IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 10, 2019, accepted September 29, 2019, date of publication October 9, 2019,
date of current version October 23, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2946527

Enhancing Fast TCP's Performance Using Single TCP Connection for Parallel Traffic Flows to Prevent Head-of-Line Blocking

SARFRAZ AHMAD AND MUHAMMAD JUNAID ARSHAD
Department of Computer Science and Engineering, University of Engineering and Technology at Lahore, Lahore 54890, Pakistan
Corresponding author: Sarfraz Ahmad (sarfaraz.awan@hotmail.com)

ABSTRACT Hypertext Transfer Protocol-2 (HTTP/2) partially resolved the problem of Head-of-Line Blocking (HoLB) by multiplexing independent messages at the application layer. This enables simultaneous transmission of multiple requests over the same connection independently. However, this technique becomes ineffective when packet loss occurs in the Transmission Control Protocol (TCP) flow in which case all the independent streams are blocked until the retransmission of the lost packet; this problem is known as HoLB at TCP-level. The problem arises because the underlying TCP does not differentiate between independent messages/streams from application layer protocol. This study proposes a multistream framework for Fast TCP to support multiple independent messages/streams of the application. The proposed framework uses separate flows, buffers, and segments for each independent stream, and interleaves these segments over a single TCP connection. It makes TCP compatible with HTTP/2, reduces data delivery latency between transport and application layer, and alleviates head-of-line blocking. We implemented this framework in Fast TCP and carried out a simulation-based comparison between Multistream Fast TCP (MFast TCP) and Stream Control Transmission Protocol (SCTP). Our results show that MFast TCP significantly improved performance over SCTP in the event of HoLB.

INDEX TERMS Data transfer, fast TCP, head-of-line blocking, multistream, SCTP, transport protocol.

I. INTRODUCTION

Latest broadband facility is now available to home users via modem and ADSL. It enables the recent applications to transfer a variety of data to the end user. Data intensive applications are using Transmission Control Protocol [1] to transfer large amounts of data. TCP is the default choice of these applications due to its unique features like congestion control, flow control, reliable data transfer, in-order data delivery, and data transmission over a single bytestream. The studies [2]–[4] proposed different schemes for multimedia applications to optimize their performance on top of TCP. These studies achieved the desired goals at the application level, but some issues remained that must be addressed at the transport layer to get maximum performance benefit.

TCP follows a systematic approach for in-order data transmission. The TCP sender assigns a unique sequence number to each segment to maintain data order before dispatching it to the receiver. These segments may be lost or corrupted either due to congestion, bit errors in a bad wireless spectrum, or a bad fibre connection. This data loss disturbs the segments' order on the receiver side. In this case, the receiver places these segments in the receiver buffer to restore the order of the segments as per assigned sequence number. Although this data storage is necessary to maintain data order, it causes delay in the delivery of segments to the application [5]. For example, a server sends three different pictures to appear in parallel in a client's web browser. The server simultaneously transmits part of the data from each picture to the transport layer. This transmission process at the server continues until data of all three pictures are transferred successfully. During transmission, if a packet loss occurs in TCP and the packet contains part of the first picture's data, then the receiver will stop processing all the latter segments containing data for the other two pictures until the lost segment is recovered. This phenomenon is known as head-of-line blocking (HoLB). This causes unnecessary delay in the data delivery for the second and third picture.

The associate editor coordinating the review of this manuscript and approving it for publication was Giacomo Verticale.

Latency at the applications level increases due to HoLB. The applications which are providing online services require maximum low latency for swift response to the end user. The user experience is negatively affected with large latency response. 100 ms latency costs 1 percent of sales to Amazon and 20 percent of google traffic reduction as it takes 0.5 seconds of extra time in search page generation [6]. Therefore, these applications required an efficient transport protocol to increase the efficiency.

HTTP manages client's requests at the application layer. It has to depend upon the underlying TCP to manage data exchange in a client-server networking model. In the case of multiple requests, as in the case discussed above, it suffers head-of-line blocking due to the limitation of the TCP. Therefore, the different versions of HTTP use underlying TCP differently to avoid head-of-line blocking [7]–[12]. But none of the HTTP versions could resolve HoLB problem completely.

HTTP/2 partially resolved the problem of HoLB by introducing multiplexing at the application layer so that the application could issue new requests over the same connection without having to wait for the previous ones to complete. HTTP/2, however, still suffers from another kind of HoLB on the TCP layer, i.e., one lost packet in the TCP stream that places all HTTP's streams on hold until the lost packet is retransmitted and received. To permit this type of parallel functionality on top of the TCP, it is required that the TCP is able to identify the independent messages of the application and multiplex these messages over the transport layer. Similarly, it is required that the TCP on the receiver end could segregate the segments of independent messages. These changes will make the TCP compatible with HTTP/2 and eliminate HoLB at the transport layer. In this study, an architecture is proposed for TCP to incorporate the above-mentioned changes.

Stream Control Transmission Protocol (SCTP) is designed for telephony signaling over IP networks. It provides multistream feature to address the head-of-line blocking problem at the transport layer. Although it efficiently handles small flows of signaling messages, its loss-based congestion control algorithm degrades its performance in high bandwidth-delay product networks [22].

The framework is implemented using Fast TCP [13]. Fast TCP is a delay-based transport protocol. It is specially designed for high speed, long distance, and large bandwidth networks. It avoids packet loss during the congestion avoidance phase. Whereas, loss-based transport protocol like SCTP cannot avoid packet loss in the congestion avoidance phase. This packet loss oscillates the congestion window of loss-based protocol. This oscillation degrades its performance. The additive increase approach increases the size of the congestion window by one packet on each RTT after fast recovery phase. Therefore, it takes more time to utilize the available bandwidth of the network after packet loss compared to the delay-based congestion control scheme. Fast TCP uses a formula instead of the AIMD technique to increase its congestion window after packet loss. It performs better in the form of goodput in the fast retransmit phase. Furthermore, it consumes the available bandwidth more rapidly after fast recovery phase; therefore, it gets more advantage from the multistream feature compare to the loss-based protocols. The higher goodput of MFast TCP during HOL blocking period proves it.

We made a simulation-based comparison between Multistream Fast TCP (MFast TCP) and Stream Control Transmission Protocol (SCTP) [14]. Our results show that when HoLB occurs, the MFast TCP performs better compared to the SCTP. Furthermore, the size of the receiver buffer is critical for the transport protocols that are used in the high bandwidth-delay product network. In the presence of multiple independent messages/requests, it is observed that MFast TCP consumes the receiver buffer far less compared to native Fast TCP in the case of HoLB.

The major objectives of designing the proposed framework for TCP are: (i) eliminate unnecessary delay in transmission of the data from the transport layer to the application layer (ii) design an architecture to make TCP compatible with HTTP/2 so that its independent streams could be handled (iii) devise a procedure to control each independent stream separately within a single connection of TCP.

The rest of the paper is organized as follows: Section II describes the related work performed to resolve the HoLB problem; Section III explains the architecture of the proposed framework; Section IV provides implementation detail of the proposed framework; Section V narrates simulation setup, results, and discussion; Section VI is dedicated to the conclusion.

II. RELATED WORK

HTTP, as an application layer protocol, adopted different techniques to handle the head-of-line blocking problem. HTTP/1.0 uses separate TCP connections for each request. Although, these connections serialize the data transmission, there is a cost involved in setting up a new connection. Therefore, it increases the latency because of establishing a TCP connection, the number of network packets, and the resource requirement on the server [7]–[9]. Furthermore, browsers don't allow an unlimited number of TCP connections. In the case of multiple TCP connections, time to trigger fast retransmit is also increased. This increased time is directly proportional to the number of connections [15]. HTTP/1.1 resolved the problems faced by HTTP/1.0 and proposed to open limited persistent TCP connections. It reduces overhead like number of network packets, resource requirement on the server, and TCP connection establishment over multiple requests. It increases network utilization and effective bandwidth [10]. Both of these versions of HTTP are unable to achieve requests-concurrency over a single TCP connection. The most recent version of HTTP is HTTP/2.0 that was introduced in May 2015. It addresses the problems of HTTP/1.1 and provides new features like content priority and server push. Considering its performance and

other benefits, current websites that are using HTTP/1.1 are quickly adopting HTTP/2.0 [16], [17]. HTTP/2.0 proposed a technique to achieve requests-concurrency at the application layer by using a single TCP connection at the transport layer. It multiplexes data of all the requests over a single TCP connection. Although, this technique eliminates the need for multiple TCP connections, it could not remove the head-of-line blocking problem at the transport layer [11], [12].

Google developed SPDY to transport web contents. SPDY multiplexes multiple requests over one TCP connection. If the requests have a small amount of data, then SPDY gets a benefit over HTTP/1.1. Whereas, its performance degrades in the case of packet loss [18], [19]. This reduced performance is because of HoLB.

QUIC multiplexes multiple independent requests of the application over UDP. It establishes connection in zero-RTT. It uses Forward Error Correction (FEC) to handle losses. QUIC performs better compared to SPDY [20], but still it has limitations. FEC increases the load over network. QUIC has to work on a cross layer to handle the application and transport layer functionalities. Although QUIC recovers from congestion very efficiently, its performance may still degrade due to the same reason as SPDY, that is, using a single multiplexing channel [19]. Furthermore, it uses UDP instead of TCP in its solution.

Stream Control Transmission Protocol (SCTP) is a general purpose transport protocol specifically designed for telephony signalling over an IP network [14], [21]. The protocol efficiently transfers small flows of signalling messages. It provides two important transport layer services, that is, multihoming and multistreaming. Although its multistream feature addresses the problem of head-of-line blocking, its loss-based congestion control algorithm degrades its performance over long distance, large bandwidth networks. SCTP is not designed for large bandwidth delay product networks. It suffers performance problems while transferring large data files over this network [22].

III. SYSTEM ARCHITECTURE OF THE PROPOSED FRAMEWORK

Fig. 1 shows the system architecture of the proposed framework having a Data Distributor (DD), Data Receiver (DR), Stream Manager (SM), Data Manager (DM), Data Acknowledgment (DA), Streams' Dispatch Buffers, and Streams' Receiver Buffers.

The Data Distributor (DD) unit receives the application's data, extracts stream identifiers from the application frame, and uses this information to place the data in the relevant stream's dispatch buffer. The Stream Manager (SM) is responsible to create buffers on the sender and receiver side for each independent message/request of the application and it also releases the resources at the completion of data transfer. Data Receiver (DR) extracts data from the streams' receiver buffers and places it in the buffer. Data from that buffer is then pushed to the application layer. The Data Manager (DM)

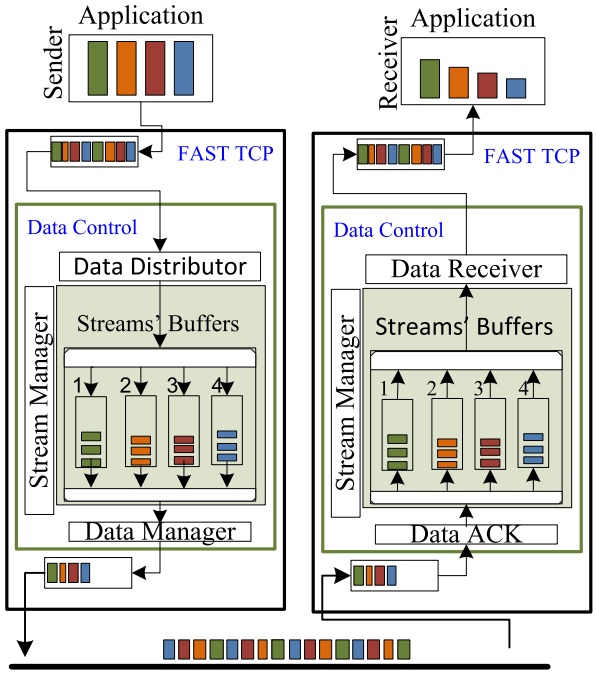


FIGURE 1. System architecture of the proposed framework.

selects data from the streams' dispatch buffers by using a specific data selection algorithm, places a stream header on the segment, and multiplexes the segments for transmission. This unit is also responsible for control of each stream flow. The Data Acknowledgment (DA) unit acknowledges each received segment on the receiver side and coordinates with the Data Manager (DM) on the sender side to identify any lost segments so that the Data Manager could adjust its data selection accordingly.

IV. IMPLEMENTATION DETAIL

The technical details of the framework are discussed in the following sections.

A. TCP HEADER

One flag bit labeled Multistream Enabled (MSE) is introduced in a reserved area of the TCP header [1] as shown in Fig. 2. It indicates that the protocol supports multiple streams to make it compatible with HTTP/2.

MSE bit is set to inform the receiver that the sender is using multistream enabled protocol. The receiver also sets MSE bit in the response segment to show that it also supports multiple streams. This information is exchanged during handshake. If both sender and receiver support multiple streams, then subsequent communication takes place using this facility; otherwise native TCP protocol procedure is followed for data communication. While processing the MSE TCP segment, the host extracts a Stream Header (SH) of four bytes from the TCP header. This SH provides further details about the stream.

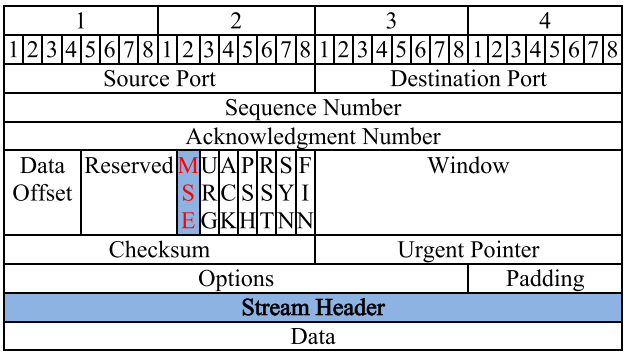


FIGURE 2. MFast TCP Header.

B. STREAM HEADER

The stream header consists of 32 bits as shown in Fig. 3. First, 8 bits are reserved for the stream id (SID) and the remaining 24 bits are reserved for the stream sequence number (SSN). The SID is used to uniquely identify the streams. The SSN is a number that is assigned in a sequence to the segments of a stream. One TCP segment contains data of one application's message up to the maximum segment size (MSS). If the application's independent message does not fit in one TCP segment, then it can be fragmented into multiple segments.

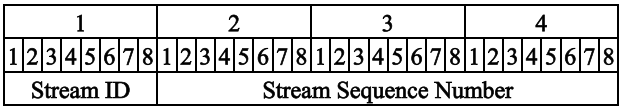


FIGURE 3. Stream Header.

SCTP has two types of headers that are common header of 12 bytes and chunk headers of 8 bytes. SCTP has 255 chunk types including defined and reserved chunk types. The header size of each chunk type varies. The data chunk of SCTP has a 16-byte header size. The protocol uses it to distinguish each stream and its sequence number. SCTP can bundle multiple data chunks in a single segment. In the case of segment loss, multiple data chunks are lost, and corresponding streams are blocked. Whereas, MFast TCP provides the feature of multistream with a stream header size of 4 bytes coupled with the TCP header of 20 bytes. The protocol places one stream's data in one segment and only that particular stream is blocked in case of segment loss.

The proposed scheme adds four bytes of stream header to the 20-byte TCP header. Therefore, the overhead in a network of the typical 1000 bytes packet size is 0.42%.

C. TCP CONNECTION

To explain the connection establishment procedure, two hosts, A and B, are shown in the Fig. 4 with "Closed" and "Listen" states respectively.

The TCP on host A initiates a "three-way handshake" procedure to establish the connection with host B. At the time of connection establishment, the sender raises its SYN and MSE control flags. The sender raises the MSE control flag

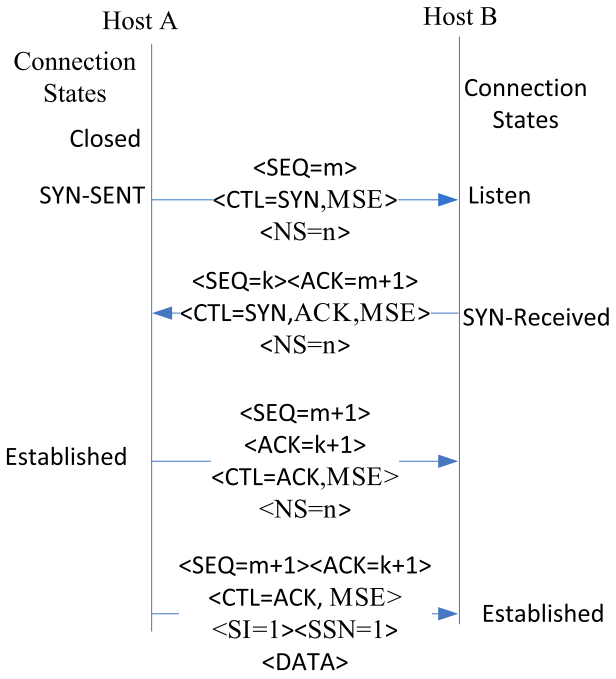


FIGURE 4. Three-way handshake procedure at the time of connection establishment.

to inform the end host that it supports multiple streams. The sender also places the information about how many streams are required to be opened with the end hosts in the SYN segment. Host A changes its connection state from Closed to SYN-SENT after sending the SYN segment.

If Host B does not support multiple streams, then it ignores the MSE control flag and responds as a normal TCP. Being a multiple-stream enabled TCP, Host B (which is in Listen state) on receipt of the SYN segment, allocates memory space required for a number of streams. Host B responds to the SYN segment with a SYN-ACK segment having SYN, ACK, and MSE control flags with an approved number of streams. The SYN-ACK segment acknowledges the SYN segment and informs the initiator that requested space is allotted. Host B changes its connection state from Listen to SYN-RECEIVED after dispatching the SYN-ACK segment.

On receipt of SYN-ACK segment, Host A changes its connection state from SYN-SENT to ESTABLISHED and sends a ACK segment with the control flags ACK and MSE. Host B, on receipt of the ACK segment, sets its connection state from SYN-RECEIVED to ESTABLISHED.

Once the connection is established, full duplex flows are open for data transmission between the end hosts. Now, end hosts can handle multiple streams within a single connection. This single connection reduces overhead of multiple TCP connections, provides better goodput during congestion, and loss recovery. Multiple independent streams within a single connection reduce latency between the transport and application layers at the receiver side.

D. DATA SENDING AND ACKNOWLEDGEMENT

The sender transfers each independent message of the application over the separate stream. It segments the data of the application's message and places a stream header over it. The TCP that supports multiple streams keeps record of the streams' segments against the Segment Sequence Number. The sender keeps this record of each stream until the acknowledgment of these segments is received.

SID and SSN are used to reassemble data segments in the correct order at the receiver side. The receiver stores the received segments in a receiver buffer. In response to this received segment, the receiver prepares an ACK segment and includes information of the corresponding stream's received segment in it. The receiver extracts in-order streams' segments from the receiver buffer and stores them in the corresponding stream buffer. The data from the streams' buffers is pushed to the application layer.

On receipt of segment acknowledgement, the sender uses ACK segment information to match it with Segment Sequence Number, SID, and SSN which was saved at its transfer time, so that ACK segments could be removed from the buffers. SID and SSN are two different values, so unacknowledged segments of one stream do not affect the data transmission of other streams.

1) EVENTS THAT OCCURRED ON SENDER SIDE

There are various events which are triggered on the sender side while providing the data transfer service to the application. These events are Data received from the application, Timeout, and Acknowledgment received.

On the sender side, at the time of data receipt, transport protocol places data of the application's message in its assigned buffer. In case a timeout event is triggered, the sender retransmits the timed-out segment and retransmits the segments whose Sequence numbers are greater than the sequence number of this timed-out segment. On receipt of Ack Segment sender checks whether it a duplicate acknowledgment; if so, then it increases the duplicate Ack counter. If the count of the duplicate Ack is three, then it resets the duplicate Ack counter and triggers retransmission of the segment whose duplicate Ack is received. The sender extracts sack options from the Ack segment to mark the sent segments as sacked received.

2) FLOW CONTROL AND ACKNOWLEDGEMENT GENERATION ON THE RECEIVER SIDE

Flow control is another important service that TCP provides. Flow control ensures that the sender is not overwhelming the receiver device with large amounts of data. The Ack segment provides very useful information to the sender that is necessary for this service. It informs the sender that the receiver has received the segment, the segment has left the network, and what is the current status of its receiver window.

In the receiver buffer (represented by the receiver window), it keeps only those streams' segments whose data is not received in-order. Whereas those streams' segments that are received in-order are removed from the receiver buffer. In this way, MFast TCP frees more space of the receiver buffer. The benefit of this technique is significant during the HoLB period.

There are three types of cases as given below that are handled by the receiver while generating the ACK segment:

1. On arrival of an in-order segment with an expected segment sequence number whereas all previous data up to that segment has been already acknowledged.
2. On arrival of an in-order segment with an expected segment sequence number whereas higher segments have already been acknowledged.
3. Arrival of an out-of-order segment with a higher than expected segment sequence number

In the case of a normal scenario, that is, the receiver receives the segment with expected sequence number, whereas all previous data up to that segment has been already acknowledged. In this case, there is no change in the segment handling algorithm except that it places the segment in its related stream buffer after extracting required information from the segment and stream header. From the stream buffer, it is pushed to the application.

In the case where a receiver receives an in-order segment with an expected segment sequence number whereas higher segments have already been acknowledged. This is the scenario which represents that a particular stream's data that was blocked in the receiver buffer becomes in-order. The receiver removes all the data from the receiver buffer and places it in the stream's receiver buffer because a lost segment has been received and now the stream's data is in-order and ready to be handed over to the application.

In the case where a receiver receives the out-of-order segment with a segment sequence number higher than the expected segment sequence number. In this scenario, if the received segment contains the data of the stream, all of whose previous segments are received correctly and there is no gap between the segments, then it will be extracted from the receiver buffer and placed in the corresponding stream buffer. This step frees the receiver buffer space and on the same time made this data available to the application. In this way, the protocol alleviates the HoLB problem and efficiently utilizes the receiver buffer. This efficient utilization of the receiver buffer is critical for a delay-based congestion algorithm that is being operated in large bandwidth-delay product networks.

V. SIMULATION SETUP, RESULTS, AND DISCUSSION

The proposed framework as described in the previous section is simulated using ns-2.35 [23] in a wired environment. For simulation, we used the Fast TCP patch that is implemented by L. Andrew [24] and the SCTP patch that exists in the ns-2.35 release. A custom application capable of tagging data segments for the transport layer is used to generate traffic. It is considered that a server needs to transfer multiple files. Therefore, the application uses a separate stream of a fixed data size for each of its independent messages over the period of the TCP session.

Procedure 1 Acknowledgement Generation

Variables:
Segment Sequence Number 'SEQ'
Expected Segment Sequence Number 'ESEQ'
Stream ID 'SID'
Stream Sequence Number 'SSN'
Expected Stream Sequence Number 'ESSN'

Procedure ACK(TCP Header tcpRec)
SEQ ← tcpRec.SEQ
SID ← tcpRec.SID
SSN ← tcpRec.SSN
TCP Header ack;
ack.SID ← SID # ACK Segment
do if SSN = ESSN of SID **then**
  case 1: ESSN of this Stream is received and there is no higher segments of this stream is in the rwnd
    ESSN of SID ← ESSN of SID + 1
    ack.SSN ← ESSN of SID
    Remove Segment from rwnd
    StreamBuffer[SID] ← tcpRec.Stream Data
  case 2: ESSN received and higher segments of this stream are
    there in the rwnd
    ESSN of SID ← cumulated ACKED SSN of SID
    ack.SSN ← ESSN of SID
    # Place stream segments in StreamBuffer
    StreamBuffer[SID] ← SSN of SID TO ESSN of SID
    Remove SSN of SID TO ESSN of SID from rwnd
end if
do if SSN > ESSN of SID **then**
  rwnd ← tcpRec.Stream Data
  ack.SSN ← ESSN of SID
end if
do if SEQ = ESEQ **then**
  case 1: Expected segment received and there is no higher segment in rwnd
    ESEQ ← ESEQ + 1
    ack.ACK_Number ← ESEQ
  case 2: Expected segment received and higher segments are there in the window
    ESEQ ← cumulated ACKED SN
    ack.ACK_Number ← ESEQ
end if
do if SEQ > ESEQ **then** # Out of order arrival of the segment
  ack.ACK_Number ← ESEQ
  set SACK option for this segment
end if
Dispatch ACK Segment

The application generates traffic in such a way that data from different sources is multiplexed, segment-by-segment. For example, in the case of FTP over Fast TCP with the proposed framework, the transfer of multiple files is started simultaneously and data equal to the size of the segment is sent in a round robin fashion from each source file.

The data segments generated by the application carry an ID, size, and sequence number of the stream/transaction. The transport layer manages the session behaviour, as required with this information.

This mechanism provides significant performance improvements in congested or vulnerable networks or where the chances of packet loss at intermediate nodes exist.

A. NETWORK TOPOLOGY

The network topology used in the simulation scenarios is shown in Fig. 5. It consists of four nodes, i.e., n0, n1, n2, and n3. Duplex links n0-n1, n1-n2, and n2-n3 have bandwidth of 2Mbps, 1.5Mbps, 2Mbps and delay of 10ms, 100ms, and 10ms respectively. Each node uses a DropTail queue of size 10. There are three pairs of agents (SctpAgent, SctpAgent), (FastTcpAgent, TcpSink/Sack1) and (MSFastTcpAgent, TcpSink/Sack1). These agents are attached with n3 and n0 separately to study the behaviour of these protocols while executing the simulation scenarios. A custom ServerApp and ClientApp are attached with the sender and receiver agents. The simulation time is 100 s, ServerApp is set to start at 1.0 s and stop at 100 s. List error model is attached with the bottleneck link n1-n2 to drop a packet so that head-of-line blocking on the receiver end could be created and its impact on goodput during this interval could be studied.

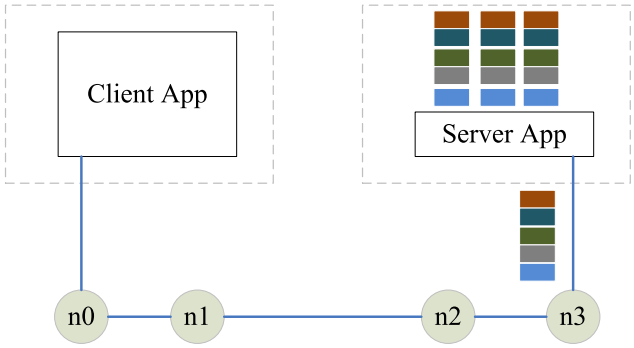


FIGURE 5. Network topology used in the simulation scenarios.

B. CONFIGURATION

We performed two simulation tests to carefully analyse and evaluate the behaviour of these transport layer protocols. In the first test, the performance of native delay-based Fast TCP and loss-based Stream Control Transmission Protocol (SCTP) is compared. Fast TCP controls its queue usage with the parameter alpha. Simulation of Fast TCP is executed twice with the value 5 and 8 of the parameter alpha. SCTP fills the bottleneck queue very quickly whereas Fast TCP aims for minimum utilization of queues during the transfer of data. We used two values of parameter alpha so that it could utilize the bottleneck queue moderately and aggressively and study its impact on the performance of Fast TCP in comparison with SCTP.

In the second test, an error-model is introduced to drop a packet. HoLB occurs on the receiver side with this packet drop. The performance of Fast TCP, SCTP, and MFast TCP in HoLB is studied. Table-1 shows various parameters related to the simulation scenarios.

TABLE 1. Simulation parameters.

Parameters	Value
Simulator	NS 2.35
Simulation Time	100 s
Transport Protocols	SCTP, Fast TCP, MFast TCP
Packet Size	1000 bytes (Header size included)
Error Model	List Error Model to drop single packet
Number of Streams	5
The values of Fast TCP's parameter alpha and beta in scenario 1	Alpha = 5, beta = 5 (Initial values assigned at the start of the simulation)
The values of Fast TCP's parameter alpha and beta in scenario 2	Alpha = 8, beta = 2 (Initial values assigned at the start of the simulation)

C. ENVIRONMENT AND ASSUMPTION

Packet received means the packets received at the transport layer of the receiver, and packet delivered means the packet from the transport layer was handed over to the application.

Packets are delivered in an order with respect to the stream sequence number (SSN) instead of the TCP sequence number (SEQ). SEQ is only used to track the received and other TCP session related functionalities. The Fast TCP functionality is modified to ensure that, as soon as the stream packets are in-order, data will be transferred to the application. If packets are missing, the flow will be held and control will be transferred to the next stream.

The algorithm simulated here is for 1 to 5 streams. SCTP and Fast TCP use a selective acknowledgement (SACK) procedure. It works on TSN and Sequence Number of SCTP and Fast TCP respectively.

D. EVALUATION METHOD

We desire to calculate goodput and see the impact of packet loss on the performance of the protocols. Therefore, in the simulation, a simple packet drop model is used to study network behaviour where one or several packets are lost either due to congestion, bit errors in a bad wireless spectrum, or a bad fibre connection on the destination node. In this approach, a packet is completely removed from the system; so the receiving node does not receive the specific packet. This packet creates a gap in the sequence number on the receiver side and data is received out of order. The receiver buffer stores all the preceding segments while waiting for the lost segment. Hence, this is a situation of head-of-line blocking. All the three protocols that are under the study enter in Fast Recovery mode. In this mode, native Fast TCP provides no goodput, whereas modified Fast TCP shows considerable performance improvement compared to native Fast TCP. In Fast Recovery mode, the proposed MFast TCP efficiently addresses the head-of-line blocking problem. Its performance is competitive with SCTP.

The analyses given in this paper shows improvements on the application layer by eliminating the impact of HoLB from streams that were not affected by the packet drop at the transport layer. Thus, the delay in data delivery to the application layer is eliminated.

E. SIMULATION RESULTS AND DISCUSSION

This section provides results obtained by using Fast TCP, SCTP, and the proposed Multistream Fast TCP (MFast TCP).

1) FAST TCP VS. SCTP

In this section, simulation results are obtained to compare the performance of Fast TCP and SCTP. SCTP rapidly increases its congestion window to fully utilize the network link. At the peak time, it fully consumes the bottleneck queue. Therefore, at this point, the number of SCTP packets in the network exceeds compared to Fast TCP. In order to make the fair comparison, we executed the simulation of Fast TCP twice, once with the value of alpha as 5 and a second time with the value of alpha as 8. This parameter controls the number of packets that Fast TCP can place in the bottleneck queue while transferring the data.

TABLE 2. Results of the three schemes (without error model) with different performance parameters.

Performance Parameters	SCTP	Fast TCP[1]	Fast TCP[2]
Average Throughput (Kbps)	1,096	1,423	1,463
Average end-2-end delay (sec)	0.148	0.135	0.156
Average Goodput (Kbps)	1,043	1,366	1,404
# of Packets Dropped	30	0	0

[1] Alpha = 5, beta = 5. [2] Alpha = 8, beta = 2.

Table 2 shows average throughput, average end-to-end delay, average goodput, and the number of packets dropped of SCTP and Fast TCP with the buffer size of 10 packets. SCTP provided lesser average throughput compared to Fast TCP during the simulation time of 99 seconds. This degradation is because SCTP is not designed for long distance and high bandwidth networks. Furthermore, its loss-based congestion control algorithm dropped 30 packets during the simulation. SCTP decreases its congestion window on each packet loss.

Therefore, its average throughput is less compared to Fast TCP which uses a delay-based congestion algorithm.

Average throughput of Fast TCP (alpha: 5) and Fast TCP (alpha: 8) is 1423 Kbps and 1463 Kbps, respectively. Throughput of Fast TCP is increased with the increase of the alpha value. This is because Fast TCP (alpha: 8) can place more packets in the bottleneck buffer. Therefore, its average throughput is higher compared to Fast TCP (alpha: 5). The delay-based algorithm of Fast TCP dropped no packets during the simulation period. Therefore, with the consistent congestion window size, its average throughput is higher compared to SCTP.

Average goodput of SCTP is 1043 Kbps whereas goodput of Fast TCP (alpha: 5) and Fast TCP (alpha: 8) is 1366 Kbps and 1404 Kbps respectively. Packet loss causes performance degradation for SCTP. Although its multistream feature removes the HoLB problem, its loss-based congestion control algorithm becomes the reason for the degradation of its performance.

It is quite clear from the statistics in Table 2 that Fast TCP has a performance edge over SCTP. Therefore, implementation of a multistream feature using Fast TCP will further improve its performance.

2) THROUGHPUT

Fig. 6 shows throughput of Fast TCP, MFast TCP, and SCTP. There is not much variation in throughput of Fast TCP and MFast TCP during the simulation except during the fast retransmit period. This is because of its delay-based algorithm, whereas a lot of variation in throughput of SCTP can be seen in the figure. This is because of its loss-based congestion control algorithm. Furthermore, there is no difference in the congestion window size of Fast TCP and Mfast TCP in slow start, congestion avoidance, fast retransmit, and fast recovery phases. Therefore, both of these protocols provide the same throughput.

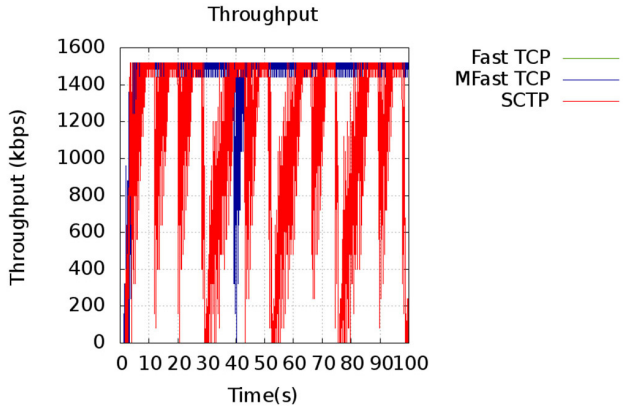


FIGURE 6. Fast TCP, MFast TCP, and SCTP throughput.

3) GOODPUT OBTAINED USING FAST TCP, MFAST TCP, AND SCTP

Fig. 7 shows the goodput of Fast TCP, MFast TCP, and SCTP. On the x-axis, time is shown and on the y-axis, goodput in Kb is shown. Goodput is calculated during the periods of 0.10-second intervals. HoLB stops the data delivery to the application until the receipt of the lost packet.

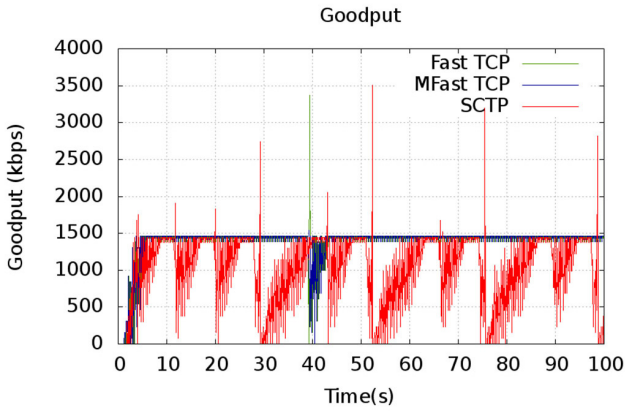


FIGURE 7. Fast TCP, MFast TCP, and SCTP Goodput.

Fast TCP does not have a multistream feature, therefore, it completely stops the data delivery to the application layer during the period of HoLB. On receipt of a lost packet, it delivers all the data stored in the receiver buffer. It is observed that Fast TCP extensively utilizes the receiver buffer in case of HoLB.

SCTP provides a multistream feature to resolve the problem of HoLB. It stops the data delivery of only that stream whose packet is lost, whereas rest of the streams continues to provide the data to the application layer.

MFast TCP does not use a receiver buffer so extensively compared to Fast TCP; rather, it stops only that stream whose packet is lost and continues to provide to the application from the rest of the streams. Fast TCP and MFast TCP, due to its delay-based congestion control algorithm, maintains its goodput during the simulation period. SCTP's goodput fluctuates due to its loss-based congestion control algorithm. Therefore, cumulated goodput of delay-based congestion control algorithm in long distance high bandwidth network is higher than a loss-based congestion control algorithm protocol. Furthermore, average goodput of MFast TCP is higher than SCTP.

4) GOODPUT DURING AN INTERVAL OF PACKET LOSS

The behavior of Fast TCP, MFast TCP, and SCTP with the traffic of multiple messages is shown in Fig. 8. This is a zoom-in view extracted from Fig. 7 to show the behavior of protocols at the time of packet loss. For the purpose of clarity, the goodputs of all the protocols during packet loss are overlapped within the same time interval.

In SCTP, a packet is dropped at 38.72 second, the sender receives a third duplicate ack at 39.12 second, retransmits the dropped packet, and reduces its congestion window by 1/2 and enters in fast recovery mode. In fast recovery mode, although SCTP is in HoL state, its goodput does not drop to zero. This is the case because SCTP is continuously providing

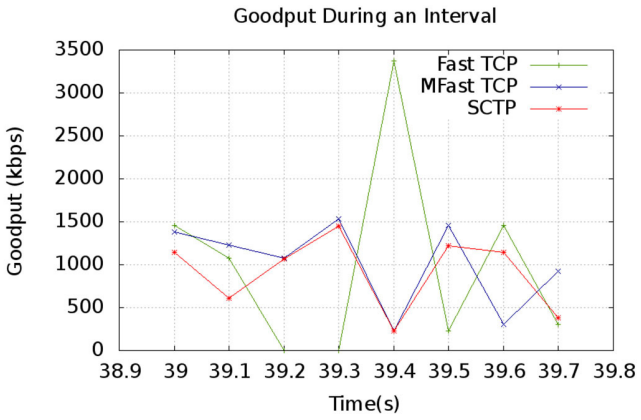


FIGURE 8. Goodput graph during the interval of packet drop.

data to the application from the rest of the streams. Goodput of SCTP during the HoLB period is less than MFast TCP. The receiver receives the retransmitted packet at 39.25 second. This packet made the data in-order and the SCTP hands over the data of blocked stream to the application layer. Therefore, the graph is showing increase in goodput at this instance of time. Sender receives cumulated ack at 39.38 second. Decrease in goodput at this moment is because most of the data has already left the network and has been handed over to the application. Therefore, the transport layer has no more data to deliver to the application layer. Moreover, with this ack, the SCTP comes out of its fast recovery mode and starts sending new packets. The receiver receives these transmitted packets during the 39.5-39.6 seconds interval. SCTP starts increasing its congestion window using an Additive increase technique to consume available bandwidth after fast recovery mode. Interpacket delay between two series of packets becomes the reason for fluctuation in goodput in the graph. The SCTP provides average goodput 964.6 Kbps during the interval of the given graph.

In MFast TCP, a packet is dropped at 38.93 second. The sender receives a third duplicate ack at 39.20 second, retransmits the dropped packet, and reduces its congestion window by 1/2. The receiver receives the retransmitted packet at 39.35 second. The sender receives the cumulated ack at 39.48 second. Meanwhile, it is observed that the receiver receives a few inflight packets and hands them over to the application at 39.5 seconds. The sender comes out of fast recovery mode at 39.48 seconds. The cumulated ack quickly slides its congestion window. It sends new packets to the receiver and waits for their acknowledgement before sending further packets. The receiver receives these transmitted packets during the interval 39.6 – 39.7. Fast TCP and MSFast TCP gradually increase its congestion window based on the RTT calculation to stabilize its congestion window. This is why we observed a sawtooth-shaped curve of their congestion window. Average goodput of the MFast TCP is 1080.8 Kbps during the interval of the given graph. The MFast TCP shows 12% better performance compared to the SCTP during the interval of the graph. Furthermore, during the HoLB period, the MFast TCP produces 31% higher goodput compare to the SCTP.

In the Fast TCP, a packet is dropped at 38.93 second. HoLB occurs on the receiver end at 39.07. The application layer receives no data from the transport layer from 39.1 – 39.35 seconds. The sender receives its third duplicate ack at 39.20 second, retransmits the dropped packet, and reduces its congestion window by 1/2. The receiver receives the retransmitted packet at 39.35 second. The receiver hands over all the blocked packets to the application layer on receipt of the dropped packet because HoLB is clear now. The sender receives the cumulated ack at 39.48 seconds. Meanwhile, it is observed that the receiver receives a few inflight packets and hands them over to the application at 39.5 seconds. The sender comes out of fast retransmit mode at 39.48 seconds. Cumulated ack clears its congestion window. It sends new packets to the receiver and waits for their acknowledgement before sending further packets. The receiver receives these transmitted packets during the interval 39.6 – 39.7. Average goodput of the Fast TCP is 1085.4 Kbps during this interval.

5) GOODPUT OF INDIVIDUAL STREAMS OF MFAST TCP

The impact of the HoLB is the reduction of data transfer to the application during the period. The MFast TCP assumes that, in the case of minor single drops or bit errors, the packet drop may not be from all current streams. Rather, it may be from one or two streams, or in the worst case, packets are dropped aggressively and all the streams' packets are dropped. Fig. 9 shows the impact of a single packet drop of selected stream-2 and its impact in case of MFast TCP. It does not block all streams but identifies the related stream whose packet is missed or lost and places only that stream on hold. Furthermore, it checks if the in-transit packets arrived in the buffer or not. If there are the packets related to non-affected streams, it selects the next, non-affected stream and transfers the in-sequence data to the application. If the MFast TCP cannot find stream data in-order, it places the corresponding stream on hold unless all related packets arrive and an

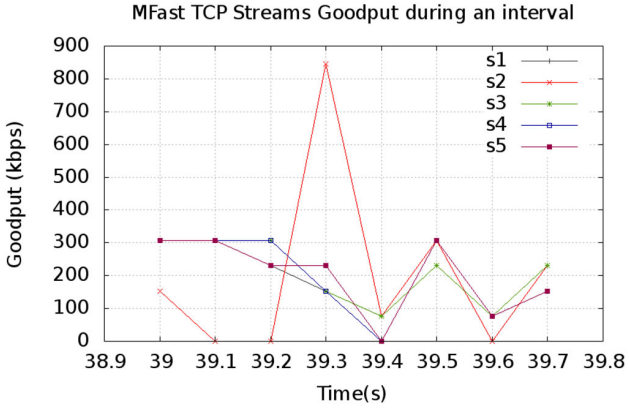


FIGURE 9. MFast TCP Goodput during an interval of packet drop.

in-order transmission can be made. Fig 9 shows that stream number 2 (red) is blocked effectively until retransmission of the dropped packet returns. However, the remaining streams continue to transfer to the application. MFast TCP ensures the relevant stream is only blocked if there is packet loss and other streams continue to transfer their data to the application. In contrast, in the case of traditional Fast TCP, all streams are placed on hold until the retransmission returns.

One of the basic drivers of implementing MFast TCP was the size of packets in transit, the larger the RTT, the larger the volume of the packets in transit and the more possibilities of buffer overflows in the case of HoLB. Therefore, networks having large bandwidth or large RTT can benefit more from MFast TCP compared to networks with small bandwidth.

6) GOODPUT OF INDIVIDUAL STREAMS OF SCTP

Goodput of each independent stream of SCTP is shown in Fig. 10. Figure 10 shows that a packet of stream-5 is dropped as HoLB scenario can be seen during the interval 39 – 39.1. All the streams, except stream-5, continuously provide data to the application during the HoLB period.

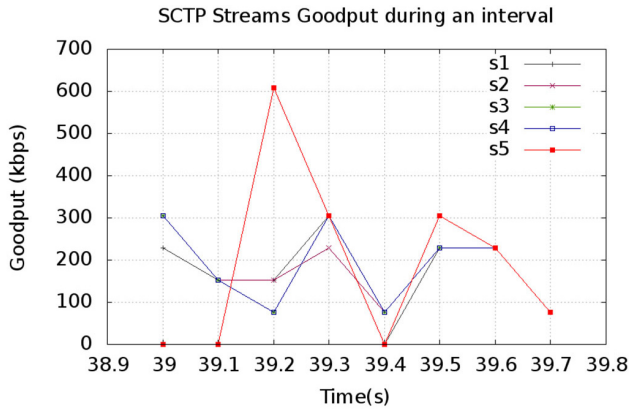


FIGURE 10. SCTP Goodput during an interval of packet drop.

VI. CONCLUSION

In this paper, we discuss the design and implementation of a framework to introduce multistream feature in Fast TCP. We study the performance of MFast TCP in comparison with Fast TCP and SCTP. Our results show that MFast TCP with the proposed framework provides the compatibility with the HTTP/2 at the application layer and reduces data delivery latency between transport and application layers. The stream ID and stream sequence number in the TCP header is introduced. The SID segregates multiple independent messages and the SSN is used to maintain the order of segments of the particular SID. In the case of packet loss, both loss-based and delay-based TCP variants enter in fast retransmit mode. Results shows that MFast TCP minimizes the impact of HoLB in the multistream environment and provides significant enhancements to effectively increase the data delivery rates to applications. It also increases the transport layer efficiency by minimizing the blocked period and the buffer growth.

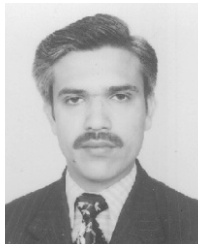
The novelty of the framework is that it provides the multistream feature with a 24-byte header size where the SCTP provides the same feature with a 28-byte header. The SCTP places multiple chunks in a single segment based on its chunk and segment size. In the case of loss, streams whose chunks were lost during the transmission are blocked, whereas proposed framework places only one stream's data in one segment at a time. Therefore, in the case of segment loss, only the corresponding stream is blocked, but the rest of the streams continuously provide data to the application. Above all, this framework enables the application layer to transfer its multiple independent messages using a single TCP connection without any fear of HoLB.

As we continue the optimization of our proposed framework, we plan to extend our study to see the benefits that can be obtained by reducing the time period to utilize available bandwidth of the network link after the fast recovery phase with the perspective of multistream feature in TCP. The SCTP and MFast TCP both use fast retransmit and fast recovery mechanisms after the packet loss. We believe that fast convergence after the fast recovery phase will help to increase the goodput and throughput of multistream-aware transport protocol more appropriately, and it will impact on the end-to-end performance of the protocols. Stream priority scheme is another future topic of this research. This scheme is required to enable the application to prioritize delay-sensitive data over delay-tolerant data.

REFERENCES

[1] J. Postel, *Transmission Control Protocol*, document RFC-793, IETF, 1981.
[2] J. Wu, C. Yuen, M. Wang, J. Chen, and C. W. Chen, "TCP-oriented raptor coding for high-frame-rate video transmission over wireless networks," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 8, pp. 2231–2246, Aug. 2016.
[3] J. Wu, C. Yuen, and J. Chen, "Leveraging the delay-friendliness of TCP with FEC coding in real-time video communication," *IEEE Trans. Commun.*, vol. 63, no. 10, pp. 3584–3599, Oct. 2015.
[4] J. Wu, B. Cheng, M. Wang, and J. Chen, "Priority-aware FEC coding for high-definition mobile video delivery using TCP," *IEEE Trans. Mobile Comput.*, vol. 16, no. 4, pp. 1090–1106, Apr. 2017.
[5] F. R. Kevin and W. R. Stevens, "TCP: The transmission control protocol (Preliminaries)," in *TCP/IP Illustrated: The Protocols*, vol. 1, 2nd ed. Reading, MA, USA: Addison-Wesley, 2011, pp. 586–587.
[6] J. Zhang, F. Ren, and C. Lin, "Survey on transport control in data center networks," *IEEE Netw.*, vol. 27, no. 4, pp. 22–26, Jul./Aug. 2013.
[7] J. C. Mogul, "The case for persistent-connection HTTP," in *Proc. ACM SIGCOMM Symp.*, Stockholm, Sweden, 1995, pp. 299–313
[8] V. N. Padmanabhan and J. C. Mogul, "Improving HTTP latency," *Comput. Netw. ISDN Syst.*, vol. 28, nos. 1–2, pp. 25–35, 1995.
[9] E. Casilari, F. J. Gonzblez, and F. Sandoval, "Modeling of HTTP traffic," *IEEE Commun. Lett.*, vol. 5, no. 6, pp. 272–274, Jun. 2001.
[10] R. Corbel, E. Stephan, and N. Omnes, "HTTP/1.1 pipelining vs HTTP2 in-the-clear: Performance comparison," in *Proc. 13th Int. Conf. New Technol. Distrib. Syst. (NOTERE)*, Jul. 2016, pp. 1–6.
[11] H. de Saxcé, I. Oprescu, and Y. Chen, "Is HTTP/2 really faster than HTTP/1.1?" in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Hong Kong, Apr./May 2015, pp. 293–299
[12] M. Belshe, M. Thomson, and R. Peon, *Hypertext Transfer Protocol Version 2*, document RFC-7540, IETF, 2015.
[13] 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.

[14] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson, *Stream Control Transmission Protocol (SCTP)*, document RFC-4960, IETF, 2007.
[15] M. Scharf and S. Kiesel, "NXG03-5: Head-of-line blocking in TCP and SCTP: Analysis and measurements," in *Proc. IEEE Globecom*, San Francisco, CA, USA, Nov./Dec. 2006, pp. 1–5.
[16] M. Varvello, K. Schomp, D. Naylor, J. Blackburn, A. Finamore, and K. Papagiannaki, "Is the Web HTTP/2 yet?" in *Proc. Int. Conf. Passive Active Netw. Meas.*, Heraklion, Greece, 2016, pp. 218–232
[17] T. Zimmermann, J. Rüth, B. Wolters, and O. Hohlfeld, "How HTTP/2 pushes the Web: An empirical study of HTTP/2 server push," in *Proc. IFIP Netw. Conf.*, Stockholm, Sweden, Jun. 2017, pp. 1–9
[18] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, "How speedy is SPDY?" in *Proc. 11th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, Seattle, WA, USA, 2014, pp. 387–399.
[19] F. Qian, V. Gopalakrishnan, E. Halepovic, S. Sen, and O. Spatscheck, "TM 3: Flexible transport-layer multi-pipe multiplexing middlebox without head-of-line blocking," in *Proc. 11th ACM Conf. Emerg. Netw. Exp. Technol.*, Berlin, Germany, 2015, Art. no. 3.
[20] C. Gaetano, L. De Cicco, and S. Mascolo, "HTTP over UDP: An experimental investigation of QUIC," in *Proc. 30th Annu. ACM Symp. Appl. Comput.*, Salamanca, Spain, 2015, pp. 609–614
[21] S. Randall, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson, *Stream Control Transmission Protocol*, document RFC-2960, IETF, 2000.
[22] M. J. Arshad and M. Saleem, "A simulation-based study of FAST TCP compared to SCTP: Towards multihoming implementation using FAST TCP," *J. Commun. Netw.*, vol. 12, no. 3, pp. 275–284, Jun. 2010.
[23] *The Network Simulator—NS-2*. Accessed: Feb. 11, 2019. [Online]. Available: https://www.isi.edu/nsnam/ns/ns-build.html
[24] L. Andrew. *FAST TCP Simulator Module for NS2*. Accessed: Mar. 5, 2006. [Online]. Available: http://www.cubinlab.ee.mu.oz.au/ns2fasttcp/



SARFRAZ AHMAD received the M.S. degree from the University of Management and Technology, Lahore, Pakistan. He is currently pursuing the Ph.D. degree in computer science with the University of Engineering and Technology (UET), Lahore. He is also an Assistant Professor with the Department of Computer Science, Virtual University of Pakistan. His research interests include computer networks and the Internet protocols.

MUHAMMAD JUNAID ARSHAD received the M.S. and Ph.D. degrees in computer science from the University of Engineering and Technology (UET), Lahore, Pakistan, in 2000 and 2009, respectively, where he is currently an Associate Professor with the Department of Computer Science and Engineering. His research interests include the Internet protocols, multihomed networks focusing on performance, security, and mobility issues, congestion control, and wireless networks.