IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 17, 2021, accepted June 22, 2021, date of publication July 5, 2021, date of current version July 13, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3094656

Protocols for Transferring Bulk Data Over Internet: Current Solutions and Future Challenges

KHAWAR KHURSHID[1], IMDAD ULLAH[2], ZAWAR SHAH[3], NAJM HASSAN[4], AND TARIQ AHAMED AHANGER[2]

[1]School of Electrical Engineering and Computer Science (SEECS), National University of Sciences and Technology (NUST), Islamabad 44000, Pakistan
[2]College of Computer Engineering and Sciences, Prince Sattam Bin Abdulaziz University, Al-Kharj 11942, Saudi Arabia
[3]Faculty of Information Technology, Sydney International School of Technology and Commerce, Sydney, NSW 2000, Australia
[4]School of Computer Science and Engineering, University of New South Wales (UNSW), Sydney, NSW 2052, Australia

Corresponding author: Khawar Khurshid (khawar.khurshid@seecs.edu.pk)

ABSTRACT Big Data has gained interests in effectively capturing, storing, analysis and visualisation from wide range of scientific, economic and business communities and is frequently communicated over internet for various purposes among government and enterprise sectors sited at different locations. Several experiments and analyses have shown that currently employed applications and transport protocols in internet are not suitable for transferring such voluminous data because of not addressing requirements of low-access latency. This paper presents issues associated with the basic mechanism of legacy protocols in the context of high speed networks for transferring Big Data e.g. conservative TCP congestion control mechanism may result in minute utilisation of high bandwidth provisioning networks. We present state-of-art alternatives proposed in the literature to solve these problems in high speed networks. We compare several underlying emerging alternatives of TCP, UDP and multi-TCP-streams protocols over a number of comparison criteria e.g. protocol convergence, responsiveness etc., to handle communication of huge data. We note that these protocol alternatives have significant importance over fulfilling requirements of emerging data-intensive applications in high-speed networks. In addition, we discuss open research issues and challenges that can be explored as a source of motivation towards development and deployment of data-intensive applications in emerging networking technologies.

INDEX TERMS TCP variants, UDP variants, high speed networks, long delay networks, congestion control, big data transfer protocols.

I. INTRODUCTION

The evolution of the Internet has made it possible for the deployment of high speed networks with the increasing use of data-intensive and high-performance applications, such as those used in scientific fields e.g. astronomy [1], meteorology [2], social computing [3], bioinformatics [4], [5] and computational biology [6]–[8]. The data from these resource-intensive applications is referred to as 'Big Data' that is typically stored (up to several petabytes i.e. 10^{15} to exabytes 10^{18}) at remote geographic sites and is frequently communicated (up to 100Gbps) [9] by the science community for visualising and scientific analysis where it requires predictable and low-latency access to this data. One such example of storing and communicating this data at remote geographic sites is among large Cloud data centres consisting of large volumes of data of various types, such as real time data, images, and videos captured from different sources etc.

We note that the computer science research community has made several efforts for transporting such large volume of data by proposing applications and transport layer protocols. The transport layer protocols can be broadly categorised into two categories [10] i.e. *connection-oriented*, *reliable* protocols e.g. Transmission Control Protocol (TCP) and *connectionless*, *unreliable* protocols e.g. User Datagram Protocol (UDP). During the design of network applications for enabling high-data rate communication, the designers/ application developers must satisfy requirements of either

The associate editor coordinating the review of this manuscript and approving it for publication was Muhammad Imran Tariq.

of the two transport protocols. These protocols have been widely adopted for end-to-end communication of data over the Internet. We note that UDP is an effective protocol for communicating data over reliable medium e.g. optical fibre, nodes connected via wired networks etc., however, being *unreliable* protocol it is not sensitive to data loss [11]–[15]. The unreliable nature of UDP protocols in multiple streams within the wireless mesh network has also been analysed for transmission failures in [16]. The TCP, on the other hand, achieves *reliable* delivery of data segments by *flow control*, *congestion control*, and *error control*, which ensures that the data segments are received uncorrupted at the received side. However, it significantly under-utilises the network bandwidth over high-speed connections with long *round trip times* (RTT) [17]–[20], when there are multiple TCP streams [21], or in the presences of DCCP[1] protocols during vertical handover [23] for multimedia applications. In addition, the performance of web applications that run on TCP over the passively collected data has also been analysed from an operational network based on wireless infrastructure [24].

In particular, the ineffectiveness of TCP has made it an unsuitable transport layer protocol for high-speed connections with longer RTT networks due to its *slow start* and basic *congestion control* mechanism. For example, considering the basic *slow start* and *congestion control* mechanisms, TCP increases its *congestion window* by one packet and reduces it by half during a loss event. For example, over 10Gbps link, 100ms RTT, and 1500-byte packets, TCP would require 83,333/2 RTTs to increase its window from half utilisation to full utilisation and it would require that there should be no loss event for approximately 1hr. This suggests that TCP can achieve theoretical limit of network's bit error rate of no more than one loss event per 2.6×10^9 packets transfer [17], [18]. Furthermore, following are few difficulties that lead to poor TCP performance in large bandwidth-delay product networks: First, the linear increase in TCP by one packet per RTT is too slow while multiplicative decrease for each loss event is too severe. Second, in order to maintain large average *congestion window* for good throughput, TCP requires to experience extremely low loss rates. Third, since the basic TCP mechanism considers a loss event as a packet loss, hence it is difficult to avoid *congestion window* oscillation since it would always reduce the *congestion window* to half. Furthermore, these *congestion window* oscillations require an accurate estimation of packet loss probability, which is missing in the current deployment of basic TCP.

Several works have modified the basic working mechanisms of these transport layer protocols i.e. UDP and TCP, introducing their different variants for efficient high-speed data transfer communication. These enhancements for enabling high-speed communication can be grouped into three categories: The **TCP variants**, such as, Scalable TCP [19], Binary Increase Congestion Control (BIC)-TCP [25], CUBIC [26], High Speed TCP (HSTCP) [18], FAST-TCP [9], TCP Westwood (TCPW) [27] and eXplicit Control Protocol (XCP) [28]. The **UDP protocols**, such as, Reliable Blast UDP (RB-UDP) [29], UDP based Data Transfer Protocol (UDT) [14], [30], Performance-Adaptive UDP (PA-UDP) [31], in addition to, the two application layer protocols that use UDP as transport layer protocols are Tsunami [32] and Fast and Secure Protocol (FASP). In addition, there are several data transfer application layer protocols that use **multiple simultaneous TCP streams**, such as GridFTP [33], Fast Data Transfer (FDT) and BaBar Copy (BBCP).

In this article we highlight the background theory and implementation of each of above enhancements to transport protocols, in particular, how these protocols improve the *flow control* and *congestion control* mechanisms to enable speedy transfer of data over high-speed connections with long RTT networks. We discuss the under-utilisation of the network's resources in high-bandwidth delay networks due to the slow growth of TCP *congestion window*, *slow-start* that creates *network congestion* and the effect of *congestion avoidance* phase and the harsh penalty over loss events. Furthermore, we present a comprehensive comparison of the performance of different variants of transports protocols that have been evaluated in the literature. We believe that this article is the first that conducts a comprehensive survey of protocols proposed for bulk data transfer in the light of literature work demonstrated so far. In addition, this work will provide detailed discussions and new research directions over emerging TCP alternatives that try to solve problems associated with large *round trip times* due to its *slow-start* mechanism in the context of very high speed networks for high performance networking applications.

We analyse various transport layer protocols for the following performance parameters: the *Congestion control* i.e. used to adjust the data transmission rate as a response to segment loss, RTT unfairness (i.e. identifying multiple flows with different RTTs that consume unfair bandwidth share); the *Inter-protocol fairness* requires that, between two different protocols, one protocol's flow does not receive larger share of the network bandwidth than a comparable flow of another protocol; the *Intra-protocol fairness* i.e. requires that two flows of the same protocol equally share the available network bandwidth, *TCP friendliness* i.e. requires that a new protocol equally shares the network than a comparable TCP flow. We further evaluate these protocols for the following performance metrics: *Throughput* i.e. the rate of successful data delivery over a communication channel, *End-to-end delay* i.e. the time taken for a packet to be transmitted across a network from source to destination, *Packet loss* and *Jitter* i.e. variations in latency in the variability over time of packet latency across a network. Furthermore, we compare various protocols for their capabilities over how are these protocols are *Easy to deploy*, *Stability* i.e. the convergence

[1]Datagram Congestion Control Protocol (DCCP) is a feedback scheme that gathers context for change in transmission state (e.g. during *vertical handover*); implemented via explicit handover notifications and exchange of link parameters using Link Characteristic Information (LCI) option for Mobile IP [22], and timely negotiates and determines transmission parameters for reusing transmission between the end nodes during the handover process.

of a protocol to its equilibrium, *Bandwidth scalability* i.e. maximum utilisation of bandwidth of high-speed networks, *Responsiveness* i.e. ability to complete tasks within given time, and *Protocol convergence* i.e. the ability of a protocol to gather information about the link state in which it operates. In addition, we evaluate these protocols for their commercial usage.

This paper provides the following main contributions:

- To present an outline of research conducted thus far for various transport layer protocols designed for high-speed connections with long RTT
- To investigate issues associated with basic mechanism of TCP specific to its operation in high-speed networks
- To present detailed reviews and classification of various transport layer protocols used for transferring bulk data in high speed networks
- To provide detailed comparison of transport layer protocols with their application to data transfer in high speed networks
- To summarise key findings by various works in literature using different evaluation criteria for various TCP and UDP variants and for protocols with multiple TCP streams
- To discuss open issues in requirements of new variants of transport layer protocols specific to their application for multimedia applications and high-speed wireless access networks

The rest of the paper is organised as follows: Section II presents various mechanisms necessary for reliable end-to-end communication. Section III discusses issues related to fast data transmission using TCP in high-speed networks i.e. issues associated with *slow-start*, *congestion avoidance phase* and the harsh penalty over loss event. Various solutions to these issues using TCP are discussed in Section IV. The application layer protocols that use multiple simultaneous TCP streams for enabling high data transmission are discussed in Section V. Section VI presents the reliable UDP-based data transport protocols that have been proposed as an alternative for enabling reliable data transmission based on UDP protocols. Section VII presents comprehensive analysis of different variants of TCP, protocols that use multiple simultaneous TCP streams and UDP-based reliable protocols in the light of critical investigations by previous research works. Future research directions are discussed in Section VIII. Finally conclusion is presented in Section IX. For simplicity, these sections along with their headings are also presented in Figure 1.

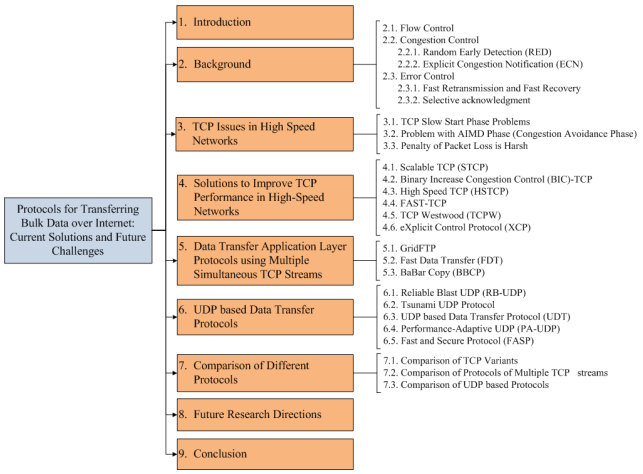


FIGURE 1. Paper organisation into sections.

II. BACKGROUND

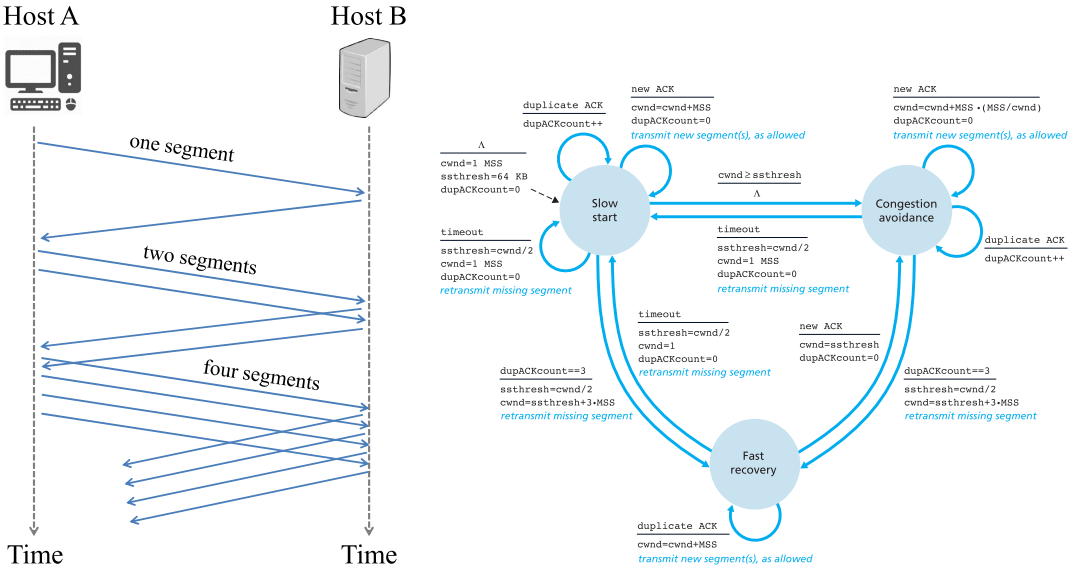
TCP is a connection-oriented transport layer protocol that assures end-to-end reliable communication of data segments with the help of its various mechanisms. These basic mechanisms include the *flow control*, *congestion control*, and *error control* that work closely to facilitate reliable delivery of data segments. TCP is used for non-real delay tolerant applications, such as file transfer, accessing a website etc. Our focus is to review and compare the performance of newer versions of TCP i.e. TCP Reno that is by far most widely deployed, in conjunction with the modifications suggested to basic mechanism of TCP for bulk data transfer in high-bandwidth networks.

A. FLOW CONTROL

To implement *flow control* i.e. to ensure smooth data transmission rate by not overwhelming a slow receiver, TCP uses a sliding window protocol that uses three sliding windows; specifically they are *advertised window*, *congestion window*, and *transmission window*. These window slides are adjusted based on mutual coordination between the sender and receiver for the number of segments sent to the receiver, e.g. receiver notifies the sender in the *advertised window* for the number of segments that receiver can receive in the next transmission cycle. The *advertised window* helps receivers to avoid buffer overflow that the receivers calculate, which is based on the available buffer size to accept subsequent data segments. The sender decides the *congestion window* i.e. maximum number of data segments that the sender can send without causing *congestion* in the network, based on the feedback from the network. Similarly, *transmission window* is the minimum of *advertised window* and *congestion window* in order to respectively avoid receiver buffer overflow and the network *congestion*.

B. CONGESTION CONTROL

The TCP *congestion control* algorithm was proposed in [34] and standardised in RFC 5681. To implement *congestion control* i.e. to adjust the data transmission rate as a response to segment loss, TCP uses *slow-start*, *congestion avoidance* and *fast recovery* mechanisms. During the *slow-start* phase, TCP-sender sets the initial size of *congestion window* (i.e. cwnd) to one maximum segment size (MSS) and exponentially increases this window size upon reception of the corresponding ACKs i.e. *congestion window* extends in 1, 2, 4, 8, ...data segments, also illustrated in Figure 2 (a) [10].



(a) TCP *slow-start* (b) TCP *congestion control* FSM

FIGURE 2. TCP *slow-start* (a) and FSM description of TCP *congestion control* (b) [10].

It undergoes the *congestion avoidance* phase after the `cwnd` reaches the *slow-start* threshold (i.e. `ssthresh`) where it linearly increases the size of the *congestion window*. During the *timeout* it restarts the *slow-start* mechanism where the TCP-sender sets the `ssthresh` to half of the current transmission window size and the *congestion window* to 1MSS. This mechanism is called the Additive Increase and Multiplicative Decrease (AIMD) algorithm [34]. Similarly, it enters the *fast recovery* mode after every duplicate `ACK` is received where the `cwnd` is increased by 1MSS. Furthermore, TCP enters the *congestion avoidance* phase when it receives an `ACK` for the missing segment or it transits to *slow-start* state if a *timeout* occurs. During these transitions, TCP sets the `cwnd` to `ssthresh` for *congestion avoidance* and `ssthresh` to half of `cwnd` for *slow-start* phase. The description of TCP *congestion control* algorithm is illustrated in detail using FSM (Finite State Machine), shown in Figure 2 (b) [10].

Figure 3 illustrates comparative analysis of two versions of TCP's *congestion window* i.e. TCP Tahoe and TCP Reno, in this example, the `ssthres` is 8MSS. In the beginning of transmission, both TCP Tahoe and TCP Reno exponentially increase the *congestion window* during the *slow-start* until they hit the threshold at the forth round of transmission. The *congestion window* is linearly increased until three duplicate `ACKs` are observed by the TCP-sender, during the 8th transmission round (note that the *congestion window* at this stage is 12MSS). At this stage, the `ssthresh` is set to half of `cwnd` i.e. 6MSS. Now, TCP Reno sets the `cwnd` = `6MSS` and grows linearly, alternatively, TCP Tahoe sets the `cwnd` = `1MSS` and starts growing exponentially until it reaches the `ssthresh`, at which point it starts growing linearly.

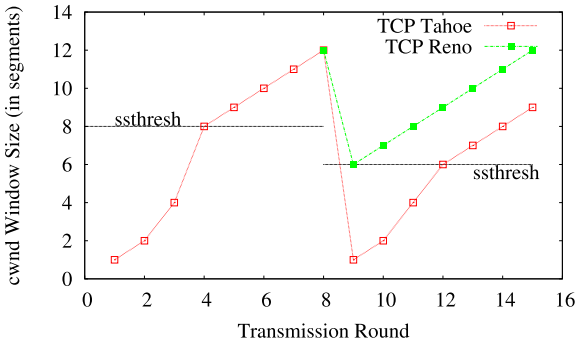


FIGURE 3. Evolution of TCP's *congestion window* (Tahoe and Reno) [10].

The authors in [35] divide the existing *congestion* control algorithms in three groups i.e. *loss-based* algorithms (e.g. TCP Reno [36], NewReno [37], High Speed-TCP [18], Hamilton-TCP [38], Scalable TCP [19], TCP Westwood (TCPW) [27], [39], TCP Westwood+ (TCPW+) [40], TCPW-A [41], and LogWestwood+ [42]), *delay-based* algorithms (such as TIMELY [43] or LoLa [44]), and *hybrid* algorithms (e.g. Bottleneck Bandwidth and Round-trip time (BBR) [45]) and study their interactions. The authors find various fairness issues among the flows with diverse `RTTs` during sharing bottleneck links, more specifically, the *delay-based* and *hybrid* algorithms result in lower performance when competing for flows compared to *loss-based* algorithm. Hence, the selection of *delay-based* and *hybrid* will result in low network performance (i.e. un-fair share of the available bandwidth, longer delays, and packet loss) when majority of the flows rely on the selection of *loss-based congestion* control algorithm. In addition, the authors notice that the *hybrid* algorithms, such as BBR, result in lower queueing delay with flows of higher `RTT`.

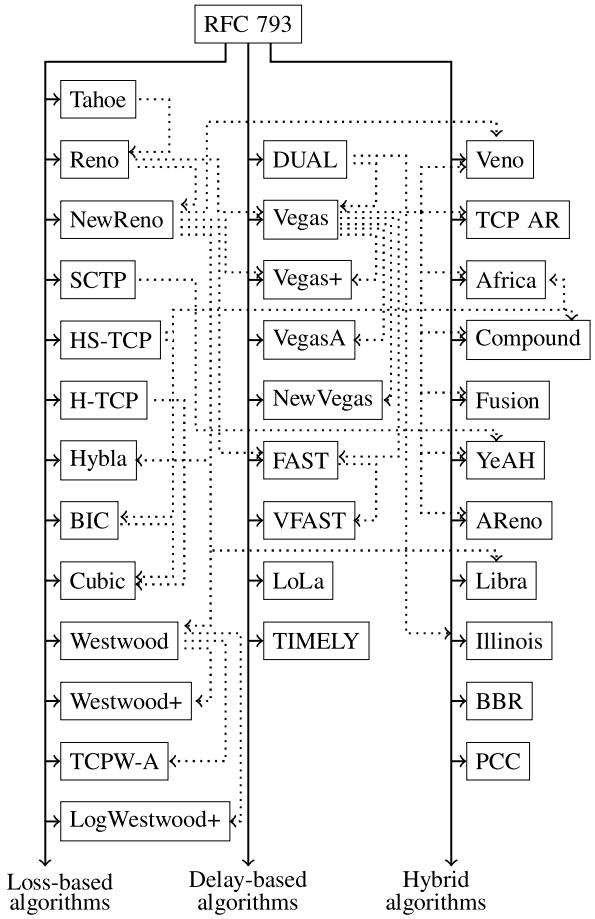


FIGURE 4. Classification of different *congestion control* algorithms. Dotted arrows indicate that one was based on the other [35].

Classification of various *congestion control* algorithms, i.e. *loss-based*, *delay-based*, and *hybrid*, is shown in Figure 4. The conservative nature of majority of the *loss-based* algorithms to loss detection events and packet loss has greatly improved the TCP-fairness issues e.g. Binary Increase Congestion Control (BIC) [25] TCP Hybla [46]. Such protocols guarantee fair share of the bottleneck link for flows with smaller RTTs due to introducing the *congestion window* size function; detailed discussion is given in Section IV-B.

Other examples: Random Early Detection (RED) [47] and Explicit Congestion Notification (ECN) [48] are the two other types of *congestion control* mechanisms:

1) RANDOM EARLY DETECTION (RED)

The RED [47] is a gateway-based *congestion control* mechanism that detects *congestion* at initial stage of data transmission and notifies the TCP sender by computing the gateway's average queue size. The *congestion notification* is caused by either dropping or marking the arriving segments at the gateway (router). The router, with certain probability, sets the mark on the segment or drops it after router's average queue exceeds the preset threshold. The TCP sender in response reduces the transmission rate. This mechanism works as follows: Let the avg and q respectively be the router's average and current queue size, then $avg = (1 - w_q) * avg + w_q * q$, where w_q is the queue weight. The avg queue size is compared with two queue thresholds i.e. minimum $\min_{th}$ and maximum $\max_{th}$ thresholds. A data segment is marked if the avg queue size is greater than the maximum threshold; otherwise, no packets are marked. Note that the data segments are dropped only if the TCP-sender is not cooperative. This process ensures that the average queue size does not exceed the maximum threshold. Furthermore, if the avg queue size is between the $\min_{th}$ and $\max_{th}$ thresholds then each arriving segment is marked with probability p_a. The data segment-marking probability p_b is calculated as $p_b \leftarrow \left(\max_p (avg - \min_{th}) / (\max_{th} - \min_{th})\right)$ when the avg varies between the $\min_{th}$ and $\max_{th}$, where $\max_p$ is the maximum value for p_b. The final segment-marking probability p_a is calculated as $p_a \leftarrow p_b / (1 - count \cdot p_b)$, $count$ is the number of segments arrived since the last segment marking/dropping. Consequently, if $p_a = 1$ then each newly arriving packets are dropped. Hence, RED prevents *congestion* at the gateway and improves fairness by controlling the average queue size before the queue overflows. This work is further extended by a number of researchers [49]–[52] to improve the basic functionality of RED.

2) EXPLICIT CONGESTION NOTIFICATION (ECN)

The Active Queue Management (AQM) mechanisms, such as RED [47], detect *congestion* before the queue overflows at the router, thus avoiding the global synchronisation problem and heavy network *congestion*. However, the only choice with these queue management systems is to drop the data segments that might cause higher end-to-end delay and bad user experience. The ECN [48] presents an alternative by allowing both the TCP-senders and TCP-receivers of the congested network participate in avoiding network *congestion*; requiring changes both at the IP and TCP headers. To enable ECN, it uses ECN field with two bits in the IP layer header. The ECN-enabled router can send *congestion* indication to the end systems. Similarly, it introduces two flags in the TCP header that both the TCP-sender and TCP-receiver negotiate to enable *congestion* indications via ECN, during the connection establishment. Hence, the TCP-receiver can also inform the TCP-sender of the network *congestion* if it has received *congestion notifications* from the intermediate routers.

C. ERROR CONTROL

To implement *error control* i.e. enabling reliable communication over unreliable channel, TCP uses acknowledgement mechanism implemented via sequence number to achieve reliable data delivery. On receiving the ACK segment, the sender confirms that the previously sent segments have been successfully received by the receiver. For example, if segments up to $n - 1$ have reached the receiver then the ACK would indicate the successful arrival of n sequenced data segments. TCP uses duplicate ACK if an *out-of-order* segment

arrives at the receiver. The *out-of-order* segment is considered as segment loss; however, the sender retransmits the same segment after it receives three duplicate ACKs. In addition, the sender assumes segment loss if ACK for a particular segment is not received with the *Retransmission Timeout* (RTO) interval. The TCP-sender dynamically calculates the RTO as the estimated *Round Trip Time* (RTT).

There are several representative works to implement error control, such as Fast Transmissions and Fast Recovery [53], and Selective Acknowledgements [54].

1) FAST RETRANSMISSION AND FAST RECOVERY

This scheme [53] offers a *fast retransmission* of the lost segment after the TCP-sender receives three duplicate ACKs that allows TCP to avoid long *timeouts*. During the *fast retransmission*, it sets the ssthresh to half of the cwnd and the cwnd is set to ssthresh with additional three segments. Furthermore, it undergoes *fast recovery* when an ACK is received approximately one RTT after the missing segment is retransmitted. During this phase, it sets the cwnd to ssthresh instead of setting up the cwnd to one segment. The TCP-sender then undergoes the *congestion avoidance* phase. Note that during the *fast retransmission* and *fast recovery* only one segment can be recovered, however, it may require that RTT expires before retransmission for additional segment losses in the same window.

2) SELECTIVE ACKNOWLEDGMENT

The *Fast Retransmission and Fast Recovery* [53] might experience low performance due to its limitation of recovering multiple segments that are lost in one window, hence, to overcome this limitation the Selective Acknowledgement (SACK) [54] has been proposed that recommends the use of SACK option as an addition to the basic TCP. The use of SACK option is decided between the TCP-sender and TCP-receiver during the connection establishment process. The SACK option contains four or three blocks that specify the adjacent segments of received data. In multiple segments loss occurrences in a particular window, the SACK-enabled TCP sender can evaluate and retransmit the lost segments with the information provided in SACK blocks.

In the following section, we present various issues specifically associated with the basic mechanism of TCP in high-speed networks that restrains it to achieve better performance in terms of high throughput, low loss rate, and low end-to-end delay.

III. TCP ISSUES IN HIGH SPEED NETWORKS

The *congestion control* mechanism of TCP creates severe problems when transferring huge data in high-speed data communicating environments. Some of these problems are discussed below.

A. TCP SLOW START PHASE PROBLEMS

Following are the two problems associated with the TCP *slow-start*:

a: INITIAL VALUE OF SLOW START IS TOO SMALL

In the beginning of every connection, TCP utilises *slow-start* phase in which the initial congestion window size of 1MSS is doubled for every ACK received. This small initial value of congestion window results in TCP to slowly probe for more throughput and increases time for TCP to utilise the large bandwidth that is available to it [10]. This works fine for controlling the congestion in regular networks, however in high-speed data communicating environments, it slows down the connection and is inefficient for transferring huge data at high speed.

b: SLOW START CREATES NETWORK CONGESTION AND RESULTS IN PACKET LOSS

At the start of a new TCP connection, the sender does not know the proper congestion window for the path. It starts with 1MSS (as mentioned above) and exponentially increases the window size. It keeps on doubling the window size until the congestion window reaches a threshold ssthresh, at which point TCP converts to a linear increase of the congestion window (i.e. congestion avoidance phase), or when packet loss occurs. The performance of Slow Start is sensitive to the initial value of ssthresh. If ssthresh is too low, TCP may need a very long time to reach the proper window size, while a high ssthresh can cause significant packet losses, resulting in a timeout that results in very low effective throughput [55]–[57].

B. PROBLEM WITH AIMD PHASE (Congestion AVOIDANCE PHASE)

The AIMD phase is inefficient to sustain data transfer between cloud data centres. It is considerably slow in linearly increasing the congestion window size compared to the exponentially increasing Slow Start. This was implemented to tame the aggressive nature of traditional TCP when congestion occurs [58] however, this slow increase to high window size is not suitable for fast bulk data transfer in a cloud computing.

C. PENALTY OF PACKET LOSS IS HARSH

Another issue with TCP is that it reduces congestion window size by half or resets the congestion window to 1MSS after detecting a packet loss (depending on the packet loss event). This reduction is required for congestion control however, it results in small window size which reduces the effective throughput and is inefficient for fast bulk data transfer [58].

We note that these various issues with basic TCP are among various motivating factors that researchers have proposed various variants of transport layer protocols. We argue that these issues should be addressed during the design of protocols used for transporting data-intensive applications in high-speed networks, in order to achieve better performance in terms of *throughput*, *end-to-end delay* etc.

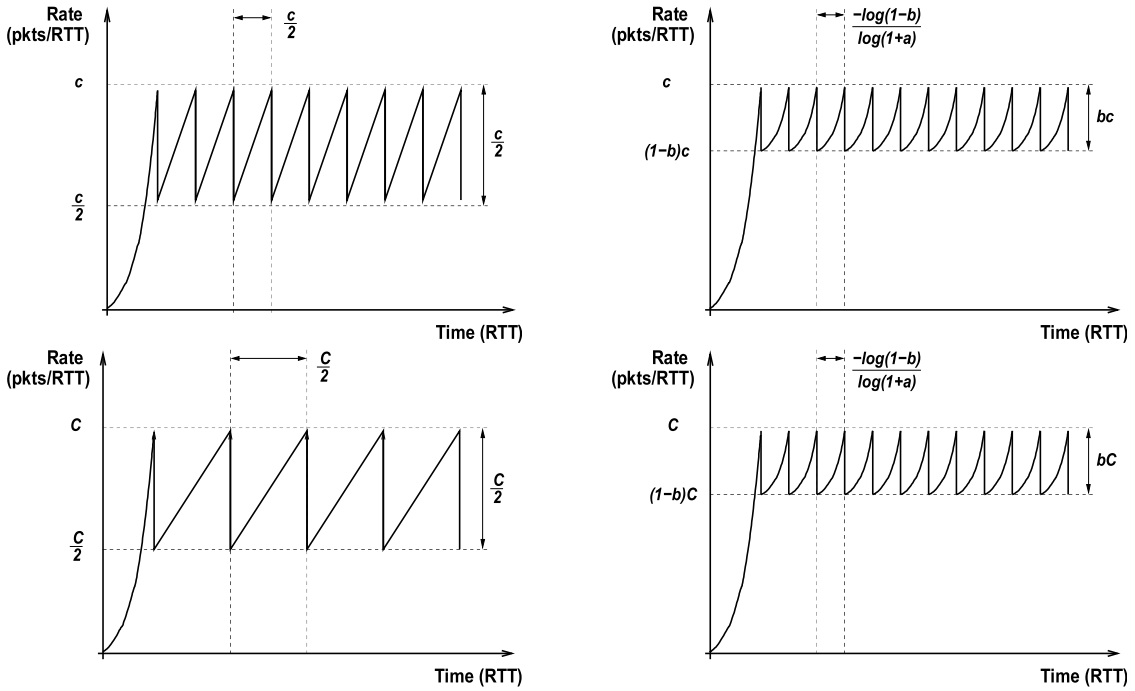


FIGURE 5. Traditional TCP scaling properties (left) Vs. Scalable TCP scaling properties [19].

IV. SOLUTIONS TO IMPROVE TCP PERFORMANCE IN HIGH-SPEED NETWORKS

We now classify various solutions proposed in literature for enhanced communication of voluminous data over high-speed communicating network i.e. *TCP variants*, *multiple simultaneous TCP streams* and *UDP variants*. We first start with various variants enhancements to basic TCP.

A. SCALABLE TCP (STCP)

The STCP is a sender side modification to the *congestion avoidance* phase of the TCP *congestion control* mechanism [19], [59], [60]. It uses Multiplicative Increase and Multiplicative Decrease (MIMD) algorithm instead of Additive Increase and Multiplicative Decrease (AIMD) used by TCP. The increase in *congestion window* can be calculated as $cwnd = cwnd + a * cwnd$ where $a = 0.01$. Similarly the decrease in *congestion window* is given as $cwnd = cwnd - b * cwnd$ where $b = 0.01$. Figure 5 illustrates and compares the *congestion window* of a single connection using traditional TCP and STCP over a link of capacity c [19]. Note that the recovery time form the *packet loss* event for STCP connection is proportional to the connection's window and `RTT`, which allows the STCP to perform better than the traditional TCP in high speed networks.

During the event of a segment loss, the `cwnd` is reduced to half of its previous value instead of reducing it to 1MSS. This results in higher throughput compared to basic TCP. It has been shown, using extensive experimentations, that STCP outperforms the basic TCP during the events of segment loss. For example, STCP, during the segment loss event, over a 10Gbps link with 200ms delay and the packet size of 1500bytes [61] takes only 2.7sec to recover the segment loss. Alternatively, basic TCP experiences extensive delay of 4hrs 43min to recover the segment loss and resumes communication.

In addition, we note that STCP, due to multiplicative increase, effectively performs with the higher sizes of *congestion windows* i.e. `cwnd` of more than 100MSS, since $a = 1/100$. Hence, although it has solved the issue of linearly increasing the *congestion window* during the *congestion avoidance* phase of AIMD, it does not solve the *slow start* issue associated with basic TCP, as discussed in Section III-A.

B. BINARY INCREASE CONGESTION CONTROL (BIC)-TCP

TCP, due to its recovery from the *congestion* events, might not utilise the full bandwidth of a high-speed network. The BIC-TCP [25] takes into consideration two properties: *TCP friendliness* i.e. avoids taking too much bandwidth from TCP flows, and *bandwidth scalability* i.e. maximum utilisation of bandwidth of high-speed networks. This BIC-TCP prevents the `RTT` unfairness by identifying multiple flows with different `RTTs` that consume unfair bandwidth share. It introduces a new *congestion control* mechanism that consists of two schemes i.e. *additive increase* and *binary search increase*. This mechanism provides `RTT` fairness, scalability and TCP-friendliness among multiple TCP flows.

The result of the *binary search increase* is the true/false feedback to implement the *congestion control*, which determines whether the current sending rate is according to the

network capacity. It calculates the minimum window $w_{\min}$ (i.e. current sending rate) as the window size before any packet loss occurs, while the maximum window $w_{\max}$ is the window size during when the packet loss occurs. The target window $w_{t\,\arg\,et}$, after a *congestion* event, is calculated using the binary search as the midpoint of $w_{\min}$ and $w_{\max}$ i.e.

$$w_{t\,\arg\,et} = {}^{(w_{\min} + w_{\max})}/_{2} \tag{1}$$

The BIC-TCP promises faster convergence and RTT fairness, in the event of network *congestion*, by combing the *binary search increase* with the *additive increase* strategy. The combination of these two strategies works in the following conditions: If the distance of w_{target} from $w_{\min}$ is too large then directly setting up the $w_{t\,\arg\,et}$ to the midpoint might still have no effect over controlling the *congestion* and the network might still experience *congestion*. Hence, it introduces another predetermined increment, called the 'maximum increment' $S_{\max}$, and sets the 'current window' to $S_{\max}$. Thus, subsequent to a large window reduction, this strategy first increases the window linearly and then increases logarithmically.

It has been reported [62] that BIC-TCP provides good performance in terms of throughput than the basic TCP. Furthermore, it solves the *slow start* issue with the high-speed network by setting up the current window to $S_{\max}$ instead of 1MSS, as is the standard operation in the basic TCP.

Another study, CUBIC-TCP [26], provides an enhancement to BIC's window growth function, which is too aggressive for TCP in the current high speed networking environment and because of its complexity due to several different phases of window control. Hence, it makes BIC window control simpler and enhances its TCP friendliness and RTT fairness by use of a function evaluated for time elapsed since the most recent occurrence of a loss event. It determines the *congestion window* as:

$$w_{cubic} = C(t - K)^3 + w_{\max} \tag{2}$$

Here C is a scaling factor, t is time elapsed since the last loss event, $w_{\max}$ is the window size at t, and $K = \sqrt[3]{{}^{w_{\max}\beta}/_{C}}$, here β is a decrease factor applied for window reduction at t.

Figure 6 shows the window growth function of BIC (upper) and CUBIC (lower). It can be observed in Figure 6 (lower part) that the growth function grows fast until it reaches $w_{\max}$ where it slows down its growth and the increment approaches almost zero. At this stage, CUBIC starts probing for additional bandwidth, starts growing window and accelerates its growth and moves away from $w_{\max}$. Alternatively, in Figure 6 (upper part), BIC carries out a binary search over $w_{\min}$ and $w_{\max}$ and calculates midpoint, which could be too much within an RTT and would affect other TCP flows.

C. HIGH SPEED TCP (HSTCP)

HSTCP [18] presents optimisation over the basic TCP for high data rate networks. It maintains two *congestion windows*

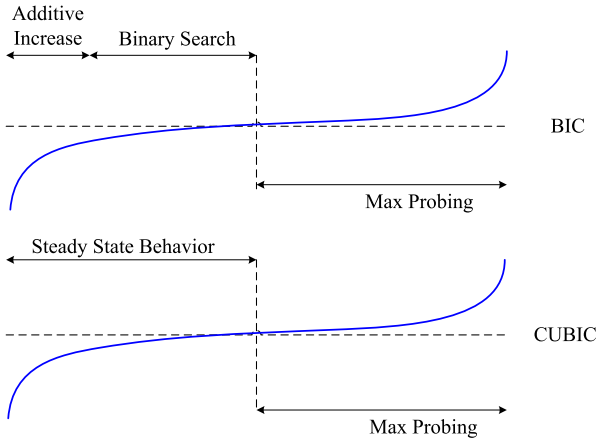


FIGURE 6. The window growth function of BIC and CUBIC [26].

i.e. minimum *congestion window* L_{win} and maximum *congestion windows* H_{win}. It dynamically calculates the new cwnd during each RTT. When the *congestion window* is less than the L_{win} then it calculates the response function W for the new *congestion windows* as:

$$W = \left({}^{p}/_{p_{low}}\right)^{S} * L_{win} \tag{3}$$

Here p is the packet drop rate, p_{low} is the drop rate corresponding to L_{win} and S is calculated as:

$$S = {}^{(\log(H_{win}) - \log(L_{win}))}/_{(\log(p_{high}) - \log(p_{low}))} \tag{4}$$

We note that the HSTCP (i.e. High Speed TCP) is proposed for large *congestion windows*, since the response function W can only work when the *congestion window* increases to certain high value. It has also been evident [18], [63] that HSTCP works similar to the basic TCP in a high packet loss networks. Furthermore, it does not utilise the full bandwidth of high speed networks during its *slow start* mode.

D. FAST-TCP

FAST-TCP [9] introduces another criterion for reporting the *congestion* event i.e. based on queuing delay instead of packet loss. The *congestion control* mechanism of FAST TCP consists of the following four components that work independently and can be upgraded asynchronously: The *data control* component that determines packets in the queue to be transmitted, the *window control* that determines the number of packets to be transmitted, the *burstiness control* that determines the time at which these packets are scheduled for sending to the receiver and finally, the *estimation* component that provides information for carrying out above decisions. It calculates the number of packets inside the queue by measuring the difference between the current RTT and the exponential weighted average RTT.

Experiments [9], [20] have shown that FAST-TCP achieves higher throughput compared to basic TCP. The limitation, however, with the FAST-TCP is due to re-routing. Since FAST-TCP uses an estimated RTT to adjust its window

TABLE 1. Internet throughput measurements [27].

Destination	Italy		Taiwan		Brazil	
RTT	170ms		250ms		450ms	
Protocol	TCPW	Reno	TCPW	Reno	TCPW	Reno
Throughput (KB/s)	78.66	73.93	167.38	152	22.16	15.4

size, hence during the re-routing, it is very important for a FAST-TCP connection to be able to have an accurate estimation of the RTT. We note that re-routing may change the RTT estimation and the new rate may result in decreased throughput and longer delays. Such issues of re-routing is also presented in the other delay based TCP variants e.g. TCP Vegas [64].

E. TCP WESTWOOD (TCPW)

TCPW [27] is an improved version of TCP Reno that is a sender-side modification of the TCP *congestion control* mechanism with improved performance in high loss environment. It constantly evaluates the sender TCP bandwidth consumption by monitoring the ACKs reception rate. It then uses this estimation in calculating the cwnd and the *slow start* threshold in the *congestion* events i.e. either after the occurrence of three duplicate ACKs or a *timeout*, that helps in determining the data to be delivered to the destination; the authors call this mechanism the *fast recovery*. The experimental evaluations discussed in [27] have shown that TCPW can effectively coexist with TCP Reno and provides better TCP *fairness* and *friendliness*. Such an example of improved performance of gaining higher throughput of TCPW over TCP Reno in different geographical regions is shown in Table 1.

F. EXPLICIT CONTROL PROTOCOL (XCP)

XCP [28] is an enhancement to TCP's basic *congestion control* mechanism in the environment of high bandwidth-delay e.g. fulfilling communication requirements of high bandwidth optical links, and large delay satellite links. It outperforms TCP and further remains fair and stable as the bandwidth-delay product increases. XCP generalises the Explicit *Congestion Notification* proposal (ECN) [65], which uses one bit *congestion* indication for informing routers about the *congestion*, conversely, the XCP-enabled routers inform the senders about the degree of *congestion* at the bottleneck.

The XCP protocol works as follows: The XCP-sender maintains and sends the cwnd and RTT to the XCP-routers via *congestion header* in every packet. The *congestion header* maintains information about the current *congestion window* H_{cwnd}, the sender's current estimated RTT H_{RTT} and feedback $H_{feedback}$ that takes positive or negative value and is initialised by the XCP-sender. The H_{cwnd} and H_{RTT}, are filled by the senders and are never modified during the communication, the $H_{feedback}$, on the other hand, can be modified by the router. The $H_{feedback}$ is calculated based on the H_{cwnd} and H_{RTT} so that the system converges to achieve fairness. Hence, the packet will contain the $H_{feedback}$ from the bottleneck router along the path, which is then returned to the XCP-sender. The cwnd is increased or decreased and is calculated based on the $H_{feedback}$ i.e.

$$cwnd = \max\left(cwnd + H_{feedback}, s\right) \tag{5}$$

Here s is the packet size. Furthermore, the XCP-receiver has the same functionality as TCP-receiver, except that it attaches the *congestion header* to the ACK of a received data packet.

Table 2 summarises various aspects of six different TCP variants discussed above. In particular, this table highlights the protocols capabilities of providing *TCP friendliness*, *intra-protocol fairness* and their usage. It also highlights issues associated with each protocol that may lead to inefficient performance resulting in lower throughput, high end-to-end delay and severe packet loss.

V. DATA TRANSFER APPLICATION LAYER PROTOCOLS USING MULTIPLE SIMULTANEOUS TCP STREAMS

Intensive data transferring applications e.g. distributed scientific and engineering applications, require access to and transfers of large amounts of data (in terabytes or petabytes) between storage systems that are geographically distributed for processing, such as analysis, visualisation etc. There are several application layer protocols proposed, which utilise *multiple simultaneous TCP streams*, for overcoming the basic problem of TCP (i.e. due to *slow-start* mechanism) with transferring huge amount of data e.g. GridFTP [33], FDT [68], BBCP [69] etc. In this section, we discuss the basic mechanism of these data transfer application layer protocols.

A. GridFTP

GridFTP is an open-source software implementation, which provides extensions to FTP for a grid computing environment. GridFTP supports automatic negotiation of TCP buffer sizes both for large files and large sets of small files. GridFTP achieves better use of bandwidth by using multiple simultaneous TCP streams. It helps to download either pieces of files simultaneously from multiple sources or even in separate parallel streams from the same source. It provides other enhancements in FTP, such as, data striping and TCP socket buffer optimisation. To enable reliability, the GridFTP server automatically sends restart markers (checkpoints) to the client. If the transfer has a fault, the client may restart the transfer and provide the markers received. The server will restart the transfer, picking up where it left off based on the markers [33], [70]. GridFTP has shown to provide better throughput than FTP [71]. Experiments carried out in [71] show that GridFTP provides much higher throughput than FTP. Similarly, results in [72] show higher throughput is obtained by GridFTP when TCP Cubic is used at the transport layer. GridFTP uses TCP at the transport layer and suffers from the problems in TCP mentioned in the section above. However, it has shown reasonable improvement with the use of other TCP variants like TCP Cubic [72]. Also, UDT (discussed later), a UDP based protocol is used with GridFTP to improve the transmission rate [73], [74].

TABLE 2. Comparison of different TCP variants: STCP, BIC-TCP, CUBIC, HSTCP, FAST, TCPW and XCP for various aspects.

	TCP Friendliness	Intra Protocol Fairness	Issues	Usage
STCP	Yes	Yes	Only effective with `cwnd` $\geq$ 100MSS	Open Source, No Commercial Use
BIC-TCP	Yes	Yes	Growth function is too aggressive for TCP and high complexity due to several different phases of *congestion window* control	Used by default in Linux kernels 2.6.8
CUBIC	Yes	Yes	CUBIC is optimised for high speed networks with high latency	Used by default in Linux kernels from 2.6.19 to 3.1
HSTCP	Yes	Yes	HSTCP has the same slow start/timeout behavior as basic TCP	No Commercial Use
FAST	No	Yes	Re-routing may result in decreased throughput and longer delays due to `RTT` estimation	Commercialised by FastSoft. FastSoft was acquired by Akamai Technologies in 2012 [66].
TCPW	Yes	Yes	TCPW bandwidth estimation algorithm does not work in the presence of reverse traffic due to ACK compression	Open Source, Implemented in the Linux kernel
XCP	Yes	Yes	XCP requires a specific knowledge of the link speed *a priori*	Integrated into ns-2.28 at USC/ISI [67].

B. FAST DATA TRANSFER (FDT)

FDT is based on an asynchronous, flexible multi-threaded system and is written in Java programming language. It transfers data in parallel with multiple simultaneous TCP streams. It has the capability to resume file transfer sessions without any loss, if needed. The number of streams that can be used by FDT are 4 by default, however, this number with other protocols e.g., BBCP (i.e. BaBar Copy) can be modified by the user. FDT can be used to transfer a list of file continuously without the network transfer restarting between files [68]. FDT uses TCP and suffers from the basic problems in TCP, as mentioned in Section III.

C. BaBar COPY (BBCP)

BBCP is a Peer to Peer (P2P) network application that is capable of high speed data transmission by breaking up data transfer into multiple simultaneously transferring TCP streams [69], [75]. BBCP provides the users with the ability to tune different parameters e.g. *congestion window* size, number of streams etc. BBCP uses four TCP streams by default, which can be increased if needed. BBCP also keeps track of copy operations so that an operation can be restarted from the point of failure at a later time. This helps in minimising the amount of network traffic in the event of a copy failure [69]. BBCP transfers data much faster than single-streaming protocols [69]. However, it suffers from the issues associated with TCP. It may also be blocked by the firewalls as it is a P2P application.

We note that the main objective of proposing above protocols is to provide a reliable and high performance file transfer consisting of very large files. These protocols are extensively used in large science projects, such as Large Hadron Collider,[2] for transferring data across geographically dispersed locations.

[2]http://home.cern/topics/large-hadron-collider

VI. UDP BASED DATA TRANSFER PROTOCOLS

We now discuss mechanisms of various *UDP variants* proposed for bulk data transfer along with various other works that evaluate the performance of these various UDP variants.

A. RELIABLE BLAST UDP (RB-UDP)

The RB-UDP [29] is an aggressive bulk data transfer scheme that uses both the UDP to transfer bulk data and the TCP for transferring control information. The motivation behind this protocol is to keep use of majority of the available bandwidth over the entire bulk data transfer and is to avoid extra overhead generated by TCP for acknowledging each received packets. Hence, the aggregated acknowledgements are transmitted at the end of a transmission phase. The sender RB-UDP sends the entire payload at the rate specified by the receiver RB-UDP using UDP. Furthermore, the sender RB-UDP sends a `DONE` message in order to indicate the end of transmission. Alternatively, the receiver RB-UDP reports the missing packets by sending an acknowledgment, where the sender RB-UDP re-sends the missing packets.

The RB-UDP predicts the achievable bandwidth as:

$$B_{achievable} = {S_{total}}/{T_{total}} \tag{6}$$

Here S_{total} is the total payload size and T_{total} is the predicted send time. The T_{total} can be estimated as:

$$\begin{aligned} T_{total} = {} & \left(T_{prop} + T_{udpSend_0}\right) \\ & + \left(\sum_{i=1}^{N_{resend}} \left(T_{prop} + T_{udpSend_i}\right)\right) \\ & + \left((N_{resend} + 1) * \left(T_{ack} + T_{prop}\right)\right) \end{aligned} \tag{7}$$

The T_{prop} is the propagation delay, $T_{udpSend_i}$ is the time to send an i^{th} iteration, N_{resend} number of times to resend and T_{ack} is time to acknowledge a blast. Furthermore, it evaluates

the best achievable performance to:

$$\frac{B_{best}}{B_{send}} = 1 \Big/ 1 + \frac{RTT * B_{send}}{S_{total}} \tag{8}$$

This shows that the throughput can be maximised by minimising the $1 + \frac{RTT * B_{send}}{S_{total}}$ ratio. Furthermore, the number of retransmitted packets, given the loss rate L, can be evaluated as [29]:

$$N_{resend} = \left\lfloor \log_L \left(S_{packet} / S_{total} \right) \right\rfloor \tag{9}$$

Performance analysis of RB-UDP carried out in [15] shows that it can achieve higher throughput than the basic TCP. Furthermore, the experimental analysis in [29] shows that RB-UDP can achieve a high link utilisation of 70% when a large file is transferred on a 1Gbps link between Chicago, US and Amsterdam, Netherland. However, we note that RB-UDP has no *congestion control* mechanism in order not to overwhelm the receiver RB-UDP, hence, the receiver machine has to be powerful enough in order to accept the bulk data transfer sent by the sender RB-UDP. Furthermore, it is important that RB-UDP has to effectively calculate the sending rate so that the sending rate should not be higher than the available bandwidth of the bottleneck link in the end-to-end path. Similarly, the sending RB-UDP has to keep all the sending packets in its memory if it requires to retransmit the missing packets, which consequently requires bigger memory buffers. The RB-UDP is hence suitable for use in private or dedicated networks.

B. TSUNAMI UDP PROTOCOL

Tsunami [32] is an application layer protocol that uses UDP for transferring data blocks while it transfers the control information via TCP. It is a reliable transfer protocol; however, it evaluates the transmission rate using inter-packet delay rather than a sliding window mechanism as a basic procedure e.g. being performed in the basic TCP. Tsunami is mainly implemented on both client and server sides. The client maintains two threads i.e. the *network thread* that communicates data and control information with the server, maintains retransmission queue and places the fetched blocks from server for storage into a buffer, alternatively, the *disk thread* moves the fetched blocks from buffer to the storage disk. The server maintains only one thread i.e. *service thread* that is responsible for fetching data blocks from the disk and sending those data blocks to the client.

At the start of communication, both the client and server carry out authentication by evaluating the `MD5` checksum over random data that has already been `XOR`ed with a shared secret. Both the client and server negotiate various parameters of UDP buffer size, tolerated error rate, sending rate etc., before transmitting the actual data packets. A study [76] suggests that, for a data rate of 1000Mbps, Tsunami UDP is tolerable to the loss rate of up to 7.5%. The client sends the request for retrieving desired data blocks from the server. Afterwards, if the desired block is available, the client sends the index of the desired block, the file transfer rate, error threshold and scaling factor for inter-packet delay. Similarly, the server responds with the file size, data block size, total number of blocks and a timestamp. The client then sends the UDP port to the server and it starts downloading file form the server. Tsunami is used by Amazon Web Services (AWS) that offers reliable, scalable and inexpensive cloud computing services.

Several studies compare the performance of Tsunami [32] protocols for efficient data transfer with other transport layer protocols, such as TCP, Secure Copy Protocol (SCP) etc. For instance, [77] compares the Tsunami UDP and SCP[3] for transferring bulk data between two Amazon EC2 instances located in two different regions (i.e. in USA-East and Singapore). This analysis shows that Tsunami UDP can transfer 50GB of file in 19min, 33sec while SCP takes a total of 1hr, 50min. Another study [78] investigates that Tsunami UDP can achieve a throughput of 651Mbps when data transferring takes place between two AWS EC2 instances located in Tokyo, Japan and Virginia, USA. Furthermore, [79] concludes that Tsunami UDP is suitable for transferring files of moderate to large databases e.g. database of sizes 100GB to 5TB.

C. UDP BASED DATA TRANSFER PROTOCOL (UDT)

UDT [14], [30] is a connection-oriented duplex protocol built on the top of UDP and is specifically designed for high-speed wide area optical networks. It has its own reliability and *congestion control* mechanism for achieving high bandwidth utilisation. The UDT architecture is the same on both the sender and receiver devices i.e. it implements two modules of `UDT-sender` and `UDT-receiver` at both sending and receiving entities. The data is sent from `UDT-sender` to `UDT-sender` while the control information is passed between `UDT-receiver` and `UDT-receiver` of the two user devices. It implements a fixed interval timer-based selective acknowledgment for faster data transfer rather than acknowledgments for every packet, which results in extra bandwidth consumptions and transmission delays. These selective `ACKs` are sent if there are new continuously received data packets. Hence, the control packets consume less bandwidth in the faster data transfer and more bandwidth if there are less number of data packets, in which case, it acts like basic TCP due to frequently sending `ACKs`.

Furthermore, it implements the DAIMD (Additive Increase Multiplicative Decrease (AIMD) with decreasing increases) as the *congestion control* algorithm, as described in [14]. It is called DAIMD because the additive parameter decreases as the data sending rate is increased. For example, it increases the data rate x by a factor of $\alpha\ (x)$, which is a non-increasing factor and it approaches 0 with the increase of sending data rate i.e. $\lim_{x\to\infty} \alpha\ (x) = 0$. This increase is added to the new

[3]Secure Copy or SCP uses TCP as a transport layer protocol and is used for securely transferring data files between a local host and a remote host or between two remote hosts.

data rate if the sender receives a positive feedback from the receiver (e.g. no loss).

UDT has several applications, for instance, it is widely used in Grid Computing e.g. GridFTP [33] uses UDT for data transfer [73], [74] that is an extension of the File Transfer Protocol (FTP) for grid computing. UDT is an open source and its implementation can be found on SourceForge.[4] Literature works reveal enhanced performance of UDT compared to other protocols e.g. [80] carries out comparative analysis of UDT and TCP with different link capacities ranging from 100Mbps to 1Gbps in various loss rates and link delays. For instance, with zero loss rate and 50ms delay, UDT can maintain bandwidth consumption over 60% and 90% bandwidth for 500Mbps and 1Gbps links, respectively, while TCP maintains link consumption of less than 10Mbps for both link capacities of 500Mbps and 1Gbps. Similarly, experiments in [30] reveal that UDT can reach up to 950Mbps over 1Gbps link with 110ms delay from Chicago to Amsterdam. Other studies [14], [15], [81] show that UDT is TCP friendly and demonstrate the intra protocol fairness. However, in high packet loss environment, UDT performs as poorly as TCP [82], [83]. This is due to the fact that UDT also uses the *congestion window* mechanism, as TCP, for evaluating the transmission rate. Furthermore, we note that UDT uses packet pair technique for capacity estimation, however, it may result in underestimation of capacity due to the cross traffic present in WAN links, which consequently might result in low throughput.

D. PERFORMANCE-ADAPTIVE UDP (PA-UDP)

PA-UDP [31] is a high-performance protocol for high speed and high latency networks with reduced configurations required at the user level. It is an open source and is used by VMware vCloud Connector 2.5.[5] The vCloud Connector is an enterprise product that allows to connect multiple clouds, both internal and external, in a single user interface. The single user interface oversees multiple public and private clouds and for transferring cloud content from one cloud to another. The PA-UDP maximises the performance of various entities of a communication system by taking into account the CPU latency in accessing data from the disk, the effect of disk throughput, the receiving application's buffer, receiver's Kernel buffer and the sending application's data sending rate since these entities may restrain the overall capacity of the network.

The PA-UDP sender sends the initial data rate by three-way handshake. The PA-UDP receiver resets the sending rate by periodically calculating packet losses, receiving rate, disk processing rate and buffer size and then sends feedback with new sending rate to the sender. The sender then adjusts the sending rate sent by the receiver through change in the inter-packet delay. In particular, let $r\,(recv)$ be the receiver data rate in bits/sec at which it can receive packets without any packet loss and $r\,(disk)$ be the rate at which data is read from the disk, then PA-DUP tries to keep the ratio of the two rates $\alpha = \frac{r(recv)}{r(disk)}$ constant. Hence, the disk's and the network's activity remains constant and the sender does not overwhelm a slow receiver.

The experimental analysis given in [15], [84] shows that PA-UDP achieves high throughput and high channel utilisation by taking into account the system hardware i.e. buffer size, disk processing rate, for transferring bulk data. However, we note that PA-UPD handles only one client at a particular time instance while it puts all the rest data transferring nodes into `Silent` mode in a wait queue. Hence, PA-UDP is suitable for a private network where individual users can achieve higher data rates. Furthermore, PA-UDP is vulnerable to high packet losses since it uses the packet loss to calculate the sending rate, as is done by TCP, which may result in reduced throughput. It is also evident [85] that TCP achieves less throughput in the presence of PA-DUP, however, it can achieve good *intra-protocol fairness* [15], [85].

E. FAST AND SECURE PROTOCOL (FASP)

FASP is an application layer protocol acquired by Aspera,[6,7]. It uses UDP at the transport layer and provides reliable transport for applications that do not require 'byte stream in-order' delivery. FASP, unlike TCP, utilises an adaptive rate control that uses packet delay i.e. `RTT`, as a *congestion* signal. The packet delay for *congestion control* enables FASP to decouple reliability and *congestion control* and it retransmits lost packets without impacting the transmission rate. It introduces a separate sending queue for transmitting the incoming packets. FASP sends probe packets into the network to obtain measure of queuing delay along the path. FASP reduces its transmission rate, proportional to the difference between targeted and current queuing delay, on detecting higher queuing delay. Similarly, it increases the transmission rate, proportional to the targeted queuing delay, when queuing delay is reduced that indicates less *congestion* [86]. Using queuing delay as *congestion* indicator can cause issues with rerouting of path (explained above). However, Aspera claims their algorithm estimates `RTT` accurately. Moreover, FASP provides an automatic checksum and bandwidth dialling capabilities in order to ensure reliable completion of data transfer with minimal interruptions. More details on their algorithm are not publicly available, however, Aspera states that this algorithm follows Van Jacobson theory explained in [34].

There are several works that utilises FASP for bulk data transfer. For example, the results in [86] show that FASP takes approximately 10sec to complete transferring the file of 1GB, irrespective of the network configuration, which shows that FASP can transport data very quickly. Another study [87] analyses that, using Aspera Enterprise Client and Aspera

[4]The C++ library containing the UDT API implementation and programming examples can be found here: http://udt.sourceforge.net/software.html.

[5]complete documentation can be found here: http://pubs.vmware.com/hybridcloud-25/index.jsp.

[6]http://asperasoft.com/

[7]The Aspera FASP utility is publically available at http://downloads.asperasoft.com/downloads

TABLE 3. Comparison of different TCP variants RB-UDP, Tsunami, PA-UDP, UDT, and FASP for various aspects.

	Congestion Detection	Rate Control	TCP Friendliness	Intra Protocol Fairness	Issues	Usage
RB-UDP	No Congestion Control	No Rate Control	No	No	Very aggressive protocol with no TCP friendliness. Sending rate may not be accurate and may result in large packet loss.	Open Source, No Commercial Use
Tsunami	Based on Packet Loss	Control Sending rate is reduced when loss rate is more than a threshold.	Yes	Yes	Default values of different parameters may not be suitable for all environments. This can result in under estimation or over estimation of the sending rate which may result in less throughput.	Open Source, Used by AWS
PA-UDP	Based on Packet Loss, Buffer size and Disk Processing rate	Sending rate is adjusted through change in the inter-packet delay based on information (packet loss, buffer size and disk processing rate) from receiver.	No	Yes	Transmission rate is coupled with packet loss. This results in less throughput in presence of high packet loss.	Open Source, No Commercial Use
UDT	Based on Packet Loss	Sending rate is increased on positive ACK receipt as shown in equation. Sending rate is decreased on ACK receipt.	Yes	Yes	Transmission rate is coupled with packet loss. This results in less throughput in the presence of high packet loss.	Open Source, Used by VMware vCloud Connector 2.5
FASP	Based on Queuing Delay (RTT)	Sending rate is increased when queuing delay is reduced. Sending rate is decreased when queuing delay is increased.	Not studied in literature	Not studied in literature	Transmission rate is based on queuing delay which may not be accurate because of path rerouting. This may result in less throughput.	Proprietary Protocol of Aspera, Used by AWS

Enterprise Server, FASP can achieve a speed of 10Mpbs over a 100Mbps network link while transferring a file of 10GB, when adding 300ms of latency to the network. Furthermore, the ConnectomeDB[8] [88] uses Aspera FASP to enable high speed downloads for distributing human medical data to the public over a web-based user interface. We note that, since Aspera claims that FASP achieves TCP fairness, FASP needs further analysis for its TCP friendliness as there is no literature available.

Table 3 summarises various aspects of five different UDP variants. In particular, we summarise the potentials of these protocols for solving various issues in relation to detecting the *network congestion*, how these protocols achieves smooth data rate and whether these protocols provide the TCP *friendliness* and *intra-protocol fairness*. In addition, we discuss the commercial viability of these protocols.

VII. COMPARISON OF DIFFERENT PROTOCOLS

We have discussed basic mechanism of various transport layer protocols designed for high speed networks, specifically, for TCP variants i.e. Section IV, application layer protocols using multiple simultaneous TCP streams in Section V and various UDP based data transfer protocols i.e. Section VI. We now discuss the performance of these protocols investigated in several research works.

A. COMPARISON OF TCP VARIANTS

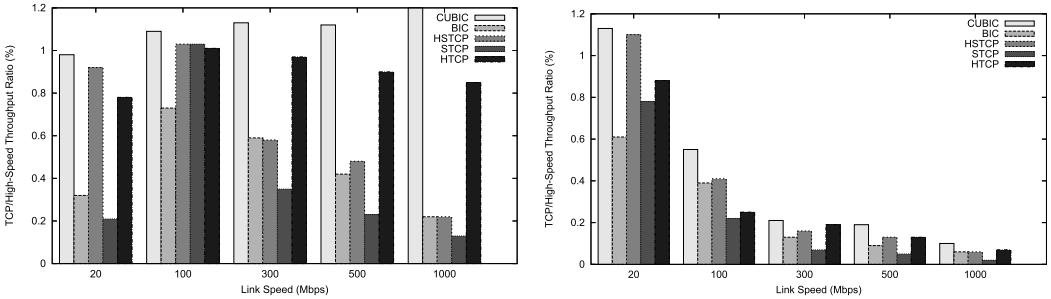
Recall that a variety of TCP variants [9], [18], [19], [25]–[28], [38], [89]–[91], among several others have been discussed in Section IV, address the under utilisation of network's resources in high-bandwidth delay networks due to the slow growth of TCP *congestion window*. Majority of these protocols manage to achieve *TCP friendliness* i.e. coexistence of TCP flows with other communicating flows, and *fairness* i.e. bandwidth sharing with other competing flows or flows with different RTTs, by modifying the *congestion window* growth of TCP.

An example evaluation of *TCP friendliness* is presented in [26] for long and short RTT networks in terms of throughput ratio for different link speeds (in Mbps). In this particular scenario [26], the authors set the RTT to 10ms and 100ms with bottleneck bandwidth ranging from 20Mbps to 1Gbps, these results are shown in Figure 7 (a) and (b) respectively for short and long RTT networks. It can be observed, Figure 7 (a), that as the bottleneck bandwidth increases, from 20Mbps to 1Gbps, the CUBIC and HTCP[9] [38] consistently performs better for *TCP friendliness* and both effectively coexist with the basic TCP flows. Alternatively, the TCP throughput ratio for BIC, HSTCP and STCP consistently decreases with the increase of bottleneck bandwidth, indicating unfair use of bandwidth with respect to TCP. Similarly, Figure 7 (b) for long RTT (e.g. 100ms in this example) network, all the TCP variants over 20Mbps show reasonable *TCP friendliness*. However, as the bottleneck bandwidth is increased from 20Mbps to 1Gbps, all the tested TCP variants perform less *TCP friendliness* and takes majority of the bandwidth. Among various TCP variants, CUBIC shows better *TCP friendliness*, followed by HTCP and HSTCP.

This study [26] further investigates the *stability* of these TCP variants. *Stability* is defined in different terms in the literature e.g. the smoothness in transmission rate variations (less oscillations) [26] or a protocol that converges to equilibrium (defined by the Control theory). The *stability* of TCP variants is evaluated for four high-speed TCP flows over long (220ms) and short (20ms) RTT network links of 10Gbps connected via the bottleneck bandwidth link of 2.5Gbps. In order to evaluate the *stability* of various protocols, the authors vary the buffer of the bottleneck router from 200% to 20% of the bottleneck link. Figure 8 (a) – (e) show the results from simulations with 20% of the bottleneck buffer (results

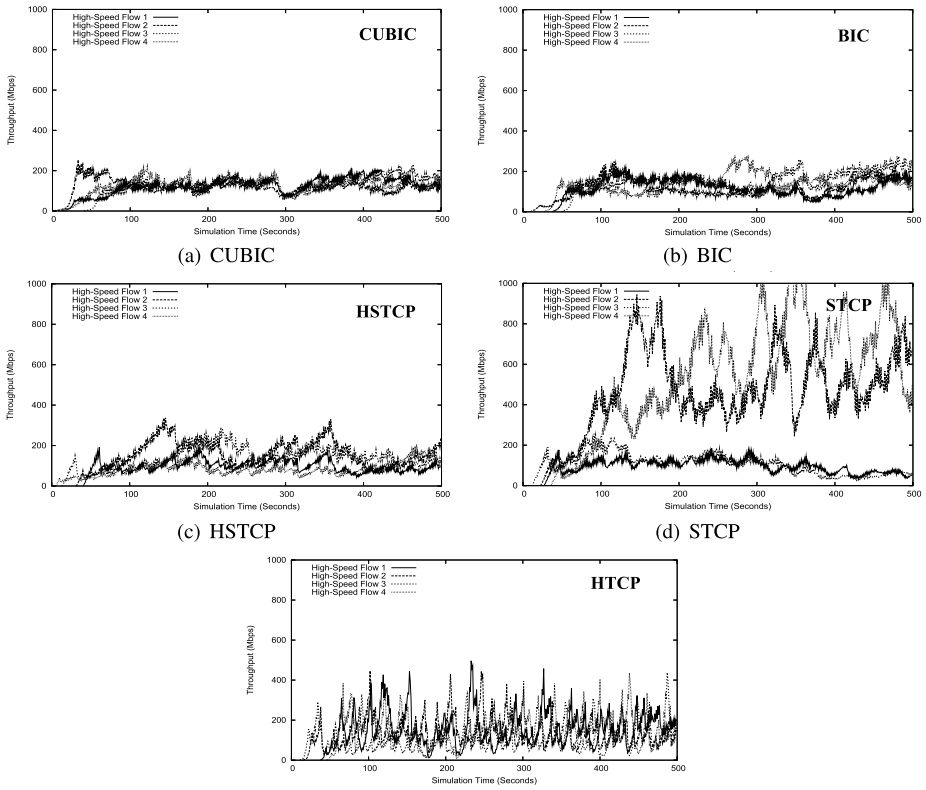
[8]ConnectomeDB is a database for housing and disseminating data about human brain structure, function, and connectivity, along with associated behavioral and demographic data [88].

[9]HTCP is another implementation of TCP for Long Fat Networks (LFN) i.e. it provides an optimised *congestion control* algorithm for high speed networks with high latency. It was created by researchers at the Hamilton Institute in Ireland. http://www.hamilton.ie/.



(a) Short-RTT Networks (b) Long-RTT Networks

FIGURE 7. TCP-friendly ratio in Short-RTT (a) Long-RTT (b) Networks [26].



(a) CUBIC (b) BIC

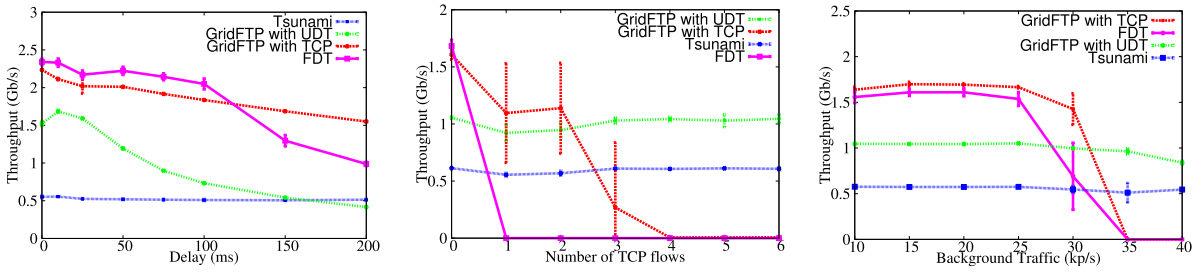
(c) HSTCP (d) STCP

(e) HTCP

FIGURE 8. Throughput of various protocols in *stability* test with 20% (a) – (e) buffer [26].

for 200% are not shown, although it shows same behavior as with 20% of the bottleneck buffer). It can be observed that the throughput achieved with STCP and HTCP highly fluctuate and hence cannot maintain communication *stability*. Conversely, CUBIC, BIC-TCP and HSTCP perform better with good *stability* than the former two TCP variants.

Furthermore, there are several other works where researchers evaluate the performance of various TCP variants (specifically those discussed in Section IV e.g. STCP, BIC-TCP, HSTCP etc.) via analytical evaluation, using simulations and via experimentations, from different perspective. For example, the authors in [92] perform a comparative analysis of FAST TCP with TCP Reno, HSTCP, STCP, and BIC-TCP for *throughput*, *intra-protocol fairness*, *stability*, and *responsiveness*. The authors determine that FAST TCP outperforms the other tested protocols for the three evaluation criteria i.e. *fairness*, *stability* and *responsiveness*, while it achieves second best overall *throughput* after the BIC-TCP. In addition, authors investigate that HSTCP and STCP achieves higher *throughput* and improved *responsiveness* compared to the TCP Reno. The STCP attains worse *intra-protocol fairness* than TCP Reno, while BIC-TCP and HSTCP accomplish similar *intra-protocol fairness* to Reno. Similarly, [93] carries out a comparative analysis TCP



(a) Varying RTT (b) TCP background flows, RTT 75ms (c) UDP background traffic, RTT 75ms

FIGURE 9. Throughput of the protocols [98].

Westwood, TCP Reno, and TCP SACK (Selective ACK) for *multi-path routes* and finds that TCP Westwood is robust to *packet reordering* introduced by the network. They realise that TCP Westwood is capable of obtaining better aggregated *throughput* than Reno and SACK when the network layer uses multiple paths, however, the authors do not discuss their performance in the presence of bottleneck [94]. Another study [95] shows that XCP converges more slowly and unnecessarily prolongs the flows due to increasing the window size of new flows and reducing the window sizes of the existing flows.

B. COMPARISON OF PROTOCOLS OF MULTIPLE TCP STREAMS

Several works have evaluate the performance of these various protocols that utilise multiple TCP connections i.e. GridFTP [33], FDT [68], BBCP [69]. The authors in [96] evaluate the GridFTP for *throughput*, *fairness* and *CPU usage* in comparison with GridCopy [97] and UDT [14] protocols for bulk data transfer. This analysis is carried out in a wide area campus network and is restricted to 2Gbps bandwidth connectivity. They investigate that GridFTP can achieve close to 95% *throughput* compared to GridCopy 82% and UDT 94.4%, similarly, they respectively achieve protocol *fairness* of 92.3%, 92% and 93.4%. However, GridFTP's *throughput* reduces with small file transfers and when it uses less than 4 connections for data transfer, in which case, it creates excessive control overhead. Another work [74] evaluates the performance of GridFTP over TCP and GridFTP over UDT in terms of *throughput* on four different geographical locations. They suggest UDT as an alternative protocol to GridFTP due to high *throughput* achieved on the entire set of testbeds [74].

There are numerous studies that investigate comparative analyses of FDT with other multiple TCP sessions enabled protocols, such as GridFTP. The authors in [98] present comparative analysis of GridFTP and FDT for long fat networks and evaluate their effectiveness in terms of *throughput* for various RTTs and the level of *congestion* induced by concurrent TCP or UDP flows. They also compare two well-known UDP variants for high-speed data transfer protocols i.e. UDT and Tsunami. This comparative analysis for *throughput* with various protocols is shown in Figure 9 with an increasing order of RTT in Figure 9 (a), number of TCP flows in Figure 9 (b) and background traffic rate i.e. Figure 9 (c); various parametric settings are also presented. It can be observed in Figure 9 (a) that FDT outperforms all the other protocols for less than 100ms RTT with the *throughput* of 2.34Gbps. However, its *throughput* rapidly decreases compared to GridFTP with TCP for greater than 100ms RTT. GridFTP with TCP performs better for greater than 100ms RTT. Overall, GridFTP and FDT with TCP outperform GridFTP with UDP or Tsunami in this setting. Furthermore, it can be observed in Figure 9 (b) that TCP-based protocols become unreliable with multiple TCP flows, e.g. FDT becomes inoperative due to induced *congestion*. In addition, *throughput* of GridFTP with TCP decrease quickly as compared to GridFTP with UDT while introducing multiple TCP flows during communication. Similarly, as shown in Figure 9 (c), *throughput* of GridFTP with TCP and FDT decreases with the increase of *congestion* induced by UDP background traffic. Alternatively, *throughput* of GridFTP with UDT and Tsunami is less affected by the UDP background traffic. Similar study [72] also evaluates the performance of GridFTP, FDT and UDT in terms of *goodput* and *fairness* in the presence of multiple traffic flows over a real network deployed in different geographical locations. Some other works e.g. [99] uses BaBar Copy Program BBCP[10] [69] to transfer hundreds of Tbytes of cosmological data between different geographic locations as well as to efficiently copy data between local systems.

C. COMPARISON OF UDP BASED PROTOCOLS

The motivation behind various modifications suggested to UDP is to achieve high performance for transferring bulk amount of data (usually in Tbytes or higher in size) around the globe e.g. across different countries or continents. This data is usually transferred over high-speed links of 10Gbps or more to researchers for storage and research and analysis. Such an example is the cosmological data generated by Dark Sky Simulations project [99] to make accessible model of the evolution of large-scale Universe and to transfer this data to research laboratories at different locations, such as, LANL,[11]

[10] BBCP is an excellent representative of peer-to-peer computing used to securely copy data from one location to another location.

[11] http://www.lanl.gov/

TABLE 4. TCP-friendliness of four protocols over 1Gbps with 160ms RTT [15].

UDP Variants	Inter-Protocol Fairness (Mbps)				Throughput (Mbps)			
	30ms		60ms		30ms		60ms	
	UDP	STCP	UDP	STCP	UDP	STCP	UDP	STCP
Tsunami	469.52	90.96	460.40	90.96	474.42	90.96	470.97	90.96
UDT	329.04		259.94		335.33		268.14	
PA-UDP	866.64		907.71		915.34		912.71	
RB-UDP	697		568		717		585	

SLAC,[12] LHC,[13] SKA[14] etc. It is critical to summarise and to accomplish comparative analyses of findings for humungous amount of data transferred with these protocols that have been carried out in the literature thus far for the benefit of the researchers.

Several works evaluate the performance of these protocols in the presence of other traffic e.g. TCP traffic, for various performance metrics, such as effect of RTT, *loss rate*, *fairness* over *throughput*, *CPU utilization*, *inter/intra-protocol fairness* etc. The authors in [15] evaluate the effect of data file size, RTT and *loss rate* over *throughput* and *inter/intra-protocol fairness* of different UDP variants of RB-UDP, Tsunami, UDT, and PA-UDP. They find that increasing the transmitting file size (e.g. from 100MB to 4GB in this experiment) has direct impact over the achieved *throughput*, where PA-UDP achieves overall highest *throughput* of up to 916Mbps followed by RB-UDP and Tsunami. The UDT, due to complex *congestion control* mechanism, achieves the least *throughput*. The *throughput* of RB-UDP declines with the files size of 2GB and higher due to the cache limitation of the end systems. They also find the same observations with varying RTTs (i.e. from 2ms to 320ms) and packet loss ratio (from 0% to 1%); they noted that, among all, the PA-UDP outperforms over others.

Further analysis [15] shows the *intra-protocol fairness* of two and four parallel flows, each UDP/STCP flow transfers a file of 3GB over 1Gbps link with 160ms RTT and 0.01% loss ratio. They examine that all the four UDP variants have similar *intra-protocol fairness*, except for PA-UDP in the existence of four parallel flows where it shows less *intra-protocol fairness* compared to two parallel flows. They further evaluate the performance of these UDP variants in the presence of TCP flows to simulate real-world networking scenario since the backhaul communication links share the bandwidth among different transport protocols. In particular, to evaluate the *inter-protocol fairness*, they use one UDP-based flow from each of the four UDP variants and one STCP flow while transferring 3GB files over 1Gbps link with 30ms and 60ms RTT along with 0.01% loss ratio. The *inter-protocol fairness* together with the achieved *throughput* are shown in Table 4. They determine that all the tested UDP variants provide good *TCP friendliness* where the TCP flows achieve reasonable *throughput* of up to 91Mbps over two different communication links.

[12]https://www6.slac.stanford.edu/
[13]http://home.cern/topics/large-hadron-collider
[14]http://www.ska.gov.au/

Another study [100] examines the *throughput* and *packet loss* of PA-UDP and Tsunami over a Gigabit Ethernet switch on a Local Area Network (LAN) with the buffer size of 750MB, while files ranging from 100MB to 5GB are being transferred using the two protocols. The authors find that PA-UDP results in better *throughput* and virtually zero *packet loss* compared to Tsunami, e.g. PA-UDP results in the *throughput* of 934Mbps and zero *packet loss* for the file transfer of 1GB, while Tsunami only achieves 295Mbps *throughput* and 41.5% *packet loss*, which in addition, results in *timeout* for a file transfer of 3GB and 5GB. The authors in [100] conclude that the Tsunami protocol cannot be set with optimal settings since its performance is not predictable and is not suitable for congested networks [31]. Similarly, another study [101] evaluates the performance of UDT, RB-UDP, Tsunami along with the basic UDP in terms of *packet delay*, *jitter*, *packet loss* and *throughput*. The authors observe that the basic UDP, UDT and Tsunami experience higher *packet delay* and *jitter* compared to RB-UDP. Furthermore, UDT experiences higher *packet loss* compared to other UDP variants, with RB-UPD experiencing the least *packet loss* while higher *throughput* than the other UDP variants. Hence, the aggressive bulk data transfer RB-UDP protocol has been designed for extremely high bandwidth networks and for achieving high Quality-of-Service (QoS) [102].

There are several other works [87], [88], [103], [104] that utilise FASP for downloading massive files of several gigabytes, such as medical imaging data of several thousand objects collected over several years. An example of data transfer between the universities in US and UK, the FASP achieves a rate of 70–85Mbps while basic File Transfer Protocol (FTP) can only obtain a transfer speed of 4–12Mbps [103]. Hence, downloading a typical file of 10GB between these two locations would take less than 15min over FASP compared to almost 3hrs over FTP. The authors in [104] examine that FASP manages to transfer at the rate of 6Gbps over a 10Gbps of link bandwidth when using an MTU of 1500byes. They further evaluate the FASP's utilisation during the handover between a lower bandwidth and higher delay networks. The analysis [104] shows that the *packet loss* rate highly fluctuates and increases with the increase of available bandwidth of 100, 400, and 1000Mbps. However, they do not provide any firm justifications for this behaviour of FASP, additionally; they do not provide analysis of FASP along with other transport protocols during the handover process for communicating data among heterogenous access networks.

Table 5 presents summary of key findings from literature using different evaluation criteria for various TCP and UDP variants and protocols with multiple TCP streams.

VIII. FUTURE RESEARCH DIRECTIONS

We note that various multimedia real-time applications utilise connectionless transport protocols, such as UDP and its variants, across the internet community. We further, to the best of our knowledge, note that researchers have evaluated the

TABLE 5. Summary of key findings by various works in literature using different evaluation criteria for various TCP and UDP variants and for protocols with multiple TCP streams.

Article	Protocols Compared	Evaluation Criteria	Key Results
			TCP based protocols
[26]	BIC, CUBIC, HTCP, HSTCP and STCP	*TCP friendliness* and *stability*	• CUBIC and HTCP achieve better *TCP friendliness* over bottleneck bandwidth of 20Mbps to 1Gbps and 10ms delay • All TCP variants show less *TCP friendliness* over bottleneck bandwidth of 20Mbps to 1Gbps and 100ms delay • BIC, HSTCP and STCP are unfair to TCP • Overall CUBIC shows better *TCP friendliness* followed by HTCP and HSTCP
[92]	FAST TCP, TCP Reno, HSTCP, STCP and BIC-TCP	*Throughput*, *intra-protocol fairness*, *stability*, and *responsiveness*	• FAST TCP outperforms for the three evaluation criteria i.e. *fairness*, *stability* and *responsiveness* except *throughput* • FAST TCP achieves second best overall *throughput* after the BIC-TCP • HSTCP and STCP achieves higher *throughput* and improved *responsiveness* compared to the TCP Reno • STCP attains worse *intra-protocol fairness* than TCP Reno • BIC-TCP and HSTCP accomplish similar *intra-protocol fairness* to TCP Reno
[93]	TCP Westwood, TCP Reno and TCP SACK	*Multi-path routes*, *packet reordering* and *throughput*	• TCP Westwood is robust to *packet reordering* • TCP Westwood is capable of obtaining better aggregated *throughput* than Reno and SACK in multiple paths at network layer
[95]	XCP	*Convergence* and *latency*	• XCP converges slowly and unnecessarily prolongs the flows
			Data transfer application layer protocols using multiple simultaneous TCP streams
[96]	GridFTP, GridCopy and UDT	*Throughput*, *fairness* and *CPU usage*	• GridFTP can achieve better *throughput* and *fairness* compared to GridCopy and UDT • GridFTP's *throughput* reduces with small file transfers and creates excessive control overhead
[74]	GridFTP, TCP and UDT	*Throughput*	• UDT as an alternative to GridFTP due to high *throughput* achieved

TABLE 5. *(Continued.)* **Summary of key findings by various works in literature using different evaluation criteria for various TCP and UDP variants and for protocols with multiple TCP streams.**

Article	Protocols Compared	Evaluation Criteria	Key Results
[98]	GridFTP, FDT, UDT and Tsunami	*Throughput*, *congestion* induced by concurrent TCP or UDP flows	• FDT outperforms for *throughput* over other protocols for less than 100ms RTT, however, its *throughput* rapidly decreases compared to GridFTP with TCP for greater than 100ms RTT • GridFTP with TCP performs better for greater than 100ms RTT • GridFTP and FDT with TCP outperform GridFTP with UDP or Tsunami • FDT becomes inoperative due to *congestion* induced by multiple TCP flows • The *throughput* of GridFTP with TCP decreases quickly as compared to GridFTP with UDT while introducing multiple TCP flows • The *throughput* of GridFTP with TCP and FDT decreases with the increase of *congestion* induced by UDP background traffic • The *throughput* of GridFTP with UDT and Tsunami is less affected by the UDP background traffic
[72]	GridFTP, FDT and UDT	*Goodput* and *fairness*	• GridFTP achieves higher *goodput* followed by FDT and then UDT • FDT achieves good *fairness* compared to GridFTP and UDT
		UDP based data transfer protocols	
[15]	RB-UDP, Tsunami, UDT and PA-UDP	*Throughput* and *inter/intra-protocol fairness*	• PA-UDP achieves overall highest *throughput* followed by RB-UDP and Tsunami • UDT, due to complex *congestion control* mechanism, achieves the least *throughput* • The *throughput* performance of RB-UDP decrease with higher files sizes due to the cache limitation of the end systems • PA-UDP outperforms over others with varying RTT and packet loss ratio • UDB variants have similar *intra-protocol fairness*, except for PA-UDP • All UDP variants provide good *TCP friendliness*
[100]	PA-UDP and Tsunami	*Throughput* and *packet loss*	• PA-UDP results in better *throughput* and virtually zero *packet loss* compared to Tsunami
[101]	UDT, RB-UDP, Tsunami and UDP	*Packet delay*, *jitter*, *packet loss* and *throughput*	• UDP, UDT and Tsunami observe higher *packet delay* and *jitter* compared to RB-UDP • UDT experiences higher *packet loss* compared to other UDP variants • RBUPD achieves higher *throughput* than other UDP variants • RB-UDP is designed for extremely high bandwidth networks
[103]	FASP and FTP	Achieved *data rate*	• Downloading a file of 10GB would take less than 15min over FASP compared to almost 3hrs using FTP
[104]	FASP	*Packet loss*	• The *packet loss* rate highly fluctuates and increases with the increase of available bandwidth of 100, 400, and 1000Mbps

performance of different UDP variants for static data file transfers. However, it requires investigating the performance of real-time multimedia applications, such as VoIP, live video/voice streaming etc., for maintaining the QoS of these applications for various performance metrics such as *packet loss*, *jitter*, *end-to-end delay* etc. Additionally, the *inter-protocol fairness* of FASP also needs to be investigated with other transport layer protocols since it makes use of majority of the capacity of a communication link. Furthermore, the *TCP friendliness* of FASP needs to be studied.

Recall that the transport layer protocols have significant role in determining the end-to-end performance in a network; it is difficult to achieve high throughput despite of high bandwidth provisioning networks e.g. available bandwidth of 5G mmWave[15] in 5G networks. Hence, it is important to see the effect of various parameters within transport layer protocol variants over the newly introduced features and capabilities of 5G networks e.g. *Network slicing*. In particular: **1.)** The use of MTU (Maximum Transmission Unit) instead of MSS (Maximum Segment Size) and optimising their values for 5G mmWave is challenging; since the TCP with default MTU size has been proved to be under utilising the high bandwidth of 5G mmWave networks. **(2.)** The initial congestion window of TCP with slow start seems not suitable for 5G and beyond networks; it is due to the reason that data sending rate in 5G networks is tremendously high whereas the small `cwnd` starting can take longer to efficiently utilise the available 5G network's bandwidth. **(3.)** The exponential backoff retransmission timeout may have severe effects over the performance of TCP for 5G and beyond networks, since when the long failure occurs, the probability of triggering timeouts are high that may have negative effect other the performance of 5G networks. **(4.)** The TCP loss detection works fine for wired networks, however, the blockage issues in 5G networks can lower the their performance that demands for mechanism for improving the loss detection mechanisms.

The smart devices i.e. smartphones, tablets, notebooks, are used for a variety of purposes and it is evident [105] that they would exceed the world's population and the average mobile cellular connection speed will reach 43.9Mbps by 2023; 5G connection speed will reach 575Mbps by 2023. Moreover, majority of mobile applications and information sharing is via video on demand. Hence, it is vital to evaluate various transport layer protocols for user experience while accessing high bandwidth demanding applications. One of the possible directions is to evaluate the extra processing that occurs in the user device's processor for accessing these applications via various UDP variants. Similarly, it also needs to evaluate the effect over battery power required to transfer data with different transport layer protocols, since some transport protocols (e.g. basic TCP vs. basic UDP) add more overheads compared to others e.g. extra computation, communication overhead. In addition, the extra communication overhead will also affect users with fixed 5G/6G data plan.

[15]Detailed discussion over 5G mmWave network procedures and parameters for reliable end-to-end communication can be found in https://upcommons.upc.edu/bitstream/handle/2117/331626/09205403.pdf

IX. CONCLUSION

In this survey, a review of several innovative transport layer protocols replacing the legacy transport protocols along with their comparison in terms of several operational performance metrics of *throughput*, *packet loss*, *inter/intra protocol fairness* etc. and non-operational criteria e.g. deployment, is presented. Additionally, we classify these protocols into three categories of *reliable*, *unreliable* and *protocols that use multiple protocol's streams*. We demonstrate the working mechanism of these protocols and further investigate their performance for transporting huge volumes of data in high-speed and low-latency networks that have been carried out in recent research literature. In summary, recent research works on innovative transport protocols and their comparative investigations have significant importance over fulfilling the requirements of emerging data-intensive applications in the presence of high-speed networks. We summarise and analyse them together into single article and we truly believe this research to be a source of motivation towards development and deployment of data-intensive applications in high-speed networks.

REFERENCES

[1] J. Lei and L. Kong, "Fundamentals of big data in radio astronomy," in *Big Data in Astronomy*. Amsterdam, The Netherlands: Elsevier, 2020, pp. 29–58.
[2] P. Pavarangkoon, K. T. Murata, K. Yamamoto, K. Muranaga, A. Higuchi, T. Mizuhara, Y. Kagebayashi, C. Charnsripinyo, N. Nupairoj, T. Ikeda, J. Tanaka, and K. Fukazawa, "Development of international mirroring system for real-time Web of meteorological satellite data," *Earth Sci. Informat.*, vol. 13, pp. 1461–1476, Aug. 2020.
[3] P. Chamoso, A. González-Briones, A. Rivas, F. D. L. Prieta, and J. M. Corchado, "Social computing in currency exchange," *Knowl. Inf. Syst.*, vol. 61, no. 2, pp. 733–753, Nov. 2019.
[4] J. S. Allwood, N. Fierer, R. R. Dunn, M. Breen, B. J. Reich, E. B. Laber, J. Clifton, N. S. Grantham, and S. A. Faith, "Use of standardized bioinformatics for the analysis of fungal DNA signatures applied to sample provenance," *Forensic Sci. Int.*, vol. 310, May 2020, Art. no. 110250.
[5] A. D. Baxevanis, G. D. Bader, and D. S. Wishart, *Bioinformatics*. Hoboken, NJ, USA: Wiley, 2020.
[6] R. Blossey, *Computational Biology: A Statistical Mechanics Perspective*. Boca Raton, FL, USA: CRC Press, 2019.
[7] S. M. Larson, C. D. Snow, M. Shirts, and V. S. Pande, "Folding@home and Genome@home: Using distributed computing to tackle previously intractable problems in computational biology," 2009, *arXiv:0901.0866*. [Online]. Available: http://arxiv.org/abs/0901.0866
[8] L. Breitwieser, A. Hesam, J. D. Montigny, V. Vavourakis, A. Iosif, J. Jennings, M. Kaiser, M. Manca, A. D. Meglio, Z. Al-Ars, and F. Rademakers, "BioDynaMo: An agent-based simulation platform for scalable computational biology research," *bioRxiv*, 2020.
[9] C. Jin, D. Wei, S. H. Low, J. Bunn, H. D. Choe, J. C. Doyle, H. Newman, S. Ravot, S. Singh, F. Paganini, G. Buhrmaster, L. Cottrell, O. Martin, and W.-C. Feng, "FAST TCP: From theory to experiments," *IEEE Netw.*, vol. 19, no. 1, pp. 4–11, Jan. 2005.
[10] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*. London, U.K.: Pearson, 2005.
[11] I. Ullah, Z. Shah, and A. Baig, "S-TFRC: An efficient rate control scheme for multimedia handovers," *Comput. Sci. Inf. Syst.*, vol. 13, no. 1, pp. 45–69, 2016.
[12] Z. Shah, A. Suleman, I. Ullah, and A. Baig, "Effect of transmission opportunity and frame aggregation on VoIP capacity over IEEE 802.11n WLANs," in *Proc. 8th Int. Conf. Signal Process. Commun. Syst. (ICSPCS)*, Dec. 2014, pp. 1–7.

[13] P. A. Dana, Z. Esmaeilbeig, and M.-R. Sadeghi, "Reliability enhancement and packet loss recovery of any steganographic method in voice over IP," *Wireless Netw.*, vol. 26, pp. 5817–5823, Mar. 2020.
[14] Y. Gu and R. L. Grossman, "UDT: UDP-based data transfer for high-speed wide area networks," *Comput. Netw.*, vol. 51, no. 7, pp. 1777–1799, May 2007.
[15] Z. Yue, Y. Ren, and J. Li, "Performance evaluation of UDP-based high-speed transport protocols," in *Proc. IEEE 2nd Int. Conf. Softw. Eng. Service Sci. (ICSESS)*, Jul. 2011, pp. 69–73.
[16] I. Ullah, K. Sattar, Z. U. Qamar, W. Sami, and A. Ali, "Transmissions failures and load-balanced routing metric for wireless mesh networks," in *Proc. 8th Int. Conf. High-Capacity Opt. Netw. Emerg. Technol.*, Dec. 2011, pp. 159–163.
[17] S. Floyd, S. Ratnasamy, and S. Shenker, "Modifying TCP's congestion control for high speeds," Tech. Rep., May 2002.
[18] S. Floyd *et al.*, "Highspeed TCP for large congestion windows," Tech. Rep., 2003.
[19] T. Kelly, "Scalable TCP: Improving performance in highspeed wide area networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 2, pp. 83–91, Apr. 2003.
[20] C. Jin, D. Wei, S. H. Low, G. Buhrmaster, J. Bunn, D. H. Choe, R. L. A. Cottrell, J. C. Doyle, H. Newman, F. Paganini, S. Ravot, and S. Singh, "FAST kernel: Background theory and experimental results," in *Proc. 1st Int. Workshop Protocols Fast Long-Distance Netw.*, 2003, pp. 3–4.
[21] I. Ullah, Z. Shah, M. Owais, and A. Baig, "VoIP and tracking capacity over WiFi networks," in *Proc. IEEE 73rd Veh. Technol. Conf. (VTC Spring)*, May 2011, pp. 1–5.
[22] J. Korhonen, S. Park, J. Zhang, C. Hwang, and P. Sarolahti. (2006). *Link Characteristic Information for IP Mobility Problem Statement*. [Online]. Available: https://draft-korhonenmobopts-link-characteristics-ps-01.txt
[23] Z. Shah, A. Baig, H. Samir, and I. Ullah, "State aware enhancement in DCCP for multimedia handovers," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2012, pp. 3328–3333.
[24] S. Ullah, I. Ullah, H. K. Qureshi, R. Haw, S. Jang, and C. S. Hong, "Passive packet loss detection in Wi-Fi networks and its effect on HTTP traffic characteristics," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, Feb. 2014, pp. 428–432.
[25] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control (BIC) for fast long-distance networks," in *Proc. 33rd Annu. Joint Conf. IEEE Comput. Commun. Soc. (INFOCOM)*, vol. 4, Mar. 2004, pp. 2514–2524.
[26] S. Ha, I. Rhee, and L. Xu, "CUBIC: A new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operating Syst. Rev.*, vol. 42, no. 5, pp. 64–74, Jul. 2008.
[27] C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi, and R. Wang, "TCP westwood: End-to-end congestion control for wired/wireless networks," *Wireless Netw.*, vol. 8, no. 5, pp. 467–479, Sep. 2002.
[28] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 4, pp. 89–102, Oct. 2002.
[29] E. He, J. Leigh, O. Yu, and T. A. Defanti, "Reliable blast UDP: Predictable high performance bulk data transfer," in *Proc. IEEE Int. Conf. Cluster Comput.*, Sep. 2002, pp. 317–324.
[30] Y. Gu and R. L. Grossman, "UDT: An application level transport protocol for grid computing," in *Proc. 2nd Int. Workshop Protocols Fast Long-Distance Netw.*, 2003, pp. 1–3.
[31] B. Eckart, X. He, and Q. Wu, "Performance adaptive UDP for high-speed bulk data transfer over dedicated links," in *Proc. IEEE Int. Symp. Parallel Distrib. Process. (IPDPS)*, Apr. 2008, pp. 1–10.
[32] M. R. Meiss, "Tsunami: A high-speed rate-controlled protocol for file transfer," Indiana Univ., Bloomington, IN, USA, Tech. Rep., 2004.
[33] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, and S. Tuecke, "GridFTP: Protocol extensions to FTP for the grid," *Global Grid ForumGFD-RP*, vol. 20, pp. 1–21, 2003.
[34] V. Jacobson, "Congestion avoidance and control," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 18, no. 4, pp. 314–329, 1988.
[35] B. Turkovic, F. A. Kuipers, and S. Uhlig, "Fifty shades of congestion control: A performance and interactions evaluation," 2019, *arXiv:1903.03852*. [Online]. Available: http://arxiv.org/abs/1903.03852
[36] J. Postel, *Transmission Control Protocol Specification*, document RFC 793, ARPANET Working Group, 1981.
[37] M. Allman *et al.*, "TCP congestion control," Tech. Rep., 1999.
[38] D. Leith and R. Shorten, "H-TCP: TCP for high-speed and long-distance networks," in *Proc. PFLDnet*, 2004, pp. 1–16.
[39] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang, "TCP westwood: Bandwidth estimation for enhanced transport over wireless links," in *Proc. 7th Annu. Int. Conf. Mobile Comput. Netw.*, 2001, pp. 287–297.
[40] L. A. Grieco and S. Mascolo, "Performance evaluation and comparison of Westwood+, New Reno, and Vegas TCP congestion control," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 2, pp. 25–38, Apr. 2004.
[41] K. Yamada, R. Wang, M. Y. Sanadidi, and M. Gerla, "TCP Westwood with agile probing: Dealing with dynamic, large, leaky pipes," in *Proc. IEEE Int. Conf. Commun.*, vol. 2, Jun. 2004, pp. 1070–1074.
[42] D. Kliazovich, F. Granelli, and D. Miorandi, "Logarithmic window increase for TCP Westwood+ for improvement in high speed, long distance networks," *Comput. Netw.*, vol. 52, no. 12, pp. 2395–2410, Aug. 2008.
[43] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, "TIMELY: RTT-based congestion control for the datacenter," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 537–550, 2015.
[44] M. Hock, F. Neumeister, M. Zitterbart, and R. Bless, "TCP LoLa: Congestion control for low latencies and high throughput," in *Proc. IEEE 42nd Conf. Local Comput. Netw. (LCN)*, Oct. 2017, pp. 215–218.
[45] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-based congestion control," *Queue*, vol. 14, no. 5, pp. 20–53, Oct. 2016.
[46] C. Caini and R. Firrincieli, "TCP hybla: A TCP enhancement for heterogeneous networks," *Int. J. Satell. Commun. Netw.*, vol. 22, no. 5, pp. 547–566, Sep. 2004.
[47] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Trans. Netw.*, vol. 1, no. 4, pp. 397–413, Aug. 1993.
[48] K. Ramakrishnan *et al.*, "The addition of explicit congestion notification (ECN) to IP," Tech. Rep., 2001.
[49] S. Floyd and K. Fall. (1997). *Router Mechanisms to Support End-to-End Congestion Control*. [Online]. Available: http://www.nrg.ee.lbl.gov/floyd/end2end-paper.html
[50] D. Lin and R. Morris, "Dynamics of random early detection," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 27, no. 4, pp. 127–137, Oct. 1997.
[51] T. J. Ott, T. V. Lakshman, and L. H. Wong, "SRED: Stabilized RED," in *Proc. IEEE 18th Annu. Joint Conf. IEEE Comput. Commun. Soc. (INFOCOM)*, vol. 3, Mar. 1999, pp. 1346–1355.
[52] R. Pan, B. Prabhakar, and K. Psounis, "CHOKe—A stateless active queue management scheme for approximating fair bandwidth allocation," in *Proc. 19th Annu. Joint Conf. IEEE Comput. Commun. Soc. (INFOCOM)*, vol. 2, Mar. 2000, pp. 942–951.
[53] W. R. Stevens, "TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms," Tech. Rep., 1997.
[54] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP selective acknowledgment options," Tech. Rep., 1996.
[55] R.-S. Cheng, H.-T. Lin, W.-S. Hwang, and C.-K. Shieh, "Improving the ramping up behavior of TCP slow start," in *Proc. 19th Int. Conf. Adv. Inf. Netw. Appl. (AINA)*, vol. 1, 2005, pp. 807–812.
[56] H. Wang, H. Xin, D. S. Reeves, and K. G. Shin, "A simple refinement of slow-start of TCP congestion control," in *Proc. 5th IEEE Symp. Comput. Commun. (ISCC)*, Jul. 2000, pp. 98–105.
[57] N. Hu and P. Steenkiste, "Improving TCP startup performance using active measurements: Algorithm and evaluation," in *Proc. 11th IEEE Int. Conf. Netw. Protocols*, Nov. 2003, pp. 107–118.
[58] B. Moraru, F. Copaciu, G. Lazar, and V. Dobrota, "Practical analysis of TCP implementations: Tahoe, Reno, Newreno," in *Proc. RoEduNet Int. Conf.*, vol. 1, 2003, pp. 125–138.
[59] F. Baccelli, G. Carofiglio, and M. Piancino, "Stochastic analysis of scalable TCP," in *Proc. IEEE INFOCOM*, Apr. 2009, pp. 19–27.
[60] G. Huston, "Gigabit TCP," *Internet Protocol J.*, 2006.
[61] H. Jamal and K. Sultan, "Performance analysis of TCP congestion control algorithms," *Int. J. Comput. Commun.*, vol. 2, no. 1, pp. 18–24, 2008.
[62] M. Nirmala and R. V. Pujeri, "Performance of TCP Vegas, Bic and Reno congestion control algorithms on iridium satellite constellations," *Int. J. Comput. Netw. Inf. Secur.*, vol. 4, no. 12, p. 40, 2012.
[63] D. Lopez-Pacheco and C. Pham, "Performance comparison of TCP, HSTCP and XCP in high-speed, highly variable-bandwidth environments," in *Proc. IEEE 3rd Int. Conf. Netw. Protocols (ICNP)*, Berlin, Germany, Oct. 2004.

[64] R. J. La, J. Walrand, and V. Anantharam, *Issues in TCP vegas*. Princeton, NJ, USA: Citeseer, 1999.
[65] K. Ramakrishnan and S. Floyd, *A Proposal to Add Explicit Congestion Notification (ECN) to IP*, RFC 2481, Jan. 1999.
[66] (2016). *Akamai Acquires Fastsoft*. [Online]. Available: https://www.akamai.com/us/en/about/news/press/2012-press/akamai-acquires-fastsoft.jsp
[67] (2016). *Implementation of XCP in NS*. [Online]. Available: http://www.isi.edu/nsnam/ns/doc/node238.html
[68] I. Legrand, H. Newman, R. Voicu, C. Cirstoiu, C. Grigoras, C. Dobre, A. Muraru, A. Costan, M. Dediu, and C. Stratan, "MonALISA: An agent based, dynamic service system to monitor, control and optimize distributed systems," *Comput. Phys. Commun.*, vol. 180, no. 12, pp. 2472–2498, Dec. 2009.
[69] A. Hanushevsky, A. Trunov, and L. Cottrell, "Peer to peer computing for secure high performance data copying," Tech. Rep., SLAC-PUB-8908, 2001.
[70] W. Allcock, J. Bresnahan, R. Kettimuthu, and M. Link, "The globus striped GridFTP framework and server," in *Proc. ACM/IEEE Conf. Supercomput.* Washington, DC, USA: IEEE Computer Society, 2005, p. 54.
[71] R. Esposito, P. Mastroserio, G. Tortone, and F. Taurino, "Standard FTP and GridFTP protocols for international data transfer in Pamela satellite space experiment," 2003, *arXiv preprint hep-ex/0305084*. [Online]. Available: https://arxiv.org/abs/hep-ex/0305084
[72] S.-Y. Yu, N. Brownlee, and A. Mahanti, "Characterizing performance and fairness of big data transfer protocols on long-haul networks," in *Proc. IEEE 40th Conf. Local Comput. Netw. (LCN)*, Oct. 2015, pp. 213–216.
[73] (2016). *Research Data Management Simplified*. [Online]. Available: https://www.globus.org/
[74] J. Bresnahan, M. Link, R. Kettimuthu, and I. Foster, "UDT as an alternative transport protocol for GridFTP," in *Proc. 7th Int. Workshop Protocols Future, Large-Scale Diverse Netw. Transports (PFLDNeT)*, Tokyo, Japan: Citeseer, 2009.
[75] (2016). *SLAC: National Acecellration Laboratory*. [Online]. Available: https://www6.slac.stanford.edu/
[76] *Faster Bulk Transfer Starring: UDP*. (2016). [Online]. Available: http://www.csm.ornl.gov/~dunigan/net100/udp/UDP_Tsunami.html
[77] (2016). *Cloud, Big Data and Mobile*. [Online]. Available: http://harish11g.blogspot.com.au/
[78] (2016). *Moving Big Data Into the Cloud With Tsunami UDP*. [Online]. Available: https://blogs.aws.amazon.com/bigdata/post/Tx33R88KHCWEOHT/Moving-Big-Data-into-the-Cloud-with-Tsunami-UDP
[79] A. S. Sait. (Dec. 2014). *Strategies for Migrating Oracle Database to AWS*. [Online]. Available: https://d0.awsstatic.com/whitepapers/strategies-for-migrating-oracle-database-to-aws.pdf
[80] S. Shin, K. Dhondge, and B.-Y. Choi, "Understanding the performance of TCP and UDP-based data transfer protocols using EMULAB," Tech. Rep., 2012.
[81] Y. Gu, X. Hong, and R. L. Grossman, "Experiences in design and implementation of a high performance transport protocol," in *Proc. ACM/IEEE Conf. Supercomput.* Washington, DC, USA: IEEE Computer Society, Nov. 2004, p. 22.
[82] (2016). *White Paper, Ultra High-Speed Transport Technology*. [Online]. Available: http://asperasoft.com/resources/white-papers/ultra-high-speed-transport-technology/
[83] Y. Ren, H. Tang, J. Li, and H. Qian, "Performance comparison of UDP-based protocols over fast long distance network," Tech. Rep., 2009.
[84] C. Xie, "A dynamic performance-based flow control method for high-speed data transfer," Tech. Rep.
[85] W. Wang, M. Tang, Y. Ren, and J. Li, "Characterization and evaluation of end-system performance aware transport schemes for fast long-distance optical networks," *Inf. Technol. J.*, vol. 9, no. 4, pp. 766–773, May 2010.
[86] Z. Conghua and C. Meiling, "Analysis of fast and secure protocol based on continuous-time Markov chain," *China Commun.*, vol. 10, no. 8, pp. 137–149, Aug. 2013.
[87] S. L. Pakula and R. O. Ernst, "Method and system of transmitting data over a network using a communication protocol," U.S. Patent 20 150 304 459, Oct. 22, 2015.
[88] M. R. Hodge, W. Horton, T. Brown, R. Herrick, T. Olsen, M. E. Hileman, M. McKay, K. A. Archie, E. Cler, M. P. Harms, G. C. Burgess, M. F. Glasser, J. S. Elam, S. W. Curtiss, D. M. Barch, R. Oostenveld, L. J. Larson-Prior, K. Ugurbil, D. C. Van Essen, and D. S. Marcus, "ConnectomeDB—Sharing human brain connectivity data," *NeuroImage*, vol. 124, pp. 1102–1107, Jan. 2016.
[89] T. Hatano, M. Fukuhara, H. Shigeno, and K.-I. Okada, "TCP-friendly SQRT TCP for high speed networks," in *Proc. APSITT*, 2003, pp. 455–460.
[90] Y. Gu, X. Hong, M. Mazzucco, and R. Grossman, "SABUL: A high performance data transfer protocol," *IEEE Commun. Lett.*, 2003.
[91] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 63–74, Aug. 2010.
[92] D. X. Wei, C. Jin, S. H. Low, and S. Hegde, "FAST TCP: Motivation, architecture, algorithms, performance," *IEEE/ACM Trans. Netw.*, vol. 14, no. 6, pp. 1246–1259, Dec. 2006.
[93] M. Gerla, S. Lee, and G. Pau, "TCP Westwood simulation studies in multiple-path cases," in *Proc. SPECTS*, 2002, pp. 1–7.
[94] J. Iyengar, "Concurrent multipath transfer using SCTP multihoming," in *Proc. Multihoming Commun. SCTP, Stream Control Transmiss. Protocol*, 2012, p. 99.
[95] N. Dukkipati and N. McKeown, "Why flow-completion time is the right metric for congestion control," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 1, pp. 59–62, Jan. 2006.
[96] J. Suresh, A. Srinivasan, and A. Damodaram, "Performance analysis of various high speed data transfer protocols for streaming data in long fat networks," in *Proc. Int. Conf. Recent Trends Inf., Telecommun. Comput. (ITC)*, Mar. 2010, pp. 234–237.
[97] R. Kettimuthu, W. Allcock, L. Liming, J.-P. Navarro, and I. Foster, "GridCopy: Moving data fast on the grid," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, Mar. 2007, p. 363.
[98] S.-Y. Yu, N. Brownlee, and A. Mahanti, "Comparative performance analysis of high-speed transfer protocols for big data," in *Proc. 38th Annu. IEEE Conf. Local Comput. Netw.*, Oct. 2013, pp. 292–295.
[99] S. W. Skillman, M. S. Warren, M. J. Turk, R. H. Wechsler, D. E. Holz, and P. M. Sutter, "Dark sky simulations: Early data release," 2014, *arXiv:1407.2600*. [Online]. Available: http://arxiv.org/abs/1407.2600
[100] Y. Zhu, A. Bassi, P. Massonet, and D. Talia, "Mechanisms for high volume data transfer in grids," Inst. Knowl. Data Manage., CoreGRID, Netw. Excellence, Tech. Rep., 2007.
[101] R. K. Ahir, "Improving a performance of MPEG video streams with different UDP variants," *Int. J. Adv. Res. Comput. Commun. Eng.*, vol. 2, no. 12, pp. 1–4, 2013.
[102] E. Sundararajan, A. Harwood, and K. Ramamohanarao, "Lossy bulk synchronous parallel processing model for very large scale grids," 2006, *arXiv:cs/0611091*. [Online]. Available: https://arxiv.org/abs/cs/0611091
[103] D. S. Marcus *et al.*, "Human connectome project informatics: Quality control, database services, and data visualization," *NeuroImage*, vol. 80, pp. 202–219, Oct. 2013.
[104] P. Hagernäs, "5G user satisfaction enabled by FASP: Evaluating the performance of Aspera's FASP," Tech. Rep., 2015.
[105] (2014). *Cisco Annual Internet Report (2018–2023) White Paper*. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

KHAWAR KHURSHID received the Ph.D. degree from Michigan State University (MSU). He is currently an Associate Professor with the National University of Sciences and Technology (NUST), Pakistan. He is the Director of the Center of Excellence in FPGA and ASIC Research (CEFAR), focusing on solutions development on PSoCs and embedded hardware. He specializes in signal processing and communications, image processing, embedded design, pattern recognition, and computer vision. His research interests include signal encoding, image enhancement, segmentation and registration, and wearable bio-sensor modules.

IMDAD ULLAH received the Ph.D. degree in computer science and engineering from the University of New South Wales (UNSW), Sydney, Australia. He has served as a Researcher at UNSW, a Research Scholar at National ICT Australia (NICTA), Data61 CSIRO Australia, NUST, Islamabad, Pakistan, and SEEMOO TU Darmstadt, Germany, and a Research Collaborator at the SLAC National Accelerator Laboratory, Stanford University, USA. He is currently an Assistant Professor with the College of Computer Engineering and Sciences, PSAU, Saudi Arabia. He has research and development experience in privacy preserving systems, including private advertising and crypto-based billing systems. His primary research interests include privacy enhancing technologies, the Internet of Things, blockchain, network modeling and design, network measurements, and trusted networking.

ZAWAR SHAH received the Ph.D. degree in electrical engineering from the University of New South Wales (UNSW), Sydney, Australia, in 2009. He has held various academic leadership positions, including an Associate Professor in IT and the Head of the Computer Networks Group, Victorian Institute of Technology (VIT), Australia, and the National University of Sciences and Technology (NUST), Pakistan, respectively. He is currently working as a Senior Lecturer with the Sydney International School of Technology and Commerce. He has supervised many postgraduate research students, published many research articles in leading journals, and published many research papers in conferences. His research interests include security issues in software-defined networking (SDN), security issues in cloud computing, and network architectures and protocols.

NAJM HASSAN received the bachelor's degree in computer science from the University of Peshawar, in 2002, the master's degree in information technology from Gomal University, in 2006, the M.S. degree in networking from the National University of Sciences and Technology (NUST), Islamabad, Pakistan, in 2009, and the Ph.D. degree in computer science and engineering from the School of Computer Science and Engineering (CSE), UNSW, Australia, in 2018. He has over ten years diverse experience in IT industry, teach, and research with a demonstrated history of working in higher education industry. His research interests include nanoscale communication in THz band, specializing in designing efficient communication protocols for events, and nodes detection in wireless nanoscale sensor networks.

TARIQ AHAMED AHANGER is currently an Associate Professor with the Department of Information Systems, College of Computer Engineering and Sciences, Prince Sattam Bin Abdulaziz University. He has authored over 40 refereed articles. His interests include the Internet of Things, cybersecurity, and artificial intelligence.