss rates, which is caused by the firewall. GridFTP also has similar goodput values as that of FASP, but with low packet loss. Although the packet loss is not that high for GridFTP, it is still not able to achieve high goodput. This is possibly caused by the destination's processor, which is not efficient enough to handle all TCP flows used by GridFTP and this requires more processor cycles. This affects the data transfer flows of GridFTP, and reduces its performance. This behaviour was also observed in previous experiments. In addition, the goodput of GridFTP seems to be more affected by FASP than with TCP BBR. The same is observed with QUIC when FASP is the competing traffic. This shows that FASP is more aggressive and does

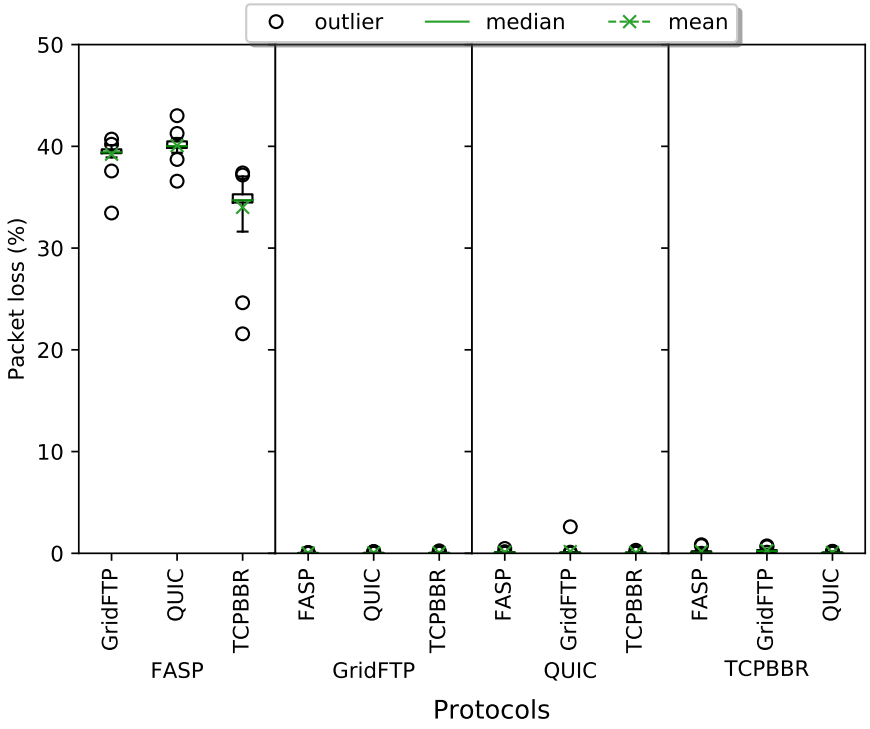
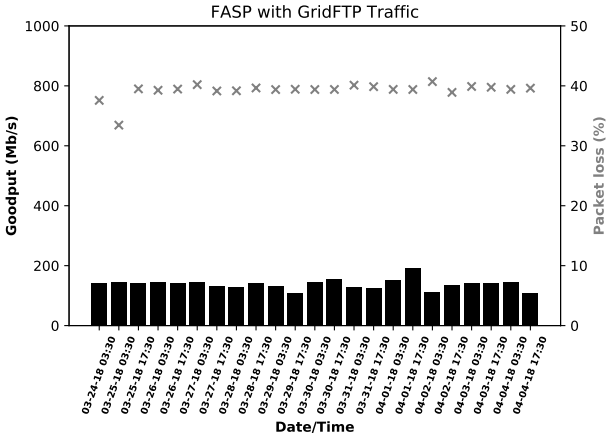
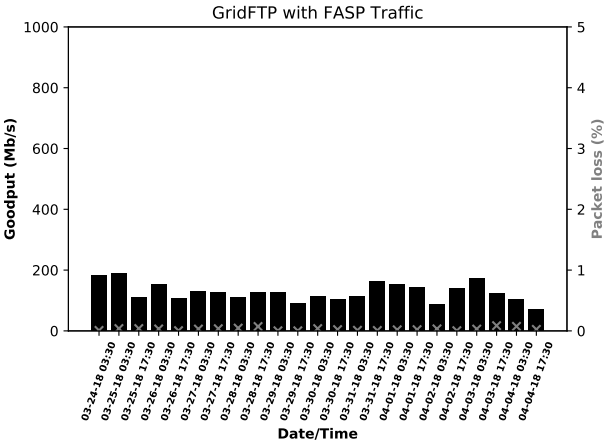


Figure 4.68: Aggregate Packet Loss for Each Protocol with Competing Traffic - Waterloo Setup

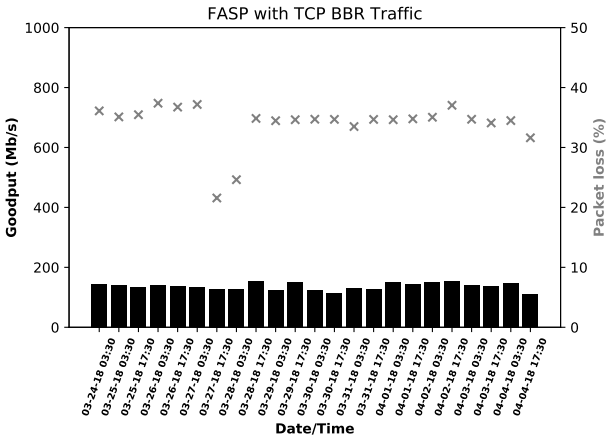
not share the bandwidth fairly with the other protocols. Although the packet loss for FASP was high, FASP was still able to achieve a high goodput for some runs compared to other runs. This was also observed in the local setup in which FASP's bandwidth over time decreased, but increased again in order to send data as fast as possible. Moreover, TCP BBR showed similar results for each run and had a more stable performance with all the competing protocols.



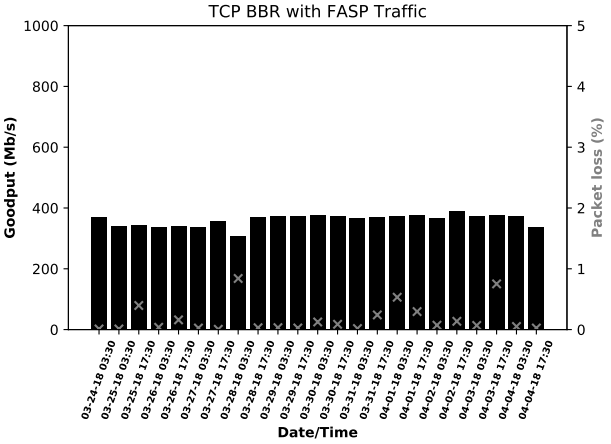
a) Goodput & Packet Loss for FASP with GridFTP Traffic



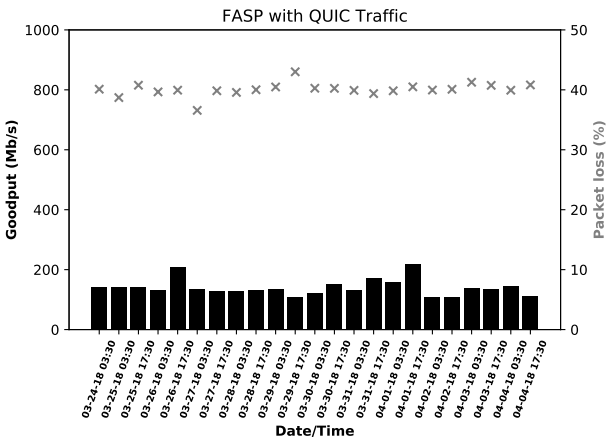
b) Goodput & Packet Loss for GridFTP with FASP Traffic



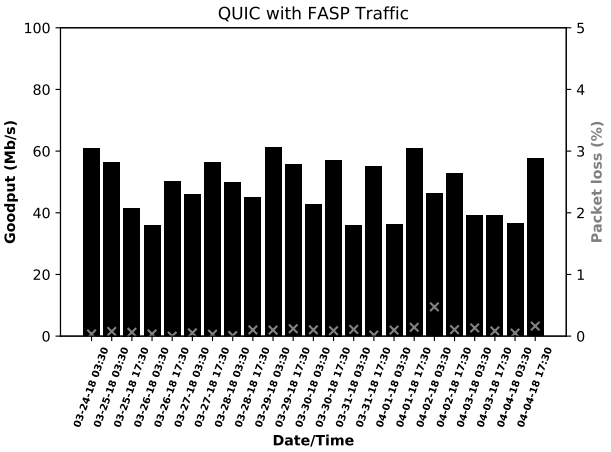
c) Goodput & Packet Loss for FASP with TCPBBR Traffic



d) Goodput & Packet Loss for TCPBBR with FASP Traffic

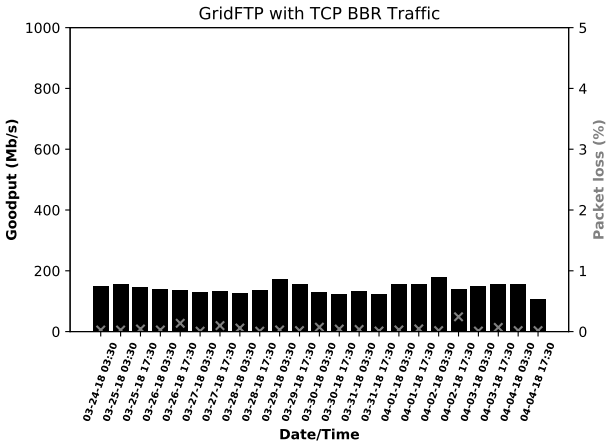


e) Goodput & Packet Loss for FASP with QUIC Traffic

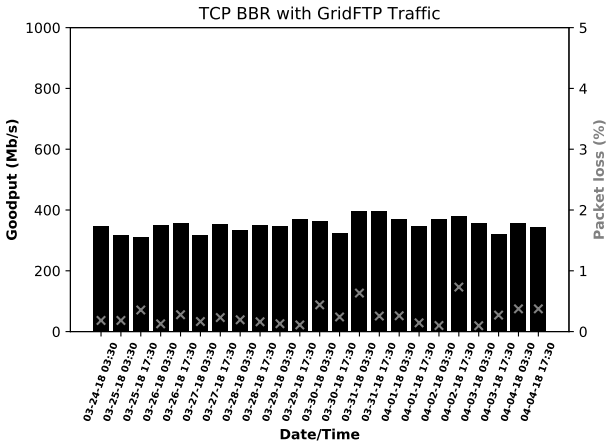


f) Goodput & Packet Loss for QUIC with FASP Traffic

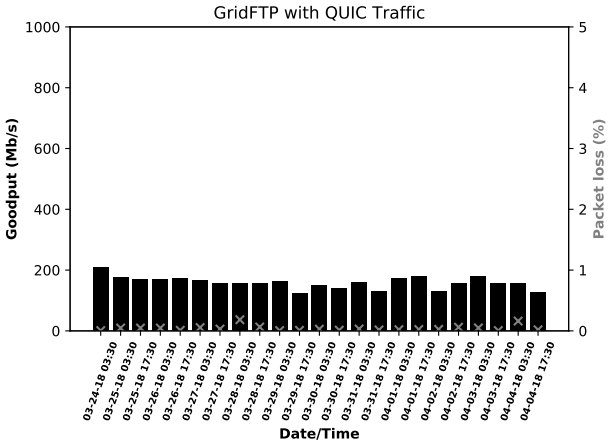
Figure 4.69: Goodput & Packet Loss for Protocols with Competing Traffic (1) - Waterloo Setup



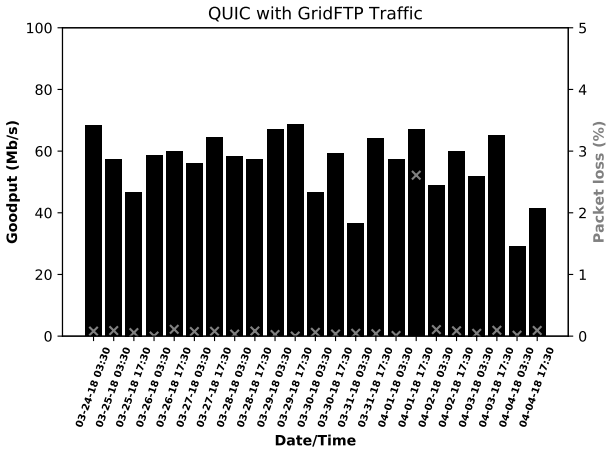
a) Goodput & Packet Loss for GridFTP with TCP BBR Traffic



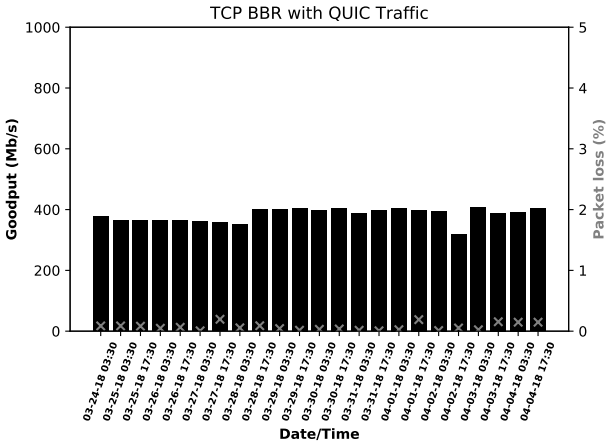
b) Goodput & Packet Loss for TCP BBR with GridFTP Traffic



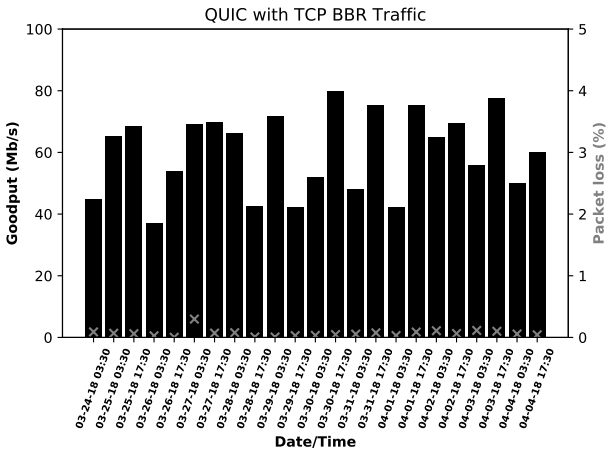
c) Goodput & Packet Loss for GridFTP with QUIC Traffic



d) Goodput & Packet Loss for QUIC with GridFTP Traffic



e) Goodput & Packet Loss for TCP BBR with QUIC Traffic



f) Goodput & Packet Loss for QUIC with TCP BBR Traffic

Figure 4.70: Goodput & Packet Loss for Protocols with Competing Traffic (2) - Waterloo Setup

4.4.3 Eastcloud Testbed Results for Transfers with Competing Traffic

Figure 4.71 shows the aggregate goodput for each protocol with competing traffic. The goodput results of FASP and GridFTP are again similar in this setup and they both have high goodput, which means that they their sending rate is also high. The results vary more in this setup, because of the increased RTT and congestion, which causes a higher variation in the sending rates for each protocol. Nevertheless, both FASP and GridFTP compete for more bandwidth aggressively, but they are able to utilize even more bandwidth when QUIC traffic is present. This is because QUIC is not aggressive in using the available bandwidth. The goodput of QUIC has decreased below 50 Mbps, because of the increased RTT and congestion on this link which also may affect all the protocols. Moreover, QUIC achieved similar goodput results with both FASP and GridFTP, but FASP was more aggressive with QUIC traffic.

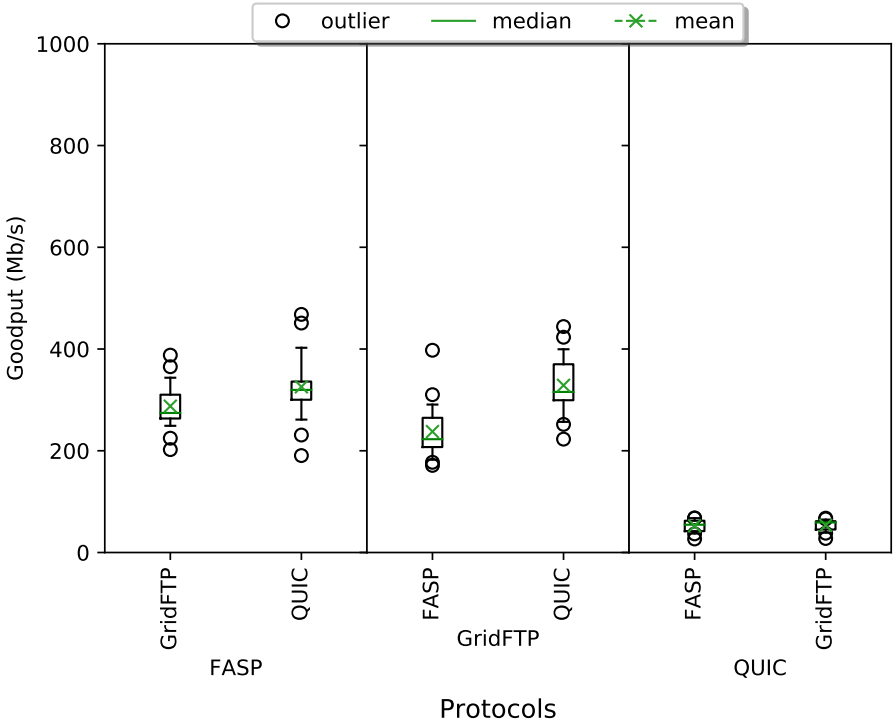


Figure 4.71: Aggregate Goodput for Each Protocol with Competing Traffic - Eastcloud Setup

Figure 4.72 shows that GridFTP has the highest aggregate packet loss in this setup. This shows that the competing traffic induces more congestion besides the congestion on this high RTT link, which causes GridFTP to lose more packets and to re-transmit these packets along with lowering its sending rate. Additionally, GridFTP uses multiple flows when transferring data and this causes the aggregate sending rate to increase, which causes congestion when it consumes all the available link capacity. Therefore, more packets are lost. For most of the runs, FASP experiences a low packet loss percentage, which shows that it is able to utilize more bandwidth efficiently on this link and there is no firewall that inspects packets. This is a low packet loss percentage compared to the local setup and Waterloo setup. The packet loss of QUIC is also

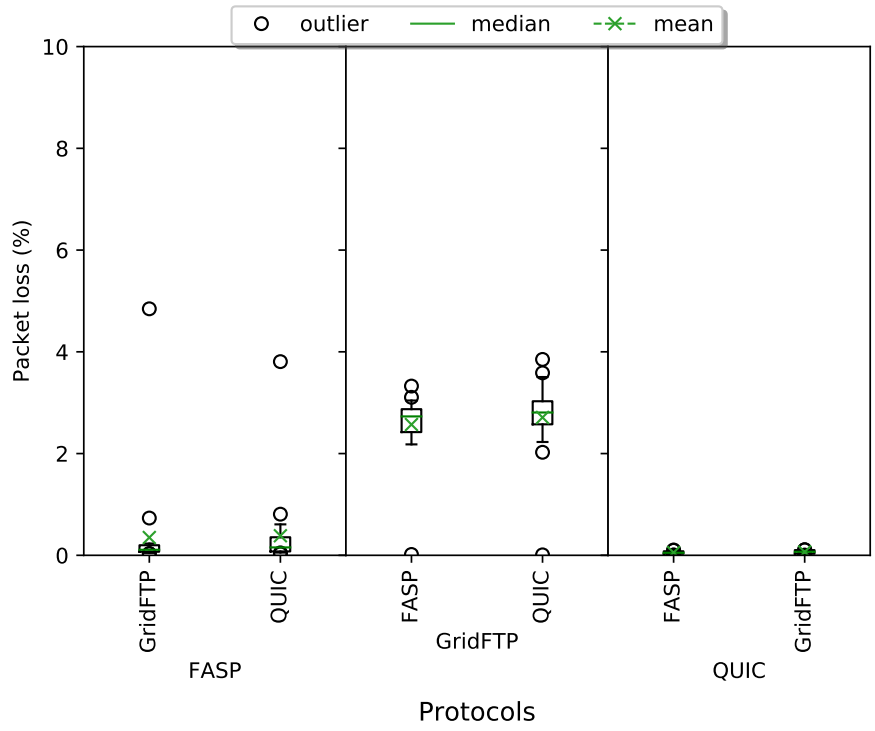


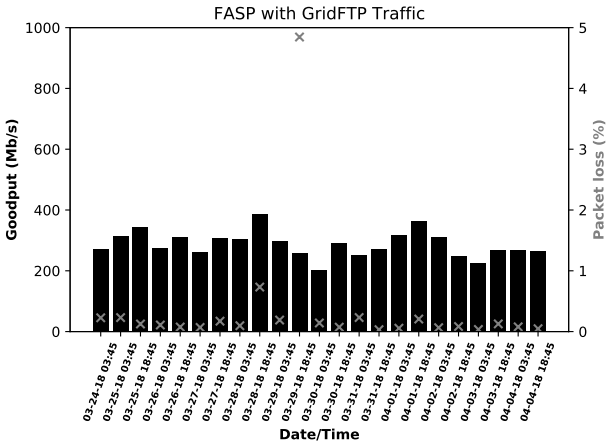
Figure 4.72: Aggregate Packet Loss for Each Protocol with Competing Traffic - Eastcloud Setup

similar to the packet loss observed in local and Waterloo testbeds.

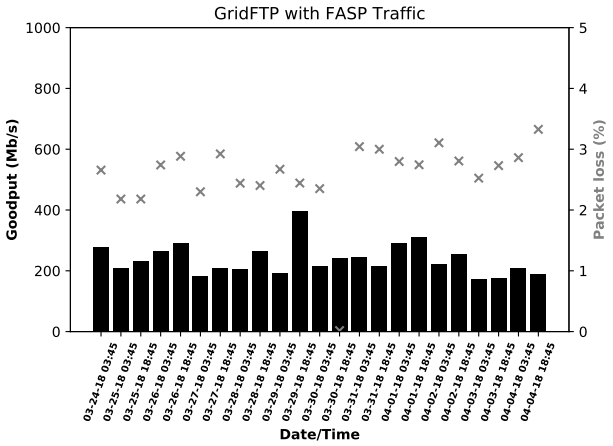
As can be seen in Figure 4.73, the goodput and packet loss results of each protocol is shown for every individual run. The goodput of FASP varied with GridFTP and QUIC, but was more with QUIC traffic. This shows that both FASP and GridFTP are trying to compete aggressively for available bandwidth, thus affecting the performance of each other. In addition, FASP achieves higher goodput with QUIC traffic, because QUIC is not utilizing a high bandwidth when competing with FASP. The packet loss of FASP is low for most of the runs with both types of competing traffic.

On the other hand, the goodput of GridFTP is lower than that of FASP and the results vary more between the runs. This behaviour could be caused by the congestion control mechanism, which reduces the sending rate in the presence of high packet loss, which is caused by either the congestion induced by other traffic on the link or the traffic generated by FASP and QUIC. GridFTP is affected more by FASP, since FASP is aggressive in using the available bandwidth; however, it performs better with QUIC traffic.

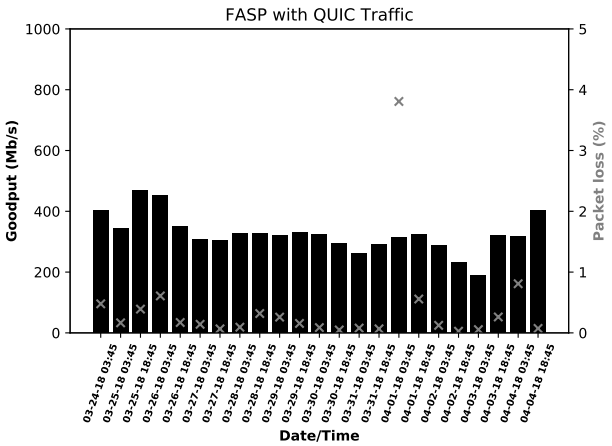
The performance of QUIC varied from one run to another. A similar behaviour is observed with FASP and GridFTP. This shows that the bandwidth utilization by both protocols affects the performance of QUIC, which fluctuates more frequently. Although a high variation is observed in the packet loss results, these results are not severe enough to degrade the performance of QUIC. In this setup, FASP performs better than the other protocols, since it had the highest goodput and the lowest packet loss for most of the runs. However, this also implies that FASP is not being fair to other protocols.



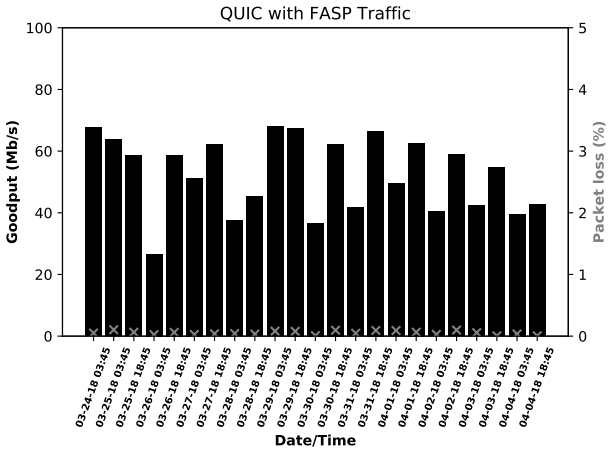
a) Goodput & Packet Loss for FASP with GridFTP Traffic



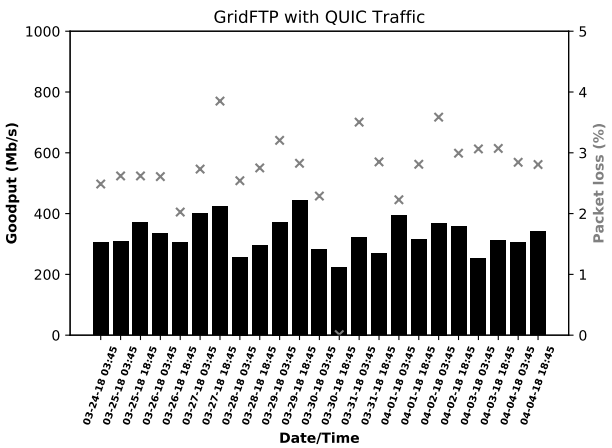
c) Goodput & Packet Loss for GridFTP with FASP Traffic



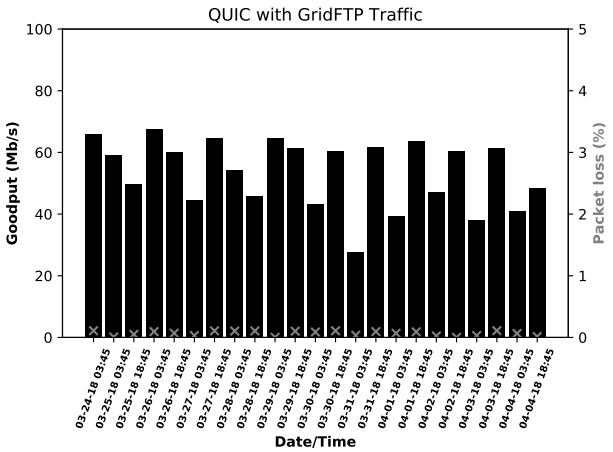
b) Goodput & Packet Loss for FASP with QUIC Traffic



d) Goodput & Packet Loss for QUIC with FASP Traffic



e) Goodput & Packet Loss for GridFTP with QUIC Traffic



f) Goodput & Packet Loss for QUIC with GridFTP Traffic

Figure 4.73: Goodput & Packet Loss for Protocols With Competing Traffic - Eastcloud Setup

4.4.4 International Testbed Results for Transfers with Competing Traffic

Figure 4.74 shows the aggregate goodput for each protocol in this setup. The aggregate goodput for FASP was the highest compared to the other protocols. This shows that FASP performs well in long RTT networks, but the goodput achieved in this setup reduced compared to previous setups. It seems that the long RTT on this link and the competing traffic affects the performance of FASP, but it still achieves similar aggregate goodput rates with all types of competing traffic. This shows that it has a stable performance on this long RTT link. The goodput achieved by all the protocols has decreased compared to the one in previous testbeds. In addition, GridFTP performs the best with FASP, but not with the TCP BBR and QUIC. QUIC and TCP BBR are causing more congestion, thus more packet loss, which affects the goodput of GridFTP in this setup.

Similar to previous setups, the performance of QUIC is low compared to the other protocols. Compared to single flow and multiple flows, QUIC had lower goodput with competing data transfer traffic, and this could be caused by QUIC trying to be fair to competing traffic so it decreases its sending rate further, which part of its congestion control algorithm. On the other hand, the goodput of TCP BBR has decreased for this setup, especially with QUIC traffic. This shows that it is not able to utilize more bandwidth on this long RTT path. Also, BBR's performance can degrade, because it is affected by delay which is present on this link, and potentially the overhead that QUIC produces on the sender's network interface.

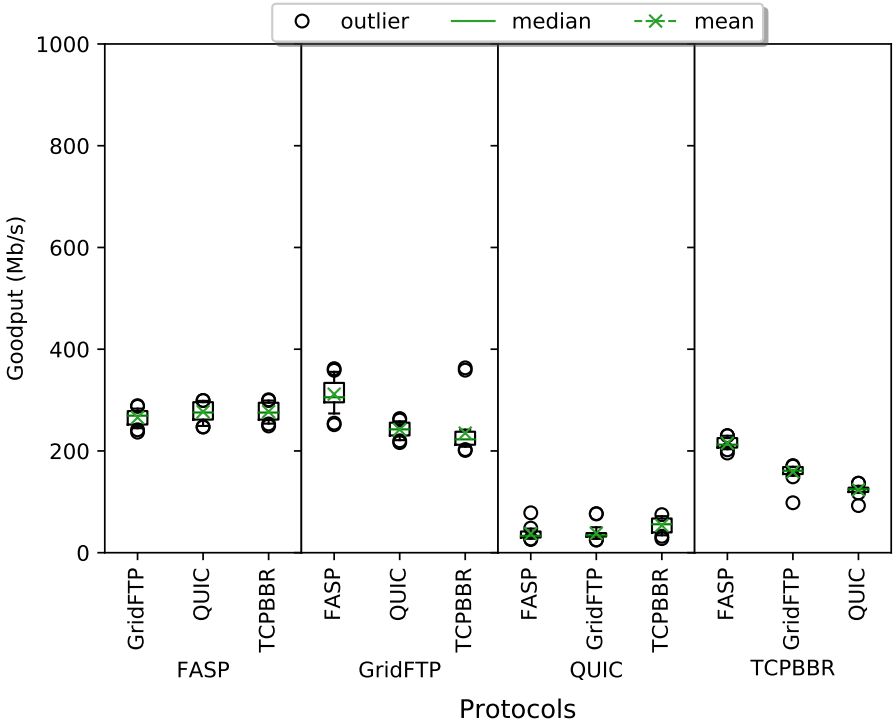


Figure 4.74: Aggregate Goodput for Each Protocol with Competing Traffic - Auckland Setup

Figure 4.75 shows the aggregate packet loss results for each protocol with competing traffic in this setup. All the protocols had some packet loss except for FASP, which barely had any packet loss. This shows that

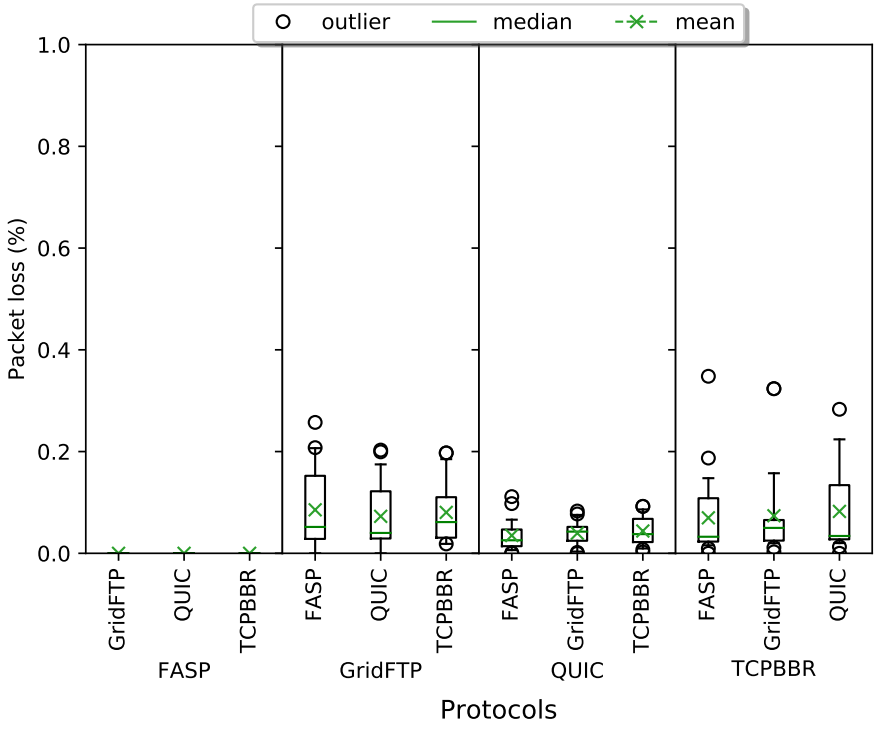
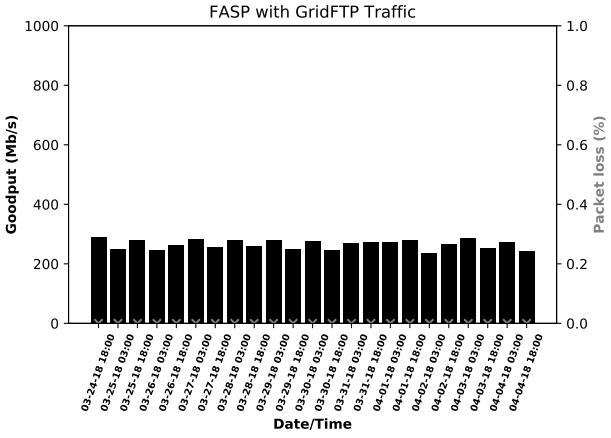


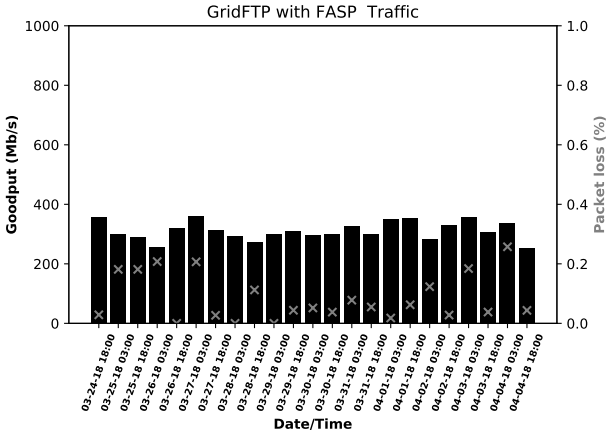
Figure 4.75: Aggregate Packet Loss for Each Protocol with Competing Traffic - Auckland Setup

FASP performs better on long RTT links. All the protocols seem to experience similar packet loss values for all types of competing traffic. The packet loss observed is caused partially by the competing traffic used, the amount of congestion, and delay on the network path. The packet loss observed is similar to previous experiments in the Auckland setup.

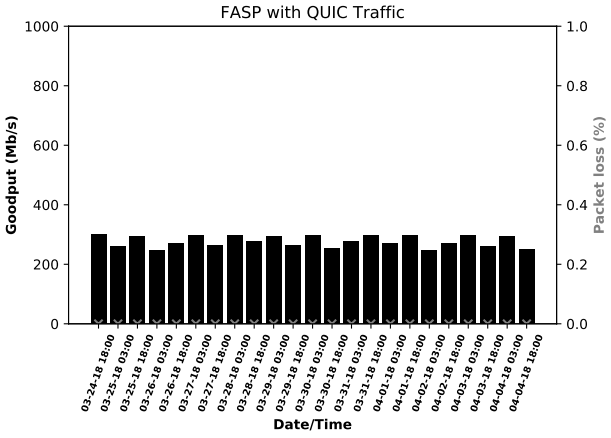
Figure 4.77 shows the goodput and packet loss results of some protocols for every individual run with competing traffic. The performance of GridFTP is affected the most by QUIC and TCP BBR, which is possibly because of QUIC and TCP BBR being unfair to GridFTP on this link. GridFTP's low goodput is caused by competing traffic, which causes congestion on the link and forces GridFTP to reduce its sending rate. In addition, the goodput of GridFTP can decrease if the destination machine is overloaded. The performance of QUIC was not stable between all the runs and it gets affected by all types of competing traffic. Moreover, the performance of FASP was more stable than the other protocols for all the runs. The results also show that FASP is able to reach high goodput for most of the runs with very low packet loss. This also shows that FASP is more resilient to packet loss on the high RTT links and is able to probe for more bandwidth, even with competing traffic. For TCP BBR, the results for all the runs were also stable and there was not significant variation observed in the results between the different runs.



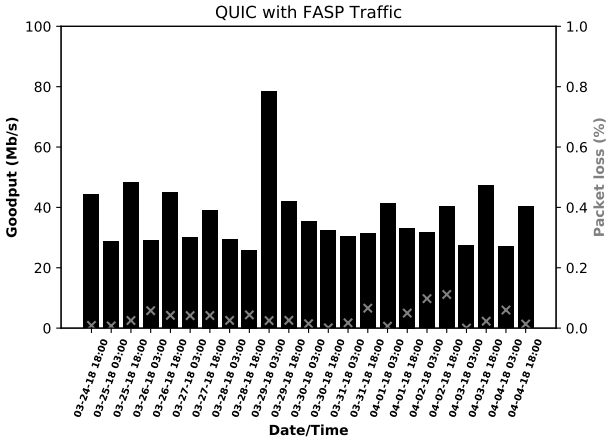
a) Goodput & Packet Loss for FASP with GridFTP Traffic



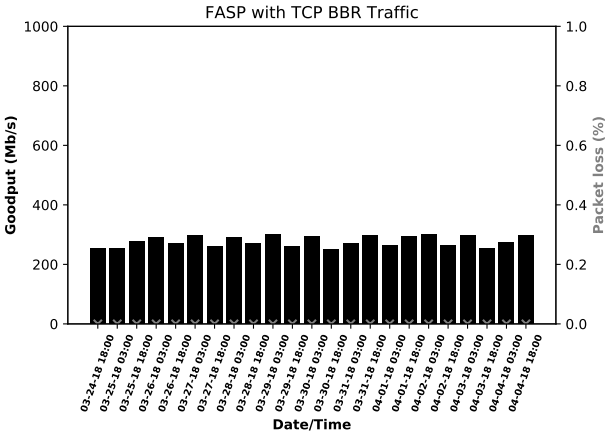
b) Goodput & Packet Loss for GridFTP with FASP Traffic



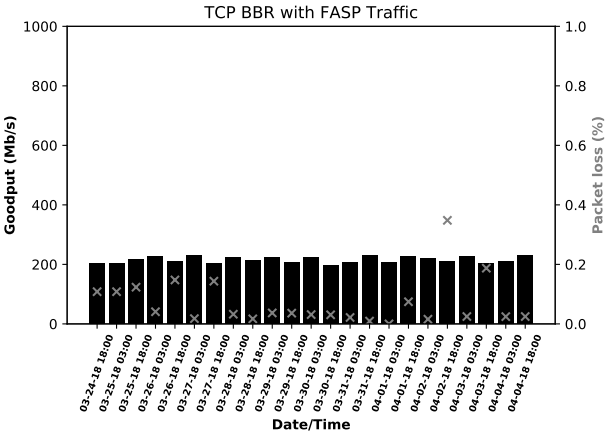
c) Goodput & Packet Loss for FASP with QUIC Traffic



d) Goodput & Packet Loss for QUIC with FASP Traffic

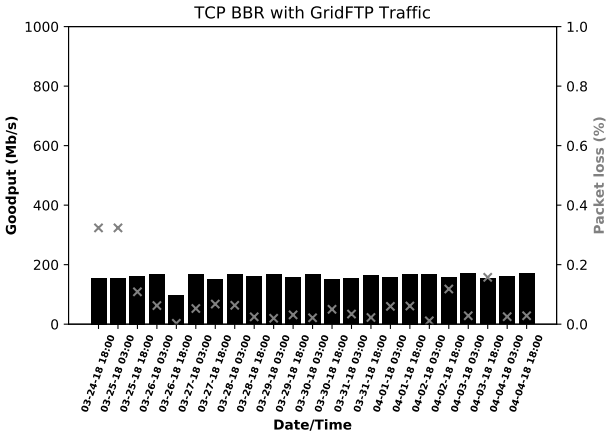


e) Goodput & Packet Loss for FASP with TCP BBR Traffic

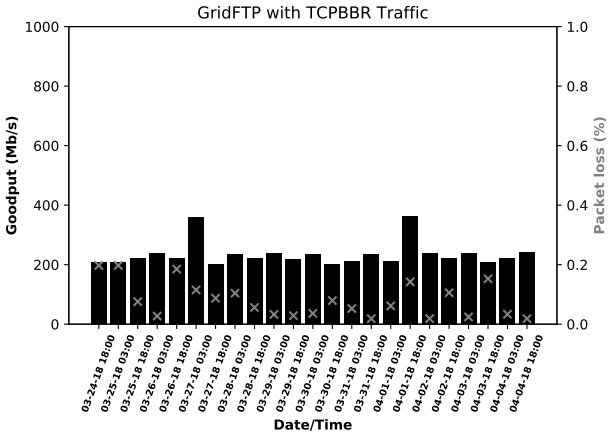


f) Goodput & Packet Loss for TCP BBR with FASP Traffic

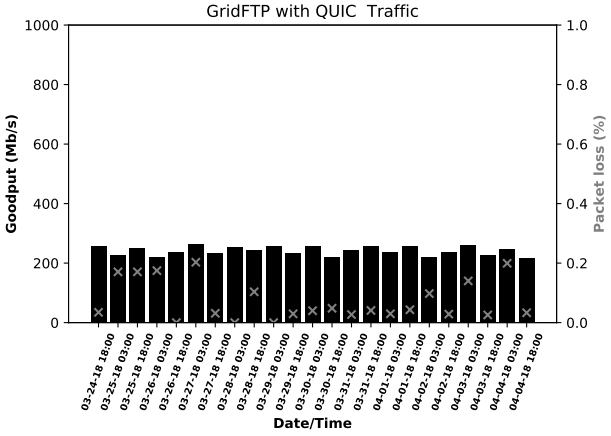
Figure 4.76: Goodput & Packet Loss for Protocols with Competing Traffic (1) - Auckland Setup



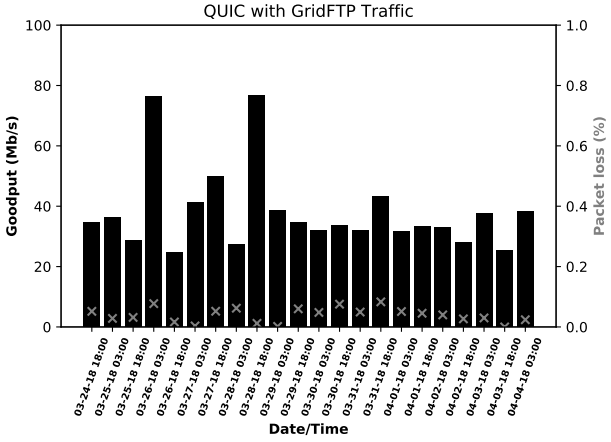
a) Goodput & Packet Loss for TCP BBR with GridFTP Traffic



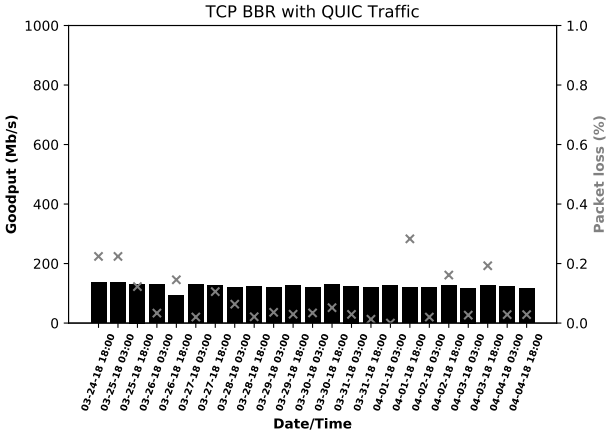
b) Goodput & Packet Loss for GridFTP with TCP BBR Traffic



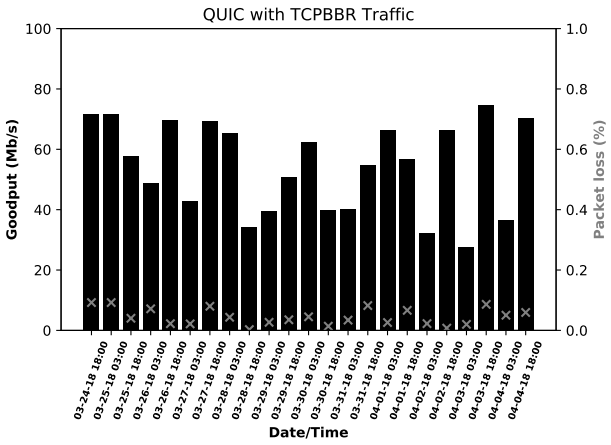
c) Goodput & Packet Loss for GridFTP with QUIC Traffic



d) Goodput & Packet Loss for QUIC with GridFTP Traffic



e) Goodput & Packet Loss for TCP BBR with QUIC Traffic



f) Goodput & Packet Loss for QUIC with TCP BBR Traffic

Figure 4.77: Goodput & Packet Loss for Protocols with Competing Traffic (2) - Auckland Setup

4.4.5 CPU Utilization

Table 4.10 shows the percentage of the user mode CPU utilization for each protocol when transferring various file sizes. QUIC had the highest CPU usage among all the protocols. It also uses an encryption algorithm, which might not be as optimized as the other protocols and this can consume more CPU. Furthermore, FASP also had high CPU utilization on the sender side and this could also be caused by the encryption of UDP packets. On the receiver end, the CPU utilization is not as high as the sender since it does not need to write data to the disk.

The CPU utilization of GridFTP and TCP BBR is low compared to the other protocols. More detailed knowledge of hardware resource consumption is required to determine the reason behind the high CPU utilization for FASP and QUIC. This can be explored in a future work.

Table 4.10: CPU Usage (%) for Each Protocol

File Size & Host		**FASP**	**GridFTP**	**QUIC**	**TCPBBR**
200 MB - CPU (%)	Sender	57.6	9.7	97	18
	Receiver	22.7	15.48	8.5	3.3
1 GB - CPU (%)	Sender	54.7	7.4	96	3.5
	Receiver	30.45	17.8	6.67	3.3
5 GB - CPU (%)	Sender	46.69	4.45	98	18.85
	Receiver	28.78	20.49	19.3	5.5

4.5 Summary and Analysis of Results

In this Chapter, various experiments were conducted using local, national and international testbeds. Using these testbeds, the average goodput and packet loss was measured during each transfer for GridFTP, FASP, QUIC and TCP BBR as permitted by the configuration and initial experimentation. These protocols were tested using various file sizes, background traffic, multiple flows, and competing traffic.

One of the main observations was FASP and GridFTP performed better in long distance transfers compared to TCP BBR and QUIC, especially when the RTT is more than 100 ms. TCP BBR performed better than the other protocols for short distance transfers, because of the low delay on the network. The performance of QUIC improves slightly in each setup, but does not reach high goodput compared to the other protocols due to implementation limitations that could not be modified. Moreover, GridFTP and FASP took a long time to ramp up to a high sending rate when transferring various file sizes with GridFTP taking longer in the Auckland setup. This behaviour can be investigated further in a future work.

Most of the protocols used were not aggressive to the background traffic except for FASP, which was aggressive in utilizing bandwidth in the presence of background traffic. This also affected the goodput of the

background traffic, especially for the TCP background traffic. In some cases, GridFTP was also aggressive in utilizing the bandwidth as it used multiple flows. The performance of TCP BBR and QUIC decreased in the presence of background traffic compared to the single file transfer experiments. On the other hand, the background traffic itself can be aggressive and affects the performance of the protocols depending on the traffic generator used. UDP traffic seems to be more aggressive than TCP with the protocols since it tries to send data as fast as it can. For competing traffic, FASP and GridFTP were still aggressively competing for bandwidth and were unfair to competing traffic.

The main benefit of using multiple flows is overcoming small buffer size and increasing the aggregate sending rate for the TCP-based protocols. In addition, the usage of multiple flows can be useful with the BDP is high. Based on the multiple flows experiments, GridFTP performed better with multiple flows in all the testbed setups compared to the other protocols, as it is able to increase its sending rate across all the flows, especially the flows that do not experience high packet loss. Additionally, multiple flows also allowed TCP flows to increase their congestion window size at the same time, which causes the aggregated congestion window to grow faster. In some cases, the flows can get packet loss at the same time, and reach the maximum sending rate at the same time. In other cases, some flows might experience packet loss while others don't which results in TCP flows having different sending rates with some having higher sending rates than others. This helps GridFTP in utilizing other flows that do not experience packet loss to send data as fast as it can.

The performance of FASP was steady across all the flows, because it has a limited target sending rate. This helps in controlling the rate at which FASP should send data over a bottleneck link. Additionally, the target rate of FASP for multiple flows should be set by the user to the link bandwidth divided by the number of flows used in order to observe low packet loss. With that being said, the performance of FASP also improves on long distance networks with the use of multiple flows compared to the other protocols. This not the case for QUIC, because it keeps a low sending rate even if there is more available bandwidth on the link. Furthermore, the performance of TCP BBR utilized nearly the full link bandwidth with 4 and 6 flows, but it decreased as the distance increased. Furthermore, the CPU usage was the highest for QUIC, because of its inefficient implementation. FASP also had a high CPU utilization, because of the mechanisms it uses to start a connection and to load the data from disk.

In many cases, the performance of protocols changes based on the various network conditions. In addition, some of these protocols can be aggressive in their approach, which could affect the performance of other traffic sharing the network path. Traffic that is TCP-based can be limited by the TCP socket buffer size, which can prevent TCP flows from reaching high sending rates. A high amount of congestion can cause packets to be dropped at the destination host, and reduce the performance of the protocols. The delay on the network was also another factor in affecting the performance of the protocols. Overall, high packet loss was observed with FASP followed by GridFTP in some testbeds, since these protocols are aggressive with using bandwidth and this causes high levels of congestion for other traffic. The network setup can also impact the performance of the protocols with different types of experiments.

Chapter 5

Conclusions

5.1 Thesis Summary

In the existing literature, the performance of the protocols used in this study was not compared side-by-side in real world scenarios. It is important to evaluate the performance of these protocols in situations that the users are likely to encounter on a real world network. This helps the users to know whether these protocols meet their requirements and to understand the benefits and limitations of these protocols in various setups. The various setups were used to run several experiments that tested the performance of several protocols FASP, GridFTP, QUIC, and TCP BBR.

The experiments revealed some interesting results. Firstly, the performance of the protocols was mainly affected by the distance and the various network conditions present in each testbed. Secondly, FASP and GridFTP had similar performance in most of the scenarios while in some scenarios FASP was considered had better performance. FASP needs to be properly configured depending on the network environment used and when using multiple flows in order to be less aggressive. Additionally, FASP performed better on the long RTT link because it is UDP-based and does not use TCP slow-start or congestion avoidance, which allows it to have faster ramp-up times than TCP slow-start. TCP BBR was able to utilize the highest bandwidth in most of the short distance setups, but its performance was degraded as the distance increased. This is because the long-distance setup had a large BDP and a high RTT. Thirdly, FASP was more aggressive than the other protocols in the presence of background and competing traffic. Fourthly, having multiple flows provided better performance for the TCP-based protocols, because it allowed the congestion window size to grow faster for all the flows than when running a single flow. However, there is also a possibility that more control overhead is produced when multiple flows are used, which may create more congestion and more packet loss events.

Furthermore, the host machines can be a transport bottleneck and degrade the performance of the protocols. In some cases, the processing speed of the system may be slower than the network transfer speed, which affects the processing of packets as they arrive at the host's network interface. Additionally, a firewall would also affect the performance of the protocols by dropping packets that might be causing flooding on the network. This was observed previously with FASP in waterloo setup. Another reason that causes the performance of the protocols to degrade having a physical system where multiple virtual machines are not

fairly sharing the CPU.

Overall, the experiments performed in this thesis indicated that FASP and GridFTP are two promising data transfer protocols for high performance long distance data transfers with GridFTP being a better candidate. Overall, GridFTP showed a more stable performance compared to the other protocols. FASP was better in some cases such as in the Auckland testbed, but GridFTP did quite well in EastCloud, especially for larger file sizes. BBR did well in local and Waterloo testbeds. There were also some poor results in some scenarios, especially for QUIC and TCP BBR.

5.2 Thesis Contributions

The goal of this work is to contribute to the area of network data transfer by evaluating the performance of several data transfer protocols in various scenarios. These scenarios involved examining the performance of the protocols in terms of goodput and packet loss with varying transfer distances. This would help the research community in understanding the performance of the data transfer protocols in various network conditions. Data such as crop images was collected on daily basis and large amounts of images are required to be transferred to various research sites for further analysis. This is a critical process and requires efficient high-speed data transfer protocols that can handle transfers at high speeds with the least packet loss possible. Moreover, experiments were conducted in a real world network to help understand the performance of these data transfer protocols and provided a more realistic study.

The first contribution is an evaluation of several data transfer protocols [1, 26, 47] and congestion control mechanisms [7, 43] that were not compared side-by-side in a real 1 Gb/s network environment. Some of these protocols were evaluated independently using emulated network conditions and were not evaluated for long distance transfers with large file sizes [17, 23, 45]; therefore, their performance is evaluated in a real network environment with different types of file transfers.

The second contribution included examining the performance of each protocol in various network environments for short-distance and long-distance transfers by running experiments in local, national and international testbeds. In each testbed, the protocols were tested with background traffic, multiple flows, and competing traffic. These scenarios showed the performance of each protocol in various distances, round trip times, and link capacities. The performance of the protocols was evaluated based on their Goodput and Packet loss rates.

5.3 Future Work

There are many possible ways to extend this work. A possible contribution could involve evaluating the performance of the protocols in an isolated network with no other traffic present on the path. Moreover, experiments can be performed on higher speed networks, at 10 Gb/s and 40 Gb/s, to understand the performance of the protocols further. This would be beneficial to examine, because the network capacities keep

increasing rapidly and it would be interesting to observe how these protocols perform on higher speed networks. Additionally, using high performance machines with more powerful processors and high speed network cards would improve the performance of the protocols. Further analysis of the performance of each protocol could be beneficial, as only the goodput and packet loss were measured for each protocol. Other metrics can be used such as throughput and delay to analyze the performance of the protocols.

A similar study can be conducted using other types of protocols or using optimized versions of the chosen protocols in this work, which would provide further valuable insight to the data transfer research community. Additionally, more insight can be provided on the relative performance of transferring many small files and a small number of large files using data transfer protocols. The performance of the protocols can be further explored by comparing results of day and night transfers and conducting further analysis.

Since QUIC had poor performance and had high CPU utilization, running experiments with a more stable version of QUIC is suggested to see if any improvements were made to the its implementation. In addition, analyzing the performance of additional TCP variants would be beneficial. Since it was also observed that the traffic generated by iperf was aggressive and consumed too much CPU, it would beneficial if other types of background traffic generators are used. This can also be used to observe if there is any difference in the performance of the protocols with different types of traffic from other generators.

Since the system resource consumption by protocols was briefly explored in this thesis, it would be an interesting topic further explore. Another suggestion would be to discover techniques that would improve the performance of endpoints to increase the network utilization. This should also help in the performance of the protocols.

References

[1] William Allcock, Joseph Bester, John Bresnahan, Ann Chervenak, L Liming, and Steven Tuecke. GridFTP: Protocol extensions to FTP for the grid. *Global Grid Forum*, page 5, 2003.

[2] William Allcock, John Bresnahan, Rajkumar Kettimuthu, Michael Link, Catalin Dumitrescu, Ioan Raicu, and Ian Foster. The Globus Striped GridFTP Framework and Server. In *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, pages 1–11, Seattle, WA, 2005.

[3] Andrea Baiocchi, Angelo P Castellani, and Francesco Vacirca. YeAH-TCP: yet another highspeed TCP. In *Proceedings of the fifth International Workshop on Protocols for Fast Long-Distance Networks*, volume 7, pages 37–42, Los Angeles, CA, 2007.

[4] Martin Bateman, Saleem Bhatti, Greg Bigwood, Devan Rehunathan, Colin Allison, Tristan Henderson, and Dimitrios Miras. A comparison of TCP behaviour at high speeds using ns-2 and Linux. In *Proceedings of the 11th Communications and Networking Simulation Symposium*, pages 30–37, Ottawa, Canada, April 2008.

[5] Lawrence S Brakmo, Sean W O'Malley, and Larry L Peterson. Tcp vegas: New techniques for congestion detection and avoidance. In *Proceedings of the SIGCOMM Conference on Communications Architectures, Protocols and Applications*, pages 24–35, London, UK, August 1994.

[6] John Bresnahan, Michael Link, Rajkumar Kettimuthu, and Ian Foster. UDT as an alternative transport protocol for gridFTP. In *Proceedings of the International Workshop on Protocols for Future, Large-Scale and Diverse Network Transports (PFLDNeT)*, pages 21–22, Tokyo, Japan, 2009.

[7] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. BBR: Congestion-based congestion control. *ACM Queue*, 14(5):50, October 2016.

[8] Quentin De Coninck and Olivier Bonaventure. Multipath QUIC: Design and evaluation. In *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*, pages 160–166, Incheon, Republic of Korea, 2017.

[9] Mo Dong, Qingxi Li, Doron Zarchy, Philip Brighten Godfrey, and Michael Schapira. PCC: Re-architecting congestion control for consistent high performance. In *proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation*, pages 395–408, Oakland, CA, May 2015.

[10] Ben Eckart, Xubin He, and Qishi Wu. Performance adaptive UDP for high-speed bulk data transfer over dedicated links. In *2008 IEEE International Symposium on Parallel and Distributed Processing*, pages 1–10, Miami, FL, 2008.

[11] Kevin Fall and Sally Floyd. Simulation-based comparisons of Tahoe, Reno, and SACK TCP. *ACM SIGCOMM Computer Communication Review*, pages 5–21, July 1996.

[12] Sally Floyd. Highspeed TCP for large congestion windows. Technical report, 2003.

[13] Sally Floyd and Vern Paxson. Difficulties in simulating the internet. *IEEE/ACM Transactions on Networking (ToN)*, 9(4):392–403, August 2001.

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

[15] Eric He, Jason Leigh, Oliver Yu, and Thomas A DeFanti. Reliable blast UDP: Predictable high performance bulk data transfer. In *Proceedings of the 2002 IEEE International Conference on Cluster Computing*, pages 317–324, Chicago, IL, 2002.

[16] Tom Henderson, Sally Floyd, Andrei Gurtov, and Yoshifumi Nishida. The NewReno modification to TCP's fast recovery algorithm. Technical report, April 2012.

[17] Mario Hock, Roland Bless, and Martina Zitterbart. Experimental evaluation of BBR congestion control. In *Proceedings of the 25th IEEE International Conference on Network Protocols*, pages 1–10, Toronto, Canada, October 2017.

[18] Mario Hock, Felix Neumeister, Martina Zitterbart, and Roland Bless. TCP lola: Congestion control for low latencies and high throughput. In *Proceedings of the 2017 IEEE 42nd Conference on Local Computer Networks*, pages 215–218, Singapore, Singapore, October 2017.

[19] Chromium QUIC Implementation. Google Inc. `https://cs.chromium.org/chromium/src/net/quic/`.

[20] Van Jacobson. Congestion avoidance and control. *ACM SIGCOMM Computer Communication Review*, 18(4):314–329, August 1988.

[21] Cheng Jin, David Wei, Steven H Low, Gary Buhrmaster, Julian Bunn, Dawn H Choe, RLA Cottrell, Johe C Doyle, Harvey Newman, Fernando Paganini, and S. Ravot. FAST kernel: Background theory and experimental results. In *Proceedings of the First International Workshop on Protocols for Fast Long-Distance Networks*, pages 3–4, Geneva, Switzerland, February 2003.

[22] Dmitry Kachan, Eduard Siemens, and Vyacheslav Shuvalov. Comparison of contemporary solutions for high speed data transport on WAN 10 Gbit/s connections. *Journal of Communication and Computer*, 10(6):783–795, 2013.

[23] Arash Molavi Kakhki, Samuel Jero, David Choffnes, Cristina Nita-Rotaru, and Alan Mislove. Taking a long look at QUIC: an approach for rigorous evaluation of rapidly evolving transport protocols. In *Proceedings of the 2017 Internet Measurement Conference*, pages 290–303, London, UK, November 2017.

[24] Tom Kelly. Scalable TCP: Improving performance in highspeed wide area networks. *ACM SIGCOMM computer communication Review*, 33(2):83–91, 2003.

[25] Kazumi Kumazoe, Masato Tsuru, and Yuji Oie. Performance of high-speed transport protocols coexisting on a long distance 10-gbps testbed network. In *Proceedings of the First International Conference on Networks for Grid Applications*, pages 2:1–2:8, Lyon, France, October 2007.

[26] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, Jeff Bailey, Jeremy Dorfman, Jim Roskind, Joanna Kulik, Patrik Westin, Raman Tenneti, Robbie Shade, Ryan Hamilton, Victor Vasiliev, Wan-Teh Chang, and Zhongyi Shi. The QUIC transport protocol: Design and internet-scale deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 183–196, Los Angeles, CA, August 2017.

[27] Iosif Legrand, Harvey Newman, Ramiro Voicu, Catalin Cirstoiu, Costin Grigoras, Ciprian Dobre, Adrian Muraru, Alexandru Costan, Mihaela Dediu, and Corina Stratan. MonALISA: An agent based, dynamic service system to monitor, control and optimize distributed systems. *Computer Physics Communications*, 180(12):2472–2498, 2009.

[28] Douglas Leith and Robert Shorten. H-TCP: TCP for high-speed and long-distance networks. In *Proceedings of the third International Workshop on Protocols for Fast Long-Distance Networks*, pages 1–16, Lyon, France, Feb 2004.

[29] Roger Les Cottrell, Saad Ansari, Parakram Khandpur, Ruchi Gupta, Richard Hughes-Jones, Michael Chen, Larry McIntosh, and Frank Leers. Characterization and evaluation of tcp and udp-based transport on real networks. *Annales des télécommunications*, 61(1-2):5–20, February 2006.

[30] Yee-Ting Li, Douglas Leith, and Robert N Shorten. Experimental evaluation of tcp protocols for high-speed networks. *IEEE/ACM Transactions on Networking (ToN)*, 15(5):1109–1122, October 2007.

[31] Saverio Mascolo, Luigi Alfredo Grieco, Roberto Ferorelli, Pietro Camarda, and Giacomo Piscitelli. Performance evaluation of Westwood+ TCP congestion control. *Performance Evaluation*, 55(1-2):93–111, 2004.

[32] Mark R Meiss et al. Tsunami: A high-speed rate-controlled protocol for file transfer. pages 1–10, 2004.

[33] Serban Munson and Michelle Christine. Method and system for reliable data transfer. `https://www.google.com/patents/US8214707`, July 2012.

[34] Serban Munson and Michelle Christine. Method and system for reliable data transfer. `https://www.google.com/patents/US8583977`, November 2013.

[35] Serban Munson and Michelle Christine. Bulk data transfer. `https://www.google.com/patents/US8996945`, March 2015.

[36] Truc Anh N Nguyen, Siddharth Gangadhar, and James PG Sterbenz. Performance evaluation of tcp congestion control algorithms in data center networks. In *Proceedings of the 11th International Conference on Future Internet Technologies*, pages 21–28, Nanjing, China, June 2016.

[37] Jon Postel. User datagram protocol. STD 6, RFC Editor, 1980. `http://www.rfc-editor.org/rfc/rfc768.txt`.

[38] Jon Postel and Joyce Reynolds. File transfer protocol. STD 9, RFC Editor, 1985. `http://www.rfc-editor.org/rfc/rfc959.txt`.

[39] K Satyanarayan Reddy and C Lokanatha Reddy. A survey on congestion control protocols for high speed networks. *International Journal of Computer Science and Network Security*, 8(7):44–53, July 2008.

[40] Jan Rüth, Ingmar Poese, Christoph Dietzel, and Oliver Hohlfeld. A first look at QUIC in the wild. In *International Conference on Passive and Active Network Measurement*, pages 255–268, Berlin, Germany, 2018.

[41] Ha Sangtae, Yusung Kim, Long Le, Injong Rhee, and Lisong Xu. A step toward realistic performance evaluation of high-speed TCP variants. In *Proceedings of the Fourth International Workshop on Protocols for Fast Long-Distance Networks*, pages 1–8, Nara, Japan, Feb 2006.

[42] Ha Sangtae, Yusung Kim, Long Le, Injong Rhee, and Lisong Xu. Impact of background traffic on performance of high-speed TCP variant protocols. *Computer Networks*, 51(7):1748–1762, May 2007.

[43] Sea Shalunov, Greg Hazel, Janardhan Iyengar, and Mirja Kuehlewind. Low extra delay background transport (LEDBAT). RFC 6817, RFC Editor, 2012. `http://www.rfc-editor.org/rfc/rfc6817.txt`.

[44] Marko Sosic and Vladimir Stojanovic. Resolving poor tcp performance on high-speed long distance linksoverview and comparison of BIC, CUBIC and Hybla. In *Proceedings of the 11th IEEE International Symposium on Intelligent Systems and Informatics (SISY)*, pages 325–330, Subotica, Serbia, 2013.

[45] Joshua Suresh, Aravinda Srinivasan, and Avula Damodaram. Performance analysis of various high speed data transfer protocols for streaming data in long fat networks. In *Proceedings of the International Conference on Recent Trends in Information, Telecommunication and Computing*, pages 234–237, Kochi, India, May 2010.

[46] Andrew Tanenbaum and David Wetherall. *Computer Networks*. Prentice Hall Press, 5th edition, 2011.

[47] Aspera Technology. FASP Overview. Technical report, Aspera Technology. `http://asperasoft.com/technology/transport/fasp#overview-464`, 2013.

[48] Jörg Widmer, Robert Denda, and Martin Mauve. A survey on tcp-friendly congestion control. *IEEE Network*, pages 28–37, May 2001.

[49] Lisong Xu, Ha Sangtae, and Injong Rhee. Cubic: a new TCP-friendly high-speed TCP variant. *ACM SIGOPS Operating Systems Review*, pages 64–74, July 2008.

[50] Lin Xue, Suman Kumar, Cheng Cui, and Seung-Jong Park. A study of fairness among heterogeneous TCP variants over 10Gbps high-speed optical networks. *Optical Switching and Networking*, pages 124–134, July 2014.

[51] Se-Young Yu, Nevil Brownlee, and Aniket Mahanti. Comparative performance analysis of high-speed transfer protocols for big data. In *Proceedings of the 38th IEEE Conference on Local Computer Networks (LCN)*, pages 292–295, Sydney, Australia, October 2013.

[52] Se-Young Yu, Nevil Brownlee, and Aniket Mahanti. Characterizing performance and fairness of big data transfer protocols on long-haul networks. In *Proceedings of the 40th IEEE Conference on Local Computer Networks(LCN)*, pages 213–216, Clearwater Beach, FL, October 2015.

[53] Se-Young Yu, Nevil Brownlee, and Aniket Mahanti. Comparative analysis of big data transfer protocols in an international high-speed network. In *Proceedings of the 34th IEEE International Performance Computing and Communications Conference (IPCCC)*, pages 1–9, Nanjing, China, December 2015.

[54] Zhaojuan Yue, Yongmao Ren, and Jun Li. Performance evaluation of udp-based high-speed transport protocols. In *Proceedings of the 2011 IEEE 2nd International Conference on Software Engineering and Service Science*, pages 69–73, Beijing, China, July 2011.

[55] Liang Zhang, Phil Demar, Wenji Wu, and Bockjoo Kim. MDTM: Optimizing data transfer using multicore-aware I/O scheduling. In *Proceedings of the 42nd IEEE Conference on Local Computer Networks*, pages 104–111, Singapore, Singapore, October 2017.