Computing (2020) 102:2541–2563
https://doi.org/10.1007/s00607-020-00829-2

REGULAR PAPER

A BBR-based congestion control for delay-sensitive real-time applications

Sayed Najmuddin[1] · Muhammad Asim[1] · Kashif Munir[1] · Thar Baker[2] · Zehua Guo[3] · Rajiv Ranjan[4]

Received: 21 September 2019 / Accepted: 22 June 2020 / Published online: 6 July 2020

Abstract
The current User Datagram Protocol (UDP) causes unfairness and bufferbloats to delay sensitive applications due to the uncontrolled congestion and monopolization of available bandwidth.This causes call drops and frequent communication/connection loss in delay sensitive applications such as VoIP. We present a Responsive Control Protocol using Bottleneck Bandwidth and Round trip propagation time (RCP-BBR) as an alternate solution to UDP. RCP-BBR achieves low latency, high throughput, and low call drops ratio by efficiently customizing Transmission Control Protocol (TCP) Bottleneck Bandwidth and Round-trip propagation time (TCP-BBR) congestion control. We conducted comprehensive experiments, and the results show that proposed protocol achieves better throughput over UDP in stable networks. Moreover, in unstable and long-distanced networks, RCP-BBR achieved smaller queues in deep buffers and lower delays as compared to UDP, which performed poorly by keeping delays above the call drop threshold.

Keywords Congestion control · UDP · Delay sensitive application · VoIP

Mathematics Subject Classification 68 Computer science

1 Introduction

RCP-BBR (Responsive Control Protocol-Bottleneck Bandwidth and Round-trip) is a transport protocol that provides the efficiency of UDP (User Datagram Protocol) along with an enhanced congestion control mechanism. The protocol provides a balanced solution for delay sensitive applications which requires efficiency of UDP but do not want a protocol that is un-responsive to congestion. The proposed protocol is based on TCP-BBR (Transmission Control Protocol- Bottleneck Bandwidth and Round-trip) congestion control algorithm proposed by Google [1]. TCP-BBR monitors available bandwidth and minimum round trip time for the estimation of congestion whereas

Extended author information available on the last page of the article

traditional TCP algorithms uses packet loss as a measure of congestion. RCP-BBR follows the same approach for the detection of congestion however it provides a modified mechanism to handle the reliability of packet delivery. The modified mechanism follows a UDP like behavior to achieve better than TCP's efficiency, for this it sacrifices in-order delivery and packet re-transmission features. The proposed protocol is well suited for applications that generate delay sensitive traffic such as voice service over telephone but avoids to use UDP due to associated issues such as bufferbloat, protocol unfairness, congestion collapse.

Recent advances in smart devices and cellular networks have introduced new brands of internet applications which generates real-time, delay sensitive traffic, such as games, audio/video calling and social media apps. With every passing year this trend is growing and it is estimated that by 2020, 50 percent of the internet applications will be multimedia-oriented [2,3]. For example VoIP is one prominent and rapidly growing delay-sensitive application of Internet that has been widely adopted as a new way of voice communication [4]. It is also observed that there is a significant increase in multimedia traffic over UDP [5,6] which is causing critical issues such congestion collapse and protocol fairness [7,8].

In this changing situation the conventional transport protocol i.e. TCP/UDP needs to be reconsidered to for enhanced efficiency and quality of service requirements posed by these new applications. A recent study about the performance comparison of TCP, UDP and STCP indicated that TCP is slow for mutlimedia applications [9] while STCP is more efficient than UDP. TCP is a reliable transport protocol but this reliability comes at the cost of extra delays [10,11] and traffic [12] shows that TCP has largest delay in a simulated environment when compared with TCP, SCTP, DCCP and UDP, it further explains that this long delay is due to TCP's congestion control mechanism. On the other hand UDP is very efficient however it is not only unreliable but also is non responsive to congestion, furthermore it has some congestion issues of its own . UDP has no congestion control mechanism and a sender can generate unbounded network traffic which can easily overload a router and make it unavailable for any other traffic. The objective of our proposed RCP-BBR protocol is to provide efficiency comparable with UDP but without causing network congestion.

To efficiently estimate and avoid the network congestion, RCP-BBR adopts the idea of using recent bandwidth and round-trip time from TCP-BBR, the protocol handles congestion issues just like TCP-BBR. However in case of congestion or packet loss the proposed protocol adopts a UDP like behavior and maintains efficiency by not res-sending the lost packets, and hence do not need packet re-ordering service as well. RCP-BBR not only solves issues related to delay-sensitive applications, but also overcomes protocol fairness, bufferbloats [13], and network congestion problems. To evaluate the proposed scheme we developed many simulations on NS2 (Network Simulator version 2) [14]. NS2 is an open source discrete event simulation framework that supports modeling of variety of protocols including TCP, UDP and IP, along with provision to develop custom network simulations. The tool is widely used by network research community and is maintained by a variety of well know research organization such as DARPA, Sun Microsystem etc. The performance of RCP-BBR is compared with an efficient reliable transport protocol TCP-BBR and an unreliable transport protocol UDP. The performance is measured in terms of latency and throughput, as latency

indicates efficient delivery of packets at destination while throughput indicates packet loss and re-transmissions. Our simulation results indicates that proposed RCP-BBR protocol is more efficient than TCP-BBR and offers better throughput as compared to UDP. The Transmission Control Protocol (TCP) is a connection-oriented protocol that guarantees reliable data transfer with congestion and flow control mechanisms. TCP offers reliable data transfer with retransmission of lost packets, in-order delivery of data, acknowledgments of received packets and congestion control mechanisms. Internet applications (e.g., email, and file transfer) that require reliable data stream services use TCP for data transportation. However, TCP requires additional delay when a packet is lost or corrupted within the network [10,11]. On the contrary, delay-sensitive applications (e.g., VoIP, online games, and real time streaming) employ User Datagram Protocol (UDP), which is a connectionless and best effort delivery service protocol. It provides a basic packet delivery service without the guarantee of packet delivery, ordering or duplicate protection [11]. UDP does not provide congestion control or packet recovery mechanisms. Thus, delay-sensitive applications opt for UDP due to its speedy transmission. Moreover, applications prefer to use UDP when packet loss up to a certain extent is acceptable and delays due to the retransmission of the lost packets are not desirable.

1.1 Motivation

Recently, Internet has seen dramatic growth in the real-time traffic generated by the emergence of various real-time applications (e.g., VoIP, online gaming, and real-time broadcasting), which typically use UDP protocol [7,8]. By 2020, it is estimated that 50% of the Internet applications will be multimedia-oriented [2,3], which require fast data transmission and reliable connection. VoIP is one prominent and rapidly growing delay-sensitive application of Internet that has been widely adopted as a new way of voice communication [4]. However, the use of UDP for VoIP can cause sever congestion in real-time traffic, leading to call drops and frequent reconnections. Thus, resulting in negatively affecting the users' overall experience and satisfaction with the application. Therefore, there is a need to develop a new protocol that achieves low latency, high throughput, and low call drops ratio in delay-sensitive applications.

1.2 Problem statement

The increasing growth of the UDP-based real-time traffic could easily cause critical issues, such as congestion collapse, VoIP call drops [5,6] and protocol fairness problems [7]. It could even prevent the traffic of well-mannered congestion-controlled flows of TCP [15], and therefore, causes severe congestion and bufferbloat problems in deep buffer [13]. Bufferbloat problem is the buffering of excessive packets inside a network, causing congestion, unnecessary delay and reduced throughput [16]. Protocol fairness requires that multiple competing flows receive equal shares of the available bandwidth in a network. If any protocol obtains unfair capacity, it may tend to cause problems such as congestion collapse [15]. This makes UDP less suitable for the transportation of VoIP and other delay-sensitive applications due to its unfairness to TCP and other

protocols, even with its own competing UDP flows. On the contrary, TCP has a built-in congestion control mechanism, but it does not suit real-time applications because of its complex retransmission mechanism.

1.3 Contribution

In this paper, we introduce an enhanced responsive transport protocol as an alternate solution to UDP for delay-sensitive applications. Our protocol is based on the TCP-BBR (Bottleneck Bandwidth and Round-trip) congestion control mechanism proposed by Google [17]. We customize TCP-BBR congestion control framework to propose an enhanced Responsive Control Protocol (RCP-BBR) by efficiently removing reliability features to achieve low latency and high throughput. RCP-BBR solves not only delay-sensitive applications issues, but also overcomes protocol fairness, bufferbloats [16], and network congestion problems. Moreover, we focus on VoIP as a delay-sensitive application for the analyses and evaluation of our proposed work.

1.4 Paper structure

The rest of the paper is organised as follows: Sect. 2 presents an overview of the TCP-BBR protocol. Section 3 discusses our proposed approach. A proof of correctness for the proposed approach is presented in Sect. 4. Section 5 reports the experimental results. Finally, Sect. 6 concludes the paper.

2 Principles of BBR

TCP-BBR uses the recent measurements of bandwidth and round-trip Time (RTT) to significantly improve the bandwidth utilization with a low latency [16]. Different from a loss-based and delay-based congestion control mechanism, TCP-BBR is practically rate-based rather than window-based [17]. When in-flight data size is greater than the Bandwidth-Delay Product (BDP) of a path, a congestion occurs. BBR keeps in-flight data in range with a path's BDP. BBR only considers the bottleneck's bandwidth (the estimated maximum bandwidth available to a flow) and RTT (estimated from minimum RTT from a moving window). BBR tries to attain high throughput and low latency by estimating the above parameters. Rate balance is achieved when the data size related to the in-flight packets becomes equal to the BDP and that is the target operating point for BBR, as shown in Fig. 1. The changes in bottleneck's bandwidth and minimal RTT estimates are the congestion indicators for TCP-BBR. TCP-BBR limits its sending rate in response to a decrease in bottleneck's bandwidth. When a loss occurs, TCP-BBR goes into a recovery mode, but it is less conservative than TCP-Reno [15] that reduces the congestion window by a half.

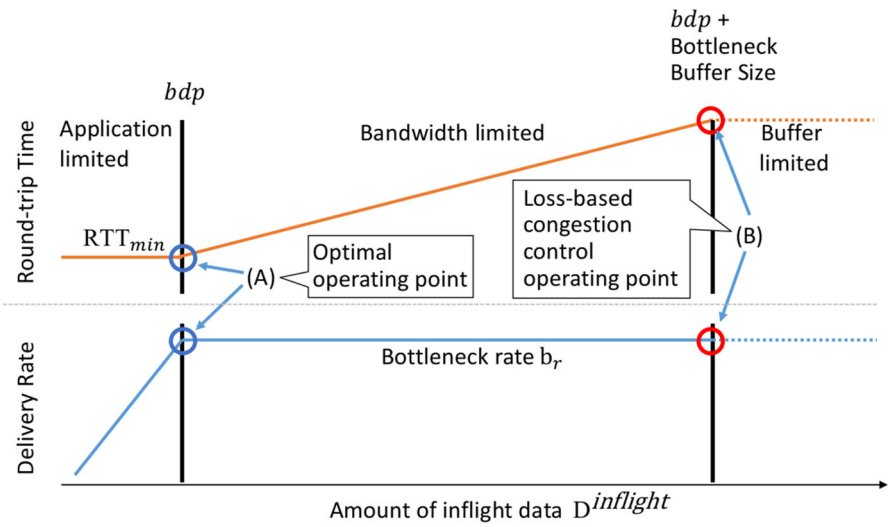

Fig. 1 Best operating point [18]

3 Proposed approach, TCP-BBR

In the standard TCP protocol, various overheads produce a delay in network transmission. These delays mainly are: TCP buffer delay, retransmission delay, queuing delay, packet re-ordering delay, Head of the Line (HOL) blocking, and delay related to packet acknowledgments [19]. TCP-BBR reduces latency and provides higher throughput without removing the aforementioned delay parameters (except reducing the queuing delay) as discussed in Sect. 2. Since the delay-sensitive applications realize retransmission and packet reordering by themselves, we can remove the features from the TCP-BBR sender's side refers to accelerate the transmission. From the receiver's side, we apply TCP-QUICKACK, in which TCP passes the received packets to the application layer as soon as it receives them. Afterwards, it sends an acknowledgment of the last received packet to the sender, where in-order delivery feature can be removed from receiver's side to avoid head of the line blocking.

3.1 Sender's side modification

We do the following modifications to TCP-BBR at the sender's side.

3.1.1 Removing packet retransmission

In the standard TCP, when a packet is lost due to triple duplicates, Retransmission Timeout (RTO) or any other reason, TCP retransmits the lost packet. In the proposed approach, we remove this feature, because in delay-sensitive applications, we cannot afford delays caused by lost packets retransmission and most applications have their

solutions to solve the problem. Algorithm 1 provides the pseudo-code for removing packet re-transmission.

Algorithm 1: Removing Packet Re transmission

```
   Input: Re-Transmission Condition is set to false
   /* Re-Transmission part                                          */
   Output: Modified TCP sender, will not re-transmit lost packets
1  Function Main(seqno, reason):
2      if (seqno == curseq) && (seqno > maxseq) then
3          idle()
4      if seqno > maxseq then
5          idle()
       /* If the packet has not been transmitted                   */
6      else
7          ++nrexmitpack
8          nrexmitbytes= nrexmitbytes+ bytes
9  return true
```

3.1.2 Removing TCP Nagle

In the TCP, TCP Nagles [20] algorithm is used at the sender's side and makes TCP transmit a larger packet until its buffer is full. Since delay-sensitive applications avoid any delay caused by waiting for the buffer to fill, we can remove TCP Nagles. In the proposed mechanism, similar to UDP, we change the receiver's behavior by not waiting for the lost packets, and forward the received packets to the application layer with no delay. Disabling TCP Nagles algorithm is also called TCP-NODELAY. Algorithm 2 provides the pseudo-code for disabling TCP Nagle in the proposed approach.

3.1.3 Modifying probe RTT state

Mostly, delay-sensitive applications (e.g., VoIP) have two states: the first one is 'burst' period where the application data is sent in burst; and the second one is 'silent' or 'idle' period where the application is idle by not sending any data or very little data. TCP-BBR estimates bandwidth and RTT based on the amount of data transmitted in recent time periods. In 'burst' period of VoIP, TCP-BBR gets an actual estimate of the bandwidth but in an idle period, the estimate of the bandwidth becomes application-limited. We maintain a variable for bandwidth estimation when the traffic suddenly switches from 'idle' state to 'burst' state. We also modify the 'probe-RTT' state of BBR which is 10 s, in the 'idle' state to not probe for RTT every 10 s when the sending rate is already low. The proposed approach keeps a record of the 'burst' period, and probes only for the RTT if the 'burst' period continues for over 20 s. We use 20 s as the threshold because TCP-BBR probes for RTT after every 10 s. If the 'burst' period continues for over 20 s, it creates a queue. Thus, to drain the queue and find the minimum RTT, we double the time of probe after 20 s.

Algorithm 2: TCP-Nodelay replacing the TCP Nagle's Algorithm

Input: TCP with Nagle's Algorith
Output: Modified TCP with no Nagle's Delay

```
/* TCP Nagle's Algorithm waits for TCP buffer to fill. This
   creates delay in VoIP. Hence, we need to disable Nagle's
   Algorithm, which by default is enabled                    */
/* Nagle's Algorithm                                         */
```

1 **if** *there is no bufferd data to send* **then**
2 | **if** *the window_size >= MSS and available data is >= MSS* **then**
3 | | send complete MSS segment immediately
4 | **else**
5 | | **if** *there is still unconfirmed data in the buffer* **then**
6 | | | enqueue data in the buffer until an ACK is received
7 | | **else**
8 | | | send data as quick as received

```
/* To remove TCP Nagle's Algorithm which waits for TCP buffer to
   fill. Here we remove If statement in second else statement   */
```

9 **else**
10 | **if** *there is still unconfirmed data in the buffer* **then**
11 | | enqueue data in the buffer until an ACK is received
12 | **else**
13 | | send data as quick as received

First, we check whether the RTT value is higher than the last minimum-RTT. If yes, we then check whether the RTT is less than 700 ms. If it is so, we continue in idle state, otherwise, we go for probe-RTT state. We use 700 ms as a threshold, because it is a two-way propagation delay, and most delay-sensitive applications delay requirement is to keep the delay below 400 ms. Taking 700 ms as a threshold is fair enough to keep one-way propagation below the applications delay requirement. We take 700 ms because in UDP, only one-way propagation delay matters. In our approach, we take a two-way propagation delay into account. One-way propagation takes 350 ms if both sending and receiving links take the same amount of time. As sending can take more time because of the routing or congestion from the sender's side, we reserve 100 ms for this purpose. Even if a receiver takes 300 ms, the sender does not exceed 400 ms in most cases.

3.1.4 Adding AppLimited variable

We propose maintaining the average BBR maximum bandwidth estimate (10 RTT) to address the problem of switching state between the 'idle' and 'burst'. Algorithm 3 provides the pseudo-code of all the listed modifications made to TCP-BBR on the sender side.

$$AppLimited = \sum_{i=bw}^{recent} Btlbw_{Max} every 10 Rtt's/n$$

```
/* where n is the number of recent max.bandwidth records
   */
```

Algorithm 3: RCP-BBR

```
   Output: Enabling TCP-NODELAY No Nagle Algorithm
1  if there is no bufferd data to send then
2      if the window_size >= MSS and available_data is >= MSS then
3          send complete MSS segment now
4      else
5          send data immediately
   Output: Adding application limited variable
   /* To use it when VoIP jumps from idle to burst mode                    */
6  Function App-limited(estimated.btlbw, recentMax.bw, state):
7      if estimated.btlbw >= 160 bytes && recentMax.bw>= 100 && state == ProbeBW then
8          burst-mode == true
9          recentmax-counter++
10         applimited-bandwidth += recent-Max.bw
11     if burst-mode then
12         app-limit = applimited-bandwidth / recentMax-counter
13     else
14         app-limit = 1
15     return Max(estimated.btlbw, app − limit);
   Output: Checking for ProbeRTT state whether to probe or not
16 Function Check-ProbeRTT(estimated.rtt, recentmin.rtt, state):
17     if estimated.rtt >= 700ms && recentMin.rtt>= 700 && state==ProbeBW then
18         burst-mode == true
19         rttseconds++
20     if burst-mode then
           /* Entering ProbeRTT state                                      */
21         ProbeRTT()
22         rttseconds++
23     else
24         if !burst-mode && rttseconds < 20sec then
25             rttseconds++
26             else
27                 ProbeRTT()
28                 rttseconds = 0
29     return rttseconds;
```

3.2 Receiver's side modification

After the sender's side modifications, now we discuss the receiver's side modifications.

3.2.1 Applying TCP QUICKACK

We apply the modified TCP QUICKACK at the receiver's end where TCP passes the received packets to the application layer as soon as it receives them, and then sends an acknowledgment of the last received packet to the sender, on which BBR depends.

3.2.2 Removing in-order delivery check

In the proposed approach, we remove the in-order delivery check from the receiver's side in order to remove the head of the line blocking, which creates seconds of lag in

communication. Thus, we achieve all the functionalities of UDP through the modified TCP-BBR with its acknowledgment feature.

Algorithm 4: QUICKACK

```
Output: QUICKACK and removing In-Order delivery
/* QUICKACK part                                                    */
Output: Modifying TCP reciever, to acknowledge newly received packet without waiting for other
        packets
1  Function output(packet, lastack):
2      if (seq_no == cur_seq_)&&(seq_no > max_seq_) then
3          Duplicate();
4      if seq_no > max_seq_ then
5          sendACKtoSender()
6          passPacketToApplication()
       /* If the packet the out of order                           */
7
8      else
           /* Let the Application Handle it                        */
9
10         passPacketToApplication()
11     return true;
```

Figure 2 presents a complete view of the proposed methodology.

4 Proof of correctness

UDP has a less overhead for generating header information than TCP and it is free from maintaining the variables that are needed for congestion and flow control (like the variables maintained in TCP). The end-to-end path delays and other hop-to-hop impairments that are added to the overall delay are the same for both TCP and UDP. However, UDP has no control over the queuing delay in routers unlike TCP. This leads to problems like bufferbloat and severe congestion. The overall delay for sending a segment using UDP takes an amount of time that can be calculated by using Equation *I*.

$$d_{udp} = \sum_{i=1}^{N} \{d^i_{proc} + dqueue^i + d^i_{trans} + d^i_{prop}\} \tag{1}$$

d_{udp} = End to end delay of a UDP packet
N = Total number of links between sender and receiver
d_{proc} = Processing delay, the time elapsed between arrival of a packet till it is assigned to outgoing queue.
d_{trans} = Transmission delay, time required to transmit the packet on medium
d_{queue} = Queuing delay at router
d_{prop} = Propagation delay or travel time in the medium

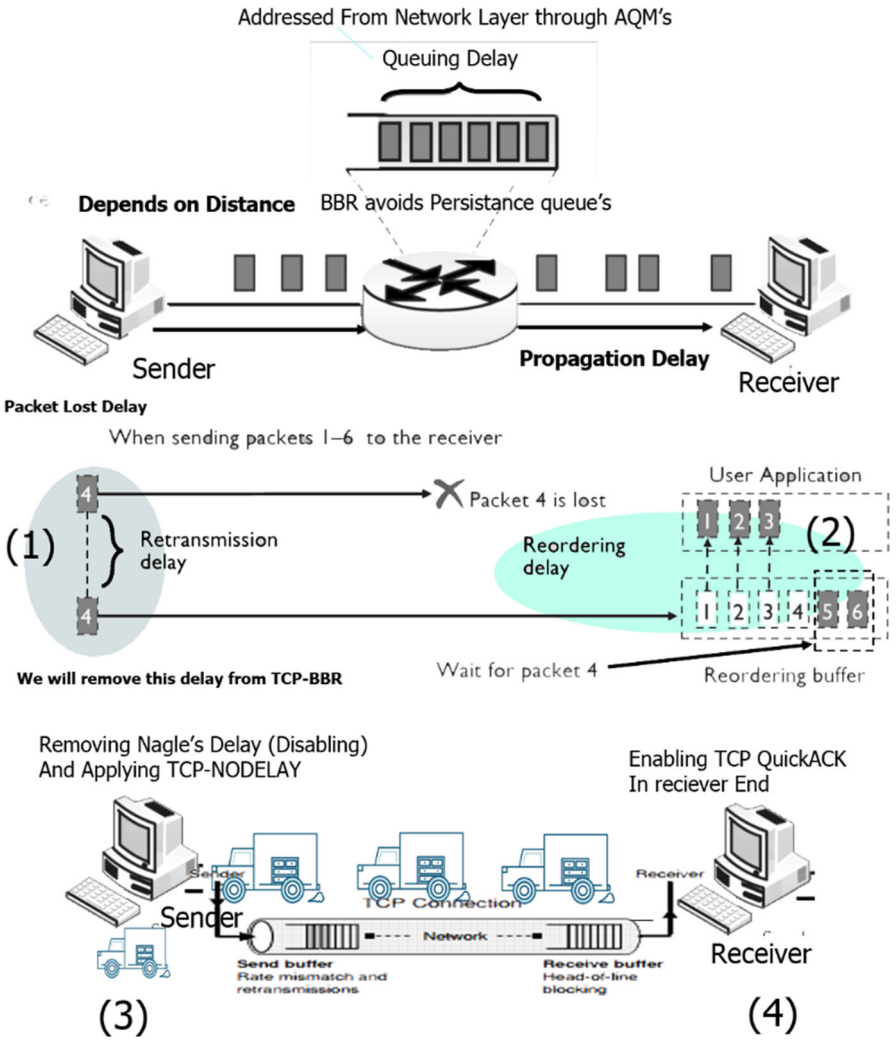

Fig. 2 A complete diagram of methodology

This delay should not exceed 400 ms threshold, otherwise, a VoIP application discards the late packets. Equation I clearly formalises the delay involved in sending a UDP segment. Standard TCP is not appropriate for delay-sensitive applications like online gaming and streaming. In TCP, all reliability features such as retransmission, buffering, in-order delivery etc. are considered. These produce a delay, which makes TCP infeasible for delay-sensitive and streaming applications. Using Equation II, the delay overhead of standard TCP can be calculated.

$$d_{tcp} = d_{Nagle} + d_{udp} + d_{rtx} + d_{rdr} + d_{ack} \tag{2}$$

d_{tcp} = End to end delay of a single tcp packet
d_{Nagle} = Delay caused by Nagle algorithm
d_{udp} = As given by Eq. 1
d_{rtx} = Re-transmission delay, time required to re-transmit a lost packet
d_{rdr} = Time spend in re-ordering packets at receiver
d_{ack} = Time elapsed at sender while it waits for acknowledgement

For a VoIP application, we do not need to retransmit the lost packets and also cannot afford packet reordering at the receiver's end, which creates head-of-the-line blocking. To achieve UDP-like speed, we eliminate those delays, which make TCP infeasible for delay-sensitive applications and hence, we propose modifications in TCP-BBR. The proposed mechanism also achieves congestion control like in BBR, which accurately estimates the available bottleneck bandwidth. In case of UDP, sending with more data rate than available bandwidth causes packet loss and congestion in routers, which is not handled by UDP. Hence by removing retransmission, in-order delivery from TCP, and by keeping acknowledgment feature that BBR needs for estimation, we can control packet losses, bufferbloats (severe congestion in deep buffers), and congestion collapses. It also provides help and more intelligence for VoIP applications and application developers to choose the right encoding scheme. By applying TCP-BBR on VoIP and other delay-sensitive applications, we achieve more router-friendly traffic unlike UDP traffic, which is by default blocked by most of the firewalls due to security and other issues. These changes lead us to Equation III:

$$d_{RCP-BBR} = d_{tcp} + d_{ack} \tag{3}$$

$d_{RCP-BBR}$ = End to end delay of a RCP-BBR packet
d_{tcp} = As given by Eq. 2
d_{ack} = Time elapsed at sender while waiting for acknowledgement

Equation 3 makes the basis of the proposed approach. In the proposed approach, the data is sent as quickly as it is sent by UDP (Fig. 3). The packet acknowledgment feature is kept on with some addition of bitwise overhead of TCP header and TCP variables states maintenance, which add little delay to the overall end-to-end sending of the packet. All above TCP overheads are bitwise operations which do not exceed over 5 ms. Normally, a VoIP application sends data every 20 ms. So, even if we increase that interval to 25 ms, it has no impact on the overall quality of the VoIP application. Applying this strategy controls the call drops by compromising a littl on quality. Figure 3 shows how the proposed mechanism controls the call drops by not

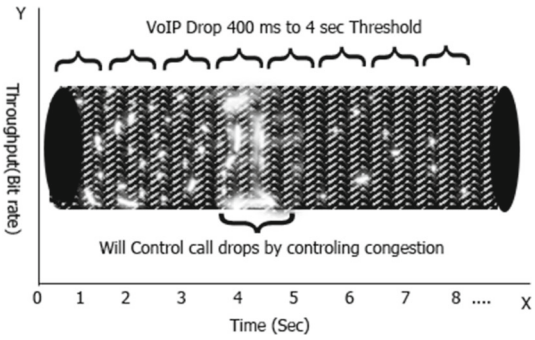

Fig. 3 Call drops after applying above technique

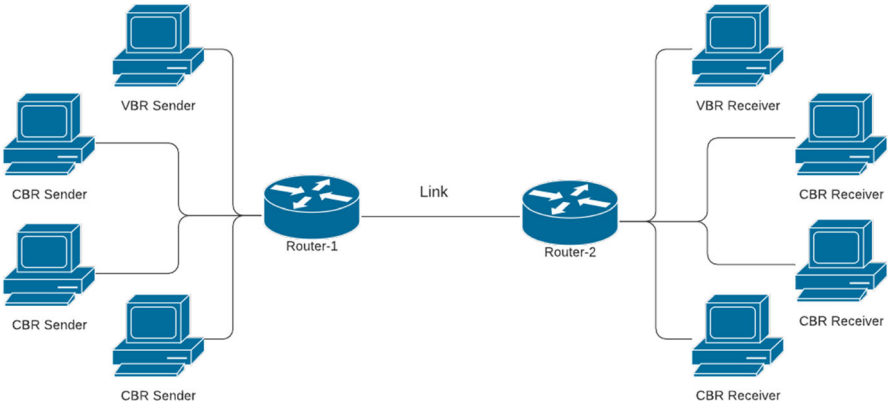

Fig. 4 Network topology of NS-2 simulations

letting buffers to get full and to cause severe congestion. As a result, we get enough data flowing which does not leave the VoIP application call drop threshold to be met [5,6].

5 Performance evaluation

We evaluate the proposed scheme using the NS2 network simulator.[1] We used a dumbbell topology with multiple senders and receiver nodes on two sides of a single link, each end of the link is connected to a router. This is consistent with the simulation setups that are used in the related literature, e.g., [21], for the evaluation of VoIP and Constant Bit Rate (CBR) traffics. The proposed scheme is compared with UDP and TCP-BBR (NODELAY) using average end-to-end delay, throughput, jitter, and the number of lost packets as the performance metrics.

To evaluate the proposed scheme we developed a number of simulations using Network Simulator 2 (NS2), the simulation parameters are taken from a similar study [21]. There are 4 senders, 4 receivers and two routers as show in the Fig. 4. The link between routers is considered as the bottleneck link where we can see the effect of

[1] https://www.isi.edu/nsnam/ns/.

Table 1 Different configurations (some results are listed here; the remaining ones are shown in the figures): Q: queue size, D: delay (ms), C: capacity (Mbps), N = number of lost packets, J = Jitter (ms)

Run	Link b/w routers			Other links		BBR		UDP		RCP-BBR	
#	*Q*	*D*	C	*D*	C	*N*	J	*N*	J	*N*	J
1	25	3	1.92	5	1	0	0.00001	0	0.00001	0	0.00001
2	5	3	1.92	5	1	0	0.00001	0	0.00001	0	0.00001
3	50	90	0.50	30	1	2930	28	1956	25	1885	27
4	5	90	0.50	30	1	3015	0.21	2046	0.019	1975	0.020
5	20	290	0.30	30	1	3386	0.067	2603	0.057	2524	0.062

traffic load. We used a number of link capacities ranging from 0.30 to 1.92 Mbps for the bottleneck link, this was done to study the behavior of traffic response of protocols. As queues at routers plays an important role in packet delays and handling congestion, we used a number of queue sizes in the routers ranging from 5 to 50 packets. The queues acting as buffers are observed to detect the development of congestion, we studied the effect of changing the size of these buffers. For performance evaluation we simulated three candidate protocols TCP-BBR, UDP and our proposed protocol RCP-BBR, for each simulation we calculated packet loss, jitter, delay and throughput for each protocol. Throughput and latency here are linked with the work which is being customized and are considered as standard terms used in TCP/UDP based networks. Formally, the term latency here is used to represent the total time elapsed from sending a packet by a source till its acknowledgment is received. Similarly throughput is taken as number of successfully received packets.

5.1 Simulation setup

The CBR traffic flows run between three sender and three receiver nodes, whereas a VBR traffic flow runs between the fourth sender and the fourth receiver nodes. The simulation time is 12 s. The configuration of the links and the routers is given in Table 1. We developed 5 simulation scenarios, which corresponds to five different networks. The first two simulations are developed to simulate network condition where there is no congestion and no packet is lost. We call it stable network condition, the other three simulation represents networks with possible congestion and we refer these networks as unstable networks. In simulation 3, 4 and 5 we changed the network conditions such that there will be more congestion and increased packet loss, this is done to study the performance of selected protocols in congested networks. We considered VoIP as network traffic , each simulation is executed for 12 s and we recorded various network parameters.

The five simulation scenarios/runs in Table 1 correspond to five different network conditions. The simulation runs 1 and 2 provide the parameter values for a stable network having large and small buffer sizes of the routers respectively. The runs 3 and 4 provide the parameter values for an unstable network having large and small buffer

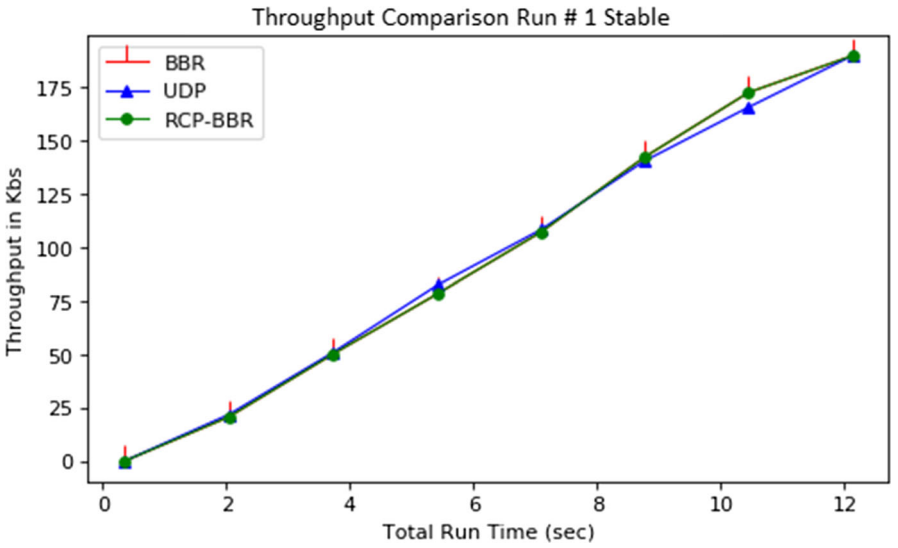

Fig. 5 Stable network scenario with large queues of routers (Run 1): throughput comparison

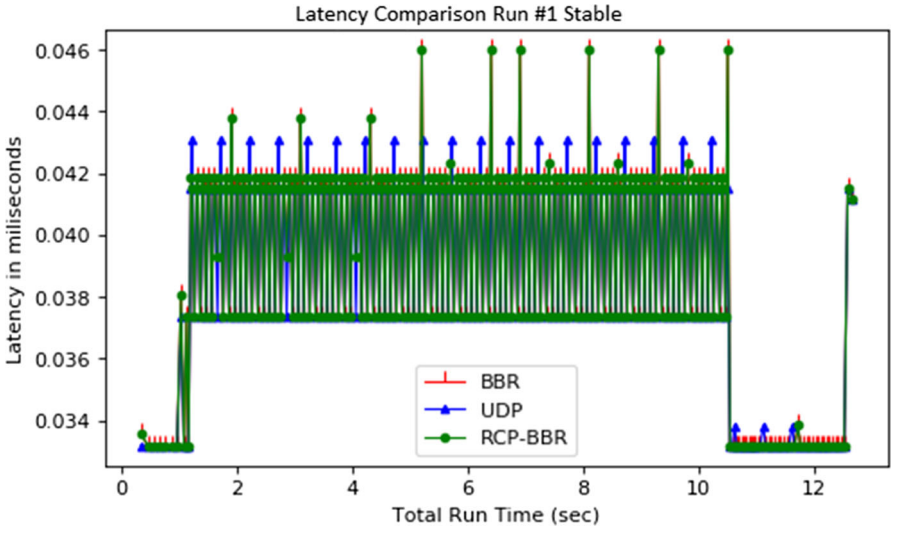

Fig. 6 Stable network scenario with large queues of routers (Run 1): latency comparison

sizes of the routers respectively. The run 5 provides the parameter values for a highly unstable network.

5.2 Results and discussion

Simulation run 1 and 2 are designed for a no loss situation. The no loss scenarios are developed to evaluate and compare protocol performances to determine their

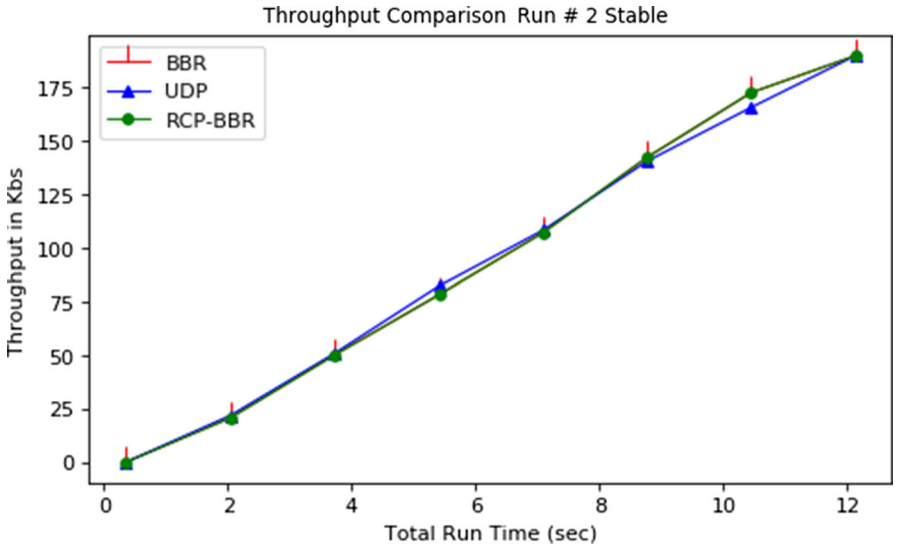

Fig. 7 Stable network scenario with small queues of routers (Run 2): throughput comparison

dependence upon network configuration and available resources. Also this is usually taken as ideal behavior and is not considered as overall performance for all situations. As shown in Figs. 5, 6, 7, and 8, all protocols performed almost in a similar way in terms of throughput and latency. It is very encouraging that BBR performed similar to UDP despite the fact that it has a congestion control mechanism which means extra processing and delays where as UDP is free from such burdens. The benefit of BBR is that it will attempt to handle congestion while keeping up the performance close to UDP, whereas UDP will only worsens the situation.

Figures 9 and 10 show the results of an unstable network scenario with a large queue (run 3) and packet losses in the network. In this scenario, the performance of the proposed scheme (RCP-BBR) and UDP is similar. Although, as shown in Fig. 9, the throughput of BBR with TCP-NODELAY is similar but due to significant number of packet drops, its performance is not satisfactory for VoIP and delay-sensitive applications. With only a few milliseconds difference, UDP and RCP-BBR finish the transmission within 13 s. However, transmitting the same amount of data by BBR takes over 16 s, which is not suitable for VoIP traffic.

Figures 11 and 12 show an unstable network scenario with a small queue (run 4). In this scenario, the number of packet losses is more than those in run 3. Considering the throughput, the performance of UDP and RCP-BBR is similar at the start (transient state) but with the passage of time the performance of RCP-BBR becomes better (steady state). Considering the latency, the performance of UDP is better at start only (transient state), while the performance of RCP-BBR is better in the long run (steady state).

Figures 13 and 14 show the results of run 5 for which there is a highly unstable network having packet losses and high link delays. In this simulation, the performance of BBR is worse than that of UDP and RCP-BBR. The performance of UDP and RCP-

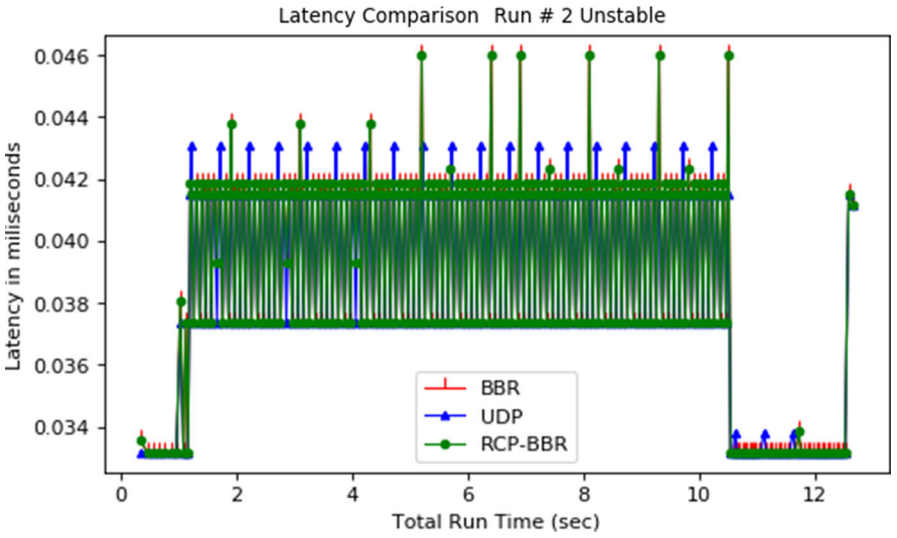

Fig. 8 Stable network scenario with small queues of routers (Run 2): latency comparison

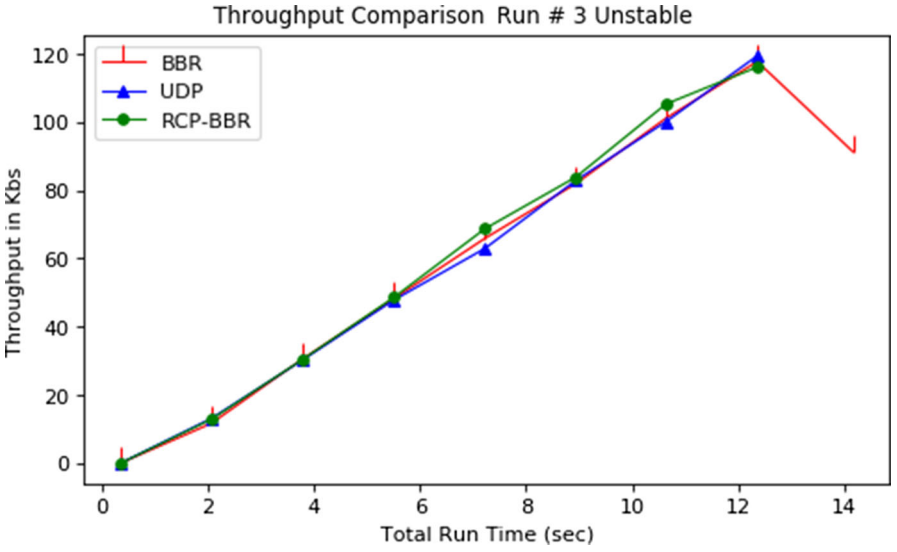

Fig. 9 Unstable network scenario with large queues of routers (Run 3): throughput comparison

BBR is similar. Considering the latency, the performance of UDP is better than that of RCP-BBR at the start (transient state) but with the passage of time (steady state), when congestion occurs, UDP keeps network buffers full and that consequently creates more delays as compared to RCP-BBR. RCP-BBR handles network congestion and does not overload a router's buffer. In highly unstable networks, the proposed approach keeps small buffers while UDP overloads the buffers and that consequently creates

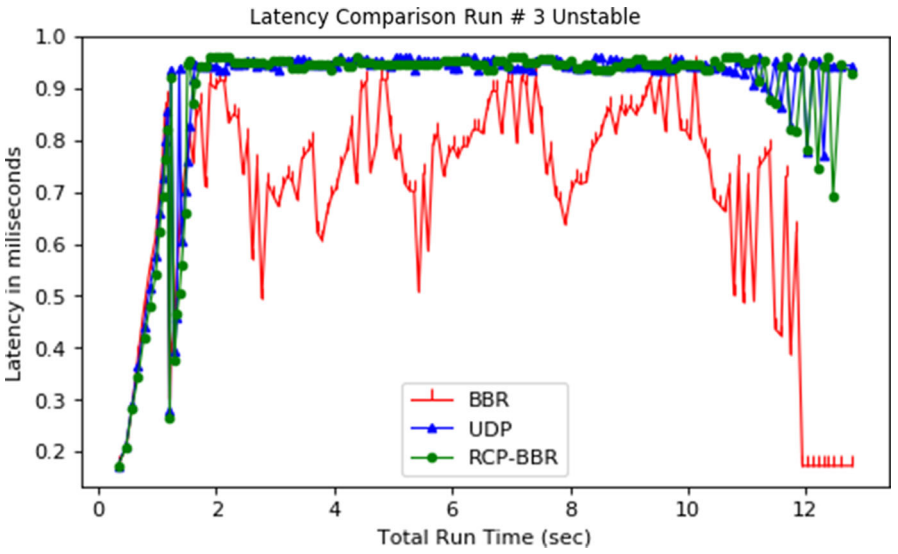

Fig. 10 Unstable network scenario with large queues of routers (Run 3): latency comparison

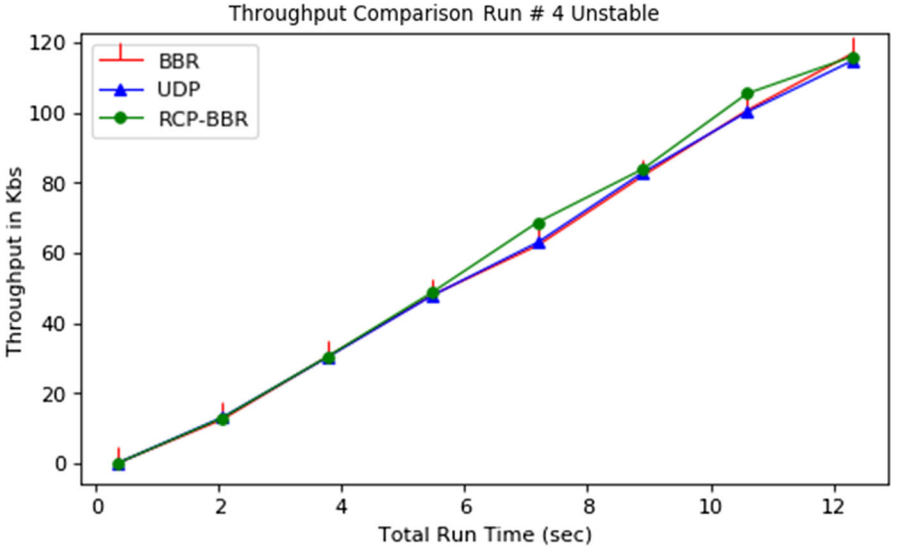

Fig. 11 Unstable network scenario with small queues of routers (Run 4): throughput comparison

buffering delays. Hence, it can be said that the proposed approach performs better in a highly unstable network.

In stable networks, all the considered protocols are suitable for VoIP applications. In unstable networks (Runs 3–5), TCP-BBR results in a significant packet loss, which is not suitable for VoIP traffic. The proposed approach (RCP-BBR) performs better than UDP in deep congested buffers (Runs 3–5) as the average latency of the proposed

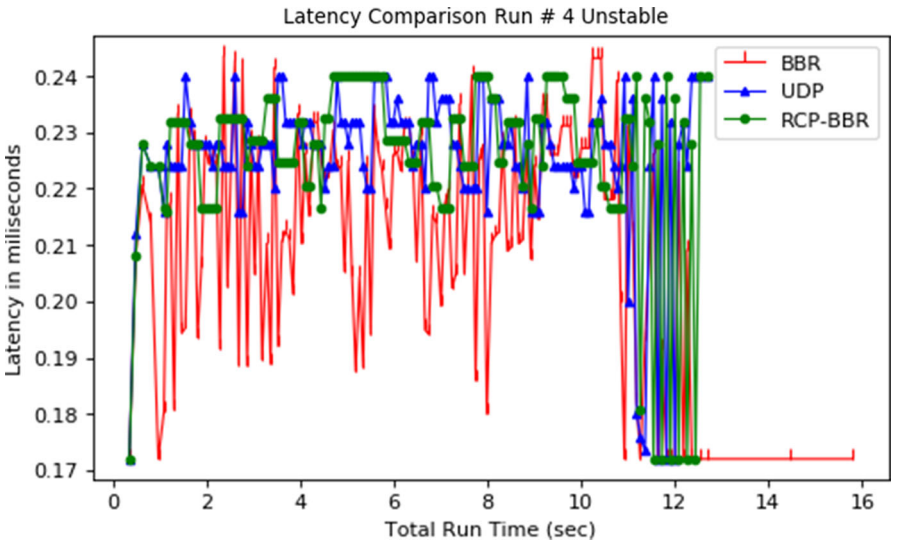

Fig. 12 Unstable network scenario with small queues of routers (Run 4): latency comparison

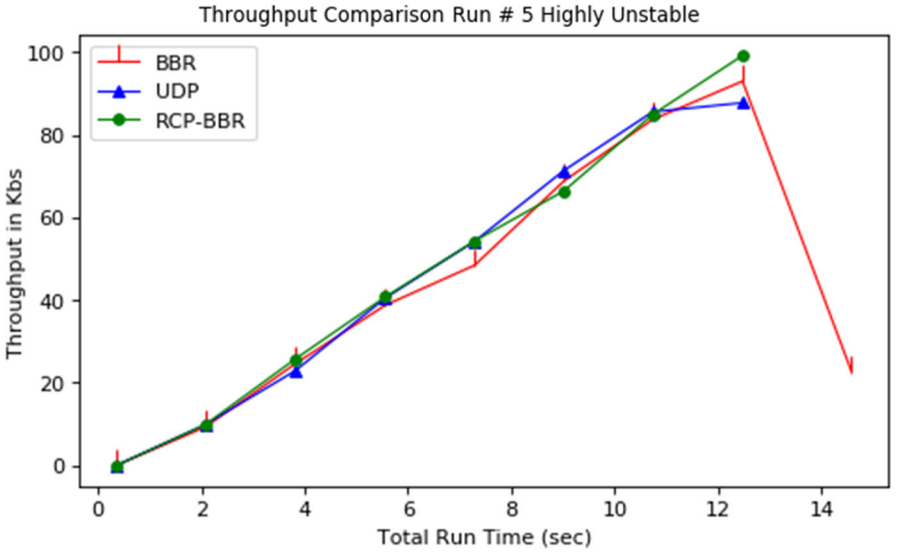

Fig. 13 Highly unstable network scenario (Run 5)

technique is less than that of UDP. Therefore, RCP-BBR performs better than UDP in practical scenarios.

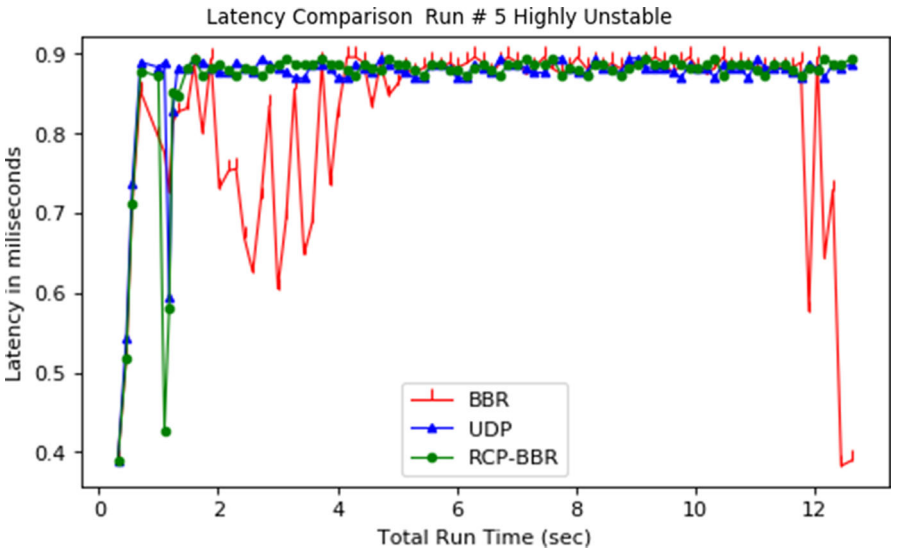

Fig. 14 Highly unstable network scenario (Run 5)

6 Related work

Congestion in wide area computer networks such as internet has always been a very active and dynamic area for researcher. As the network equipment, technology, type and amount of traffic keeps of progressing, there are always new problems related to congestion. Handling congestion is tricky as latency, throughput , and quality do not adjust with each other easily and there is always a compromise and a balanced needed. Each new solution brings up some new adjustments, in this section we are focused at viewing the congestion issues of real time traffic specifically VoIP. For such traffic throughput is usually a priority along with Quality of Service (QoS), UDP provides good throughput but is poor at QoS, also it raises many congestion issues as well. Following is a brief description of related work, a careful review of existing solutions revealed that a better solution is possible.

Although, congestion control is a network layer problem, yet it could be addressed from both network layer and transport layer. From network layer perspective, congestion could be controlled either by using queue management (QM) or Active Queue Management (AQM) [22,23]. However, AQM is more advance and adaptive than queue management. Many QMs and AQMs have been introduced and deployed in recent decades. Drop Tail [22,24] was introduced to work in FIFO manner and operate the network buffer as a FIFO queue. Later, when traffic increased on network routers, Drop Tail could not perform well because of its nature of filling the buffer and leaving no space behind, which caused persistent queues. Then, Random Early Discard (RED) [24], a new queue management approach, was introduced which keeps some threshold for dropping packets when congestion occurred. It does not access the network buffer by keeping max-min threshold. RED performs well and still used by network routers. With the passage of time several variants of RED were proposed [24]. Lately, another

AQM was introduced for network routers called CoDel (control delay), which is an adaptive queue management mechanism. Unlike RED, it performed extra calculation to drop packets by calculating packet in-time and out-time differences to drop packets randomly. It is an IETF implementation project which is an ongoing research [25,26]. A hybrid of RED and CoDel is a researched by [27] which provides a balance of these two scheme and is efficient at handling the bufferbloat problem. A similar solution is also proposed recently by Jim Gettys and his team [16] they addressed the problem from network layer as well as transport layer perspective.

Congestion as transport layer problem has long been researched, initially, AIMD [28] (Additive Increase and Multiplicative Decrease) was introduced on top of TCP to provide some basic congestion control and flow control algorithms, such as slow start and congestion avoidance. It introduced additive increase in normal transmission and a multiplicative decrease in transmission when congestion is signaled. For quite some period, this mechanism worked well, but as traffic and usage of Internet increased, many problems occurred which AIMD could not handle. Therefore, many different mechanisms introduced afterwards based on different congestion signals such as loss based and delay-based mechanisms. Basically, they are called different TCP variants, such as TCP-Reno, TCP-BIC, TCP-NewReno, TCP-Vagas, TCP-Cubic, Compound TCP (Microsoft) [19,29–31] and later on TCP-BBR on which this paper is based.

Congestion control is not only discussed for TCP but there are some solutions proposed for UDP such as [32] provides a low latency and high throughput datagram control protocol, similarly a [33] provides a comparative study of high speed congestion control protocols. These studies indicates that congestion can be handled at network layer, by TCP as well as by a UDP protocol as well.

All of mentioned congestion control algorithms are either loss/delay based or sometimes a hybrid of both such as Microsoft Compound TCP [29]. Recently, a new congestion control framework was introduced by Google called 'BBR' [17] that stands for "Bottleneck Bandwidth and Round-Trip Time" based on estimation and it precisely estimated the available bandwidth and bottleneck bandwidth plus minimum Round-Trip Time (RTT). It modeled the whole network path as a single bottleneck by which BBR claimed that it provided high throughput and low delay. Initially, these claims seemed to be promising. Google implemented this BBR on Google B4 WAN servers and YouTube edge servers [34] that improved their performance up to 20% worldwide. But in recent research [18], "Experimental evaluation of BBR congestion control" the writers explained full and deep experimental evaluation of BBR at higher speeds and different scenarios which varied in parameters such as number of flows, flows RTT and different buffer sizes at bottleneck. Among other things, they evaluated it by considering the features like throughput, packet loss, fairness comparison between Cubic and BBR. They used Linux kernel 4.9 for their experimental setup and tested on a test-bed with 1 Gbps and 10 Gbps data rate at bottleneck. At the end of a rigorous experimental evaluation Mario Hock and his team were able to conclude that BBR had a lot of retransmission on small buffers compared Cubic TCP. They also stated that with multiple flows BBR did not stay fair with Cubic TCP. As BBR is an ongoing implementation and research Neal and his team did some modification to their framework and called it BBR v2 [34] which solved the problems pointed by Mario Hock in [18].

7 Conclusion

In stable networks, both TCP and UDP protocols suit VoIP delay-sensitive applications perform, but in unstable networks TCP's performance deems deficient due to delays and reliability issues such as retransmission and in-order delivery delays. UDP causes Quality of service QoS problem in unstable networks, such as frequent call drops in VoIP and congestion and bufferbloat problems in networks. In order to overcome this problem, we introduce RCP-BBR, a new responsive control protocol that perform well in both stable and unstable networks. By removing retransmission, in-order delivery and other overheads from modifying TCP-BBR and keeping the receiver's acknowledgment, we achieve a higher throughput in shallow buffers and a lower delay in deep buffers. After evaluation of our proposed approach, we achieve similar delay performance and up to 5% improvement in throughput over UDP in stable networks, while in unstable and long-distanced networks, we achieve smaller queues, and low delays where UDP performed poorly by keeping delays above VoIP call drop threshold. Therefore, our approach performs similarly both in stable and high-speed networks, and in unstable networks. Our approach keeps queuing delays low to VoIP threshold in deep buffers while UDP keeps buffers full and delays high which frequently met call drop thresholds.

References

1. Scholz D, Jaeger B, Lukas S, Daniel R, Fabien G, Georg C (2018) Towards a deeper understanding of tcp bbr congestion control. In: 2018 IFIP networking conference (IFIP networking) and workshops, pp 1–9. IEEE
2. Najjari N, Min G, Hu J, Zhao Z, Wu Y (2017) Performance analysis of wlans with heterogeneous and bursty multimedia traffic. In: GLOBECOM 2017–2017 IEEE global communications conference, pp 1–6. IEEE
3. Gerber A, Doverspike R (2011) Traffic types and growth in backbone networks. In: Optical fiber communication conference, pp OTuR1. Optical Society of America
4. Mayo JW, Ukhaneva O (2017) International telecommunications demand. Inf Econ Policy 39:26–35
5. Baset SA, Schulzrinne H (2004) An analysis of the skype peer-to-peer internet telephony protocol. arXiv preprint arXiv:cs/0412017
6. Zhang X, Xu Y, Hu H, Liu Y, Guo Z, Wang Y (2012) Profiling skype video calls: rate control and video quality. In: INFOCOM, 2012 proceedings IEEE, pp 621–629. IEEE
7. Zhang M, Dusi M, John W, Chen C (2009) Analysis of udp traffic usage on internet backbone links. In: Ninth annual international symposium on applications and the internet, 2009. SAINT'09, pp 280–281. IEEE
8. Chu C-Y, Chen S, Yen Y-C, Yeh S-L, Chu H-H, Huang P (2018) Eq: a qoe-centric rate control mechanism for voip calls. ACM Trans Model Perform Eval Comput Syst 3(1):4:1–4:20
9. Halepoto IA, Arain AA, Hussain U (2018) Evaluation of multimedia streams in internet applications. In: Proceedings of the 8th international conference on information systems and technologies, ICIST 18, New York, NY, USA, Association for Computing Machinery
10. Giannoulis S, Antonopoulos C, Topalis E, Athanasopoulos A, Prayati A, Koubias S (2009) TCP vs. UDP performance evaluation for CBR traffic on wireless multihop networks. Simulation 14:43
11. Behrouz AF, Sophia CF (2002) TCP/IP protocol suite. McGraw-Hill Higher Education, New York
12. Nor SA, Alubady R, Kamil WA (2017) Simulated performance of tcp, sctp, dccp and udp protocols over 4g network. Proc Comput Sci 111:2–7
13. Braud T, Heusse M, Duda A (2014) Tcp over large buffers: when adding traffic improves latency. In: Teletraffic congress (ITC), 2014 26th international, pp 1–8. IEEE

14. Issariyakul T, Hossain E (2009) Introduction to network simulator 2 (ns2). In: Introduction to network simulator NS2, pp 1–18. Springer
15. Hasegawa G, Murata M (2001) Survey on fairness issues in tcp congestion control mechanisms. IEICE Trans Commun 84(6):1461–1472
16. Gettys J, Nichols K (2011) Bufferbloat: dark buffers in the internet. Queue 9(11):40
17. Neal C, Yuchung C, Stephen GC, Hassas YS, Van J (2017) BBR: congestion-based congestion control. Commun ACM 60(2):58–66
18. Hock M, Bless R, Zitterbart M (2017) Experimental evaluation of BBR congestion control. In: 2017 IEEE 25th international conference on network protocols (ICNP), pp 1–10. IEEE
19. Allman M, Paxson V, Stevens W et al (1999) TCP congestion control
20. Zhang X, Schulzrinne H (2004) Voice over TCP and UDP. Department of Computer Science, Columbia University, Tech. Rep. CUCS-033-04
21. Andrew L, Marcondes C, Floyd S, Dunn L, Guillier R, Gang W, Eggert L, Ha S, Rhee I (2008) Towards a common tcp evaluation suite. In: Proc, PFLDnet
22. Kiruthiga B, George E, Raj DP (2014) Survey on aqm congestion control algorithms. Int J Comput Sci Mob Appl 2(2):38–44
23. Dukkipati N, Cheng Y, Vahdat A (2016) Research impacting the practice of congestion control
24. Katiyar V, Jain AC (2014) A survey on red and some it's varients incongestioncontrol mechanism. Int J Eng Manag Res 4(4):184–188
25. Nichols K, Jacobson V, McGregor A, Iyengar J (2013) Controlled delay active queue management. In: Work in progress
26. Nichols K, Jacobson V, McGregor A, Iyengar J (2015) Controlled delay active queue management: draft-ietf-aqm-codel-03. In: IETF, December
27. Pan R, Natarajan P, Piglione C, Prabhu MS, Subramanian V, Baker F, VerSteeg B (2013) PIE: a lightweight control scheme to address the bufferbloat problem. In 2013 IEEE 14th international conference on high performance switching and routing (HPSR), pp 148–155. IEEE
28. Jacobson V (1988) Congestion avoidance and control. In: ACM SIGCOMM computer communication review, vol 18, pp 314–329. ACM
29. Song KTJ, Zhang Q, Sridharan M (2006) Compound TCP: a scalable and TCP-friendly congestion control for high-speed networks. In: Proceedings of PFLDnet 2006
30. Ha S, Rhee I, Lisong X (2008) CUBIC: a new TCP-friendly high-speed TCP variant. ACM SIGOPS Oper Syst Rev 42(5):64–74
31. Brakmo LS, Peterson LL (1995) TCP vegas: end to end congestion avoidance on a global internet. IEEE J Sel Areas Commun 13(8):1465–1480
32. Cheng Y, Cardwell N (2016) Making linux TCP fast. In: Netdev conference
33. Abed JB, Sinda L, Mani MA, Mbarek R (2012) Comparison of high speed congestion control protocols. Int J Netw Secur Appl 4(5):15
34. Cardwell N, Cheng Y, Gunn CS, Yeganeh SH, Jacobson V (2017) BBR congestion control: an update. In: Presentation in ICCRG at IETF 98th meeting

Affiliations

Sayed Najmuddin[1] · Muhammad Asim[1] · Kashif Munir[1] · Thar Baker[2] · Zehua Guo[3] · Rajiv Ranjan[4]

✉ Muhammad Asim
muhammad.asim@nu.edu.pk

Sayed Najmuddin
sayed.najmuddin@gmail.com

Kashif Munir
kashif.munir@nu.edu.pk

Thar Baker
T.baker@ljmu.ac.uk

Zehua Guo
guolizihao@hotmail.com

Rajiv Ranjan
raj.ranjan@ncl.ac.uk

1 National University of Computer and Emerging Sciences, Islamabad, Pakistan

2 Liverpool John Moores University, Liverpool, UK

3 Beijing Institute of Technology, Beijing 100081, China

4 Newcastle University, Newcastle upon Tyne, UK