A Comprehensive Study of Modern and High Speed TCP-Variant in Linux Kernel: TCP CUBIC

Abrar Khan, Umar Shoaib, and Muhammad Sarfraz
Department of Computer Science, Faculty of Computing and Information Technology, University of Gujrat, Pakistan

Abstract: *Transmission Control Protocol TCP is no doubt most widely used congestion control protocol designed for highly reliable and end-to-end communication over the internet. TCP is not suitable in its standard form for modern and high speed networks. Various TCP variants are solution for this issue. CUBIC is a modern TCP variant designed for high speed and scalable networks. CUBIC is also adopted as default congestion control algorithm in Linux kernel. This survey paper contains a detailed discussion about TCP CUBIC and the directions for further improvements. It describes the CUBIC design architecture with the pseudo code of the algorithm, TCP support in Linux kernel and implementation of CUBIC, Network Simulator 2 and Network Simulator 3 based study of CUBIC along with its class diagram. Finally, the performance of CUBIC is evaluated both in wired and wireless environment under the parameters of goodput and intra-protocol fairness along with TCP NewReno and TCP Compound. The simulation results demonstrate that CUBIC is very suitable for wired and high speed networks but its performance degrades in wireless and low speed networks.*

Keywords: *TCP, Congestion control, Linux kernel, CUBIC, NewReno, Compound.*

Received July 18, 2016; accepted July 20, 2017

1. Introduction

Congestion control is a technique used for the effective and efficient utilization of channel bandwidth by managing the extra network traffic that exceeds from the channel capacity. It is used to avoid packet losses, network delay or the blockage of network. With the advancement in internet and network technologies, various applications are being developed that run over the internet. The traditional networks have now been converted into high speed, long distance, large bandwidth and scalable networks. Significant amount of data is transmitted today over the internet by various wireless and hand-held devices and all the Hyper Text Transfer Protocol (HTTP) traffic relies on Transmission Control Protocol (TCP) [16]. Consequently, the performance of TCP is challenged in case of modern and high speed networks. TCP CUBIC has been designed to overcome such issues.

CUBIC [11] is a modern TCP variant with the specialty of its improved congestion window growth function. It replaces the property of linear growth congestion window function of legacy or standard variants by a cubic function to improve its performance and to achieve high scalability. The key feature of CUBIC is that its window growth is defined in real time and it does not depend upon Round Trip Time (RTT). It depends only on the occurrence of two consecutive congestion events.

It is an improved and enhanced form of Binary Increase Congestion (BIC) TCP [27]. CUBIC has also advantage over other variants and after a continuous testing and careful performance evaluation, Linux Community implemented CUBIC as its default algorithm in its kernel version 2.6.19 in 2006 [11, 20]. Other TCP variants include Tahoe [25], Reno [14], New Reno [12], Vegas [6], Fast Active Queue Management Scalable TCP (FAST) [26], Compound [24], High Speed TCP (HSTCP) [9], West-Wood [8], Illinois [18], and Hybla [7, 21]. The performance analysis for the techniques is important [13].

Rest of the paper is organized under the following structure. Section 2 includes background and related work, Section 3 emphasizes on the design architecture of CUBIC, Section 4 highlights the Linux kernel regarding TCP, Section 5 contains NS2 and NS3 support regarding the simulation of CUBIC and Section 6 implements the performance analysis of CUBIC, NewReno and Compound. Section 7 concludes the paper.

2. Background and Related Work

In TCP, congestion is inferred at the receiving end based on timeout or duplicate acknowledgement [19]. Congestion control algorithms have been classified into three general categories including loss based, delay based and hybrid methods [3, 15]. In loss based method, the size of congestion window (Cwnd) depends upon packet losses. In its normal behavior, the Cwnd size is increased after the acknowledgement of each packet. When there is a packet loss, the Cwnd decreases significantly [2, 17]. The loss based

algorithm includes TCP Tahoe, Reno, NewReno, STCP, HSTCP, BIC and CUBIC.

In delay based algorithms, the size of Cwnd relies on RTT. These methods decrease Cwnd size according to RTT increase, which causes the increase of both in network router's load as well as queue length. One difference between loss based and delay based approach is that Cwnd decreases after congestion in case of loss based while it decreases before congestion in delay based algorithms. Due to this reason, the performance of loss based method is better than that of delay based method when sharing a network link. However, the obtained throughput almost remains stable. TCP Vegas [6] is an example of delay based TCP variant.

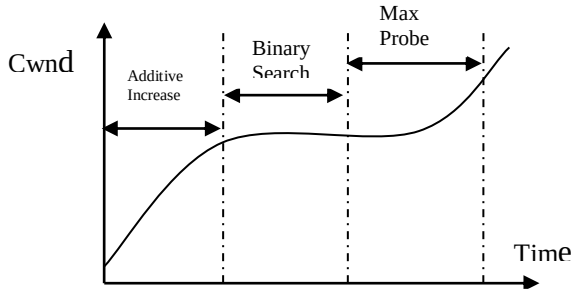
The hybrid algorithms combine both the features of loss based and delay based algorithms [15]. TCP Compound [24] is an example of such algorithms.

TCP Tahoe was the first TCP variant introduced by Jacobson [14]. Its functionality covers slow start, congestion avoidance and fast retransmission [10]. TCP Reno adds the functionality of fast recovery in Tahoe. TCP NewReno was designed in 1996 and it is in fact a modification in existing Reno by improving the retransmission mechanism during the phase of fast recovery [5]. It recovers multiple packet losses and does not exit from the fast recovery phase until the acknowledgement of all the segments is received [12]. When an ACK is received the NewReno performs two functions. In case of partial ACK, it assumes that segment is lost and it retransmits the segment by marking duplicate ACK to zero until all the data is acknowledged. In case of full ACK it exits from fast recovery phase and continues its operation in congestion avoidance phase.

Compound TCP [24] is also designed for fast and long distance networks. It was implemented as hybrid protocol as it utilizes both delay-based and loss-based approaches of congestion control to resolve the under utilization issue of channel capacity. Along achieving bandwidth scalability, it also attains RTT fairness. The core function of CTCP is that it increments a scalable component into the standard TCP Reno for the efficient utilization of link capacity and sensing the bottleneck queue.

3. The TCP CUBIC-Algorithm

The CUBIC is an enhanced form of the BIC algorithm for high scalability and RTT fairness [23]. The major specialty of BIC is its window growth function which is very suitable for high speed networks [27]. In smooth operation, the window size is set to maximum denoted by W_{max}. After a packet loss in the network, the window size shrinks to W_{min}. It then performs binary search to calculate its congestion window by moving rapidly between W_{max} and W_{min}. The maximum value that window can increase is I_{max} and the defined minimal distance to W_{max} is I_{min}. Max probing phase is to find a nearby new saturation point by slowly increasing the window size. Figure 1 shows the phases of additive increase, binary search and max probing. The good performance of BIC-TCP is due to the slow increase in the region of W_{max} and linear increment in the phase of additive increase and max probing. The pseudo code of BIC [23] algorithm is described in Algorithm 1.



Firgure 1. TCP-BIC window growth function.

Algorithm 1: Pseudo code of BIC

BIC_C_{wnd}(I_{max}, I_{min})
begin
Current congestion window is denoted by Wnd_c, target window as Wnd_t
Let $Wnd_c = 1$
#max_probing
if(ACK=yes) then Wnd_c=Wnd_c++
  else if (pckt_loss) then jump packet_loss
else jump max_probing to repeat the process
#packet_loss
$W_{max} = Wnd_c$
$Wnd_c = W_{max} / 2$
jump additive_increase
#additive_increase
if (ACK = yes) then
  if(($W_{max} – Wnd_c$) < I_{min}) then
  jump max_probing
$Wnd_t = (Wndc + W_{max}) / 2$
if($Wnd_t – Wnd_c > I_{max}$) then
  $Wnd_c = Wnd_c + I_{max}$
  jump additive_increase
else jump binary_search
if(pckt_loss) then jump packet_loss
else jump additive_increase
#binary_search
if("ACK = yes") then $Wnd_c = Wnd_t$
if($W_{max} – Wnd_c$ < I_{min}) then jump max_probing
else
  $Wnd_t = (Wnd_c + W_{max}) / 2$
  jump binary_search
if(pckt_loss) then jump packet_loss
else jump binary_search
end

Although BIC achieves stability and scalability in high-speed networks but it is not suitable for low speed networks. The CUBIC is the next version of BIC which uses BIC as base algorithm with some modification in congestion window. CUBIC growth

function is real time depending upon the two consecutive events rather than RTT [11]. It uses a CUBIC function of the elapsed time from the last congestion event. In mathematics, a cubic function is written as:

$$f(x) = ax^3 + bx^2 + cx + d \quad (1)$$

Equation of CUBIC window growth function:

$$W(t) = C(t - K)^3 + W_{max} \quad (2)$$

Where 'C' denotes a CUBIC parameter, 't' is the elapsed time from which the last packet has been lost. Finally, 'K' is the time period required to increase the window up to 'W_{max}' representing that there is no further packet loss and it is calculated by the following function:

$$K = \frac{\sqrt[3]{W \max \beta}}{c} \quad (3)$$

'β' is declared as a constant multiplicative decrease factor. After the congestion event in the network, the window continues to grow until W_{max}. After this it comes into probing state where it starts growing convexly. This process continues until next congestion event.

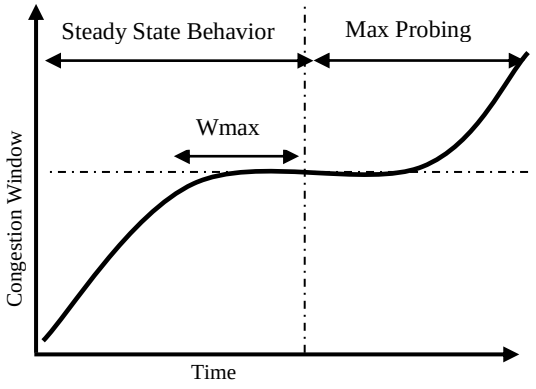


Figure 2. TCP-CUBIC window growth function.

Figure 2 depicts the CUBIC window growth function. At the initial stage the window grows very quickly up to the W_{max} and afterwards it slows down and increment almost becomes zero around W_{max}. After passing from this phase, it starts probing for more bandwidth and again the window grows slowly. This is very important feature of CUBIC that enhances the stability and efficient utilization of network bandwidth by the protocol around W_{max}. After passing W_{max}, it accelerates its window size away from the W_{max} ensuring the scalability factor of the protocol. Pseudo code of the CUBIC algorithm [2] is presented below.

Algorithm 2: Pseudo code of CUBIC

```
#initialization
last-max = 0, loss-cwnd=0;
Let epoch start (ep_s) as the time between two consecutive loss
events and sst as slow start threshold
ep_s=0, sst =100, b = 2.5, c = 0.4
#on acknowledgement
min_delay = min (min_delay, RTT)
if (Cwnd < sst) then Cwnd = Cwnd + 1
   else if (ep_s = 0) then ep_s = Ct (current time)
      K = max(0, 3√(b * (lastmax –Cwnd)))
      origin point(org_p) will be
      org_p = max(lastmax –Cwnd)
   end if
   t = Ct + (min_delay – ep_s)
   target = org_p + (c * (t-K)3)
   if(Cwnd < target) then cnt = Cwnd / (target – Cwnd)
   else cnt = Cwnd * 100
   end if
   if (min_delay > 0) then
      cnt = max (cnt, cwnd/(min_delay * 20) * 8)
   end if
   if (loss_ Cwnd = = 0) then cnt = 50
   end if
   if (Cwnd_cnt > cnt) then
      Cwnd++
      Cwnd_cnt = 0
   else Cwnd_cnt++
   end if
end if
#on packet loss
ep_s = 0
if Cwnd < lastmax then lastmax = Cwnd * 0.9
else lastmax = Cwnd
end if
loss_ Cwnd = Cwnd
Cwnd = Cwnd * 0.8
```

4. CUBIC in Linux Kernel

The Linux kernel code is embedded with the support of various portions of network stack to provide different kind of functionalities. One of these implementations is the code that relates to the description and implementation of TCP congestion control [4]. As discussed earlier, due to the unique features of CUBIC, it is implemented as default Linux kernel. Before moving to the CUBIC code in Linux kernel, it is very important to study the basic design of the kernel. The code is organized in different files and functions. The main and important files are listed in the directory "*/net/ipv4*". The main files that deal with TCP are presented in the Table 1.

Table 1. TCP files in Linux kernel.

File Name	Description
tcp.h	This is the header file including the definition of various functions.
tcp.c	Provides interface between different sockets and the rest of code.
tcp_input.c	It deals with all the incoming packets from the network. It is an important file.
tcp_output.c	It deals the outgoing packets to the network.
tcp_ipv4	IPv4 code to support TCP.
tcp_timer	It includes time management functions.
tcp_cong.c	Pluggable support for congestion control.
tcp_cubic.c	This is the file that includes the implementation detail of cubic in Linux.

The CUBIC framework achieves high level of abstraction by initializing its code and uses pluggable

support of different files. To initialize the various function calls, a static record of struct tcp_congestion_ops is stored in tcp_cubic.c.

The kernel initialization is performed by sysctl command. Tcp_register_congestion_control is used to register the algorithm with the system and is located in tcp_cong.c. The framework of CUBIC initialization in Linux kernel is shown in Figure 3.

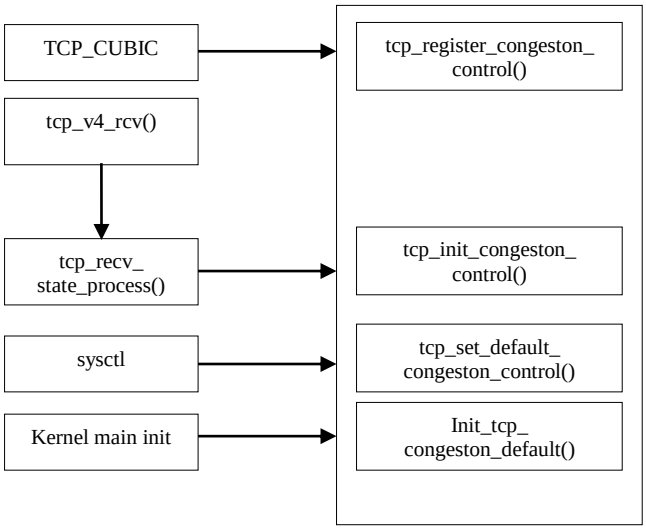


Figure 3. CUBIC initialization framework in Linux kernel.

Pluggable congestion module (tcp_congestion_ops) is used to allow different algorithms to load dynamically in Linux kernel and to be capable of switching between the algorithms without re-compiling the kernel. By the support of this module the new congestion algorithm requires only to deal with cong_avoid and ssthresh mechanisms to load itself in the kernel. CUBIC initiates itself to access the TCP code as follows:

```
static struct TcpCongestion_ops CubicTcp
    read_mostly =
{
    .init = BIC_Initialization,
    // slow start threshold 'sst'
    .sst = BIC_recalc_sst,
    .cong_avoid = BIC_cong_avoid,
    .set_state = BIC_state,
    .undo_Cwnd = BIC_undo_ Cwnd,
    .pkts_ACK = BIC_ACK,
    .owner = THIS_MODULE,
    .name = "CUBIC"
};
```

5. CUBIC in NS2 AND NS3

The NS2 [1] is written in C++ and it is a very powerful discrete event simulator. It is most commonly used simulator which supports a variety of applications including MAC, Network and Transport layer protocols (e.g., TCP CUBIC). The user interface of NS2 is an OTcl interpreter shell. Input model files are called Tcl scripts that are input to the OTcl for execution [10]. The network configuration is developed in the form of object oriented classes. The main advantage of NS2 is that it is freely distributed and mostly it is deployed on Linux distributions (e.g. Fedora, CentOS, Ubuntu etc.). The simulation is used for hypothesis testing and its results are very close to the real time implementations. The performance of network is measured over a variety of parameters known as performance metrics (e.g. throughput, delay, packet drop etc.). Trace file stores overall information of the simulation process and results. GNU Plot is used to show the results in the form of graphics. The block diagram of TCP CUBIC in NS2 is shown in Figure 4.

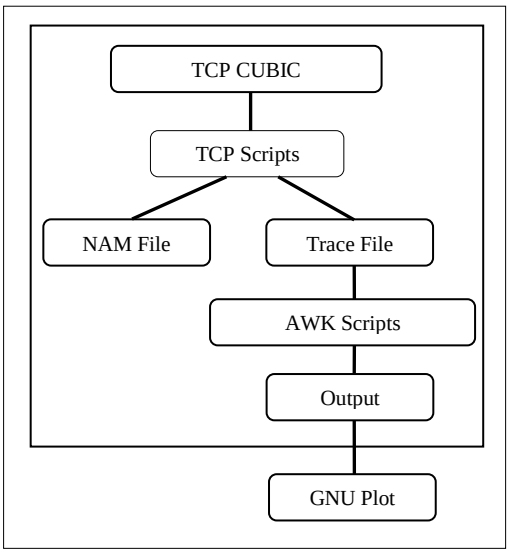


Figure 4. CUBIC block diagram in NS2.

NS3 [22] is designed to evaluate the performance of modern technologies (e.g., 4G LTE) and high speed networks. Its main objective was to improve the NS2 and enhance the support of different software modules. It is being adopted by many researchers all over the world in order to simulate a variety of network applications and real time systems. But if we discuss ns3 and its support in congestion control, it shows shortcomings in its infrastructure and support regarding various TCP options particularly window scaling and time stamping. All of the TCP variants supported in NS2 have not yet been ported in NS3. Currently, it includes TCP Tahoe, Reno, NewReno and Westwood. The class diagram of the TCP CUBIC implementation in NS3 is presented in Figure 5.

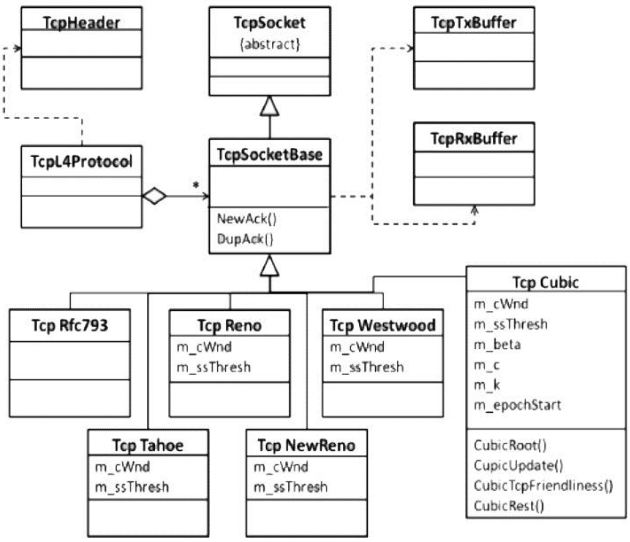


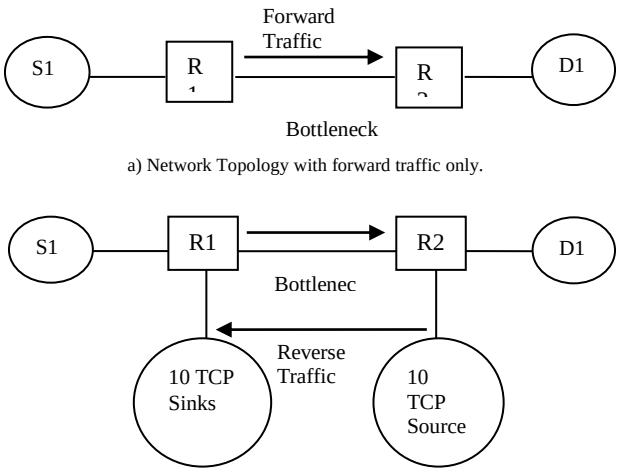
Figure 5. CUBIC block diagram in NS3 [1].

6. Methodology and Performance Analysis

This section will elaborate the performance of CUBIC, NewReno and Compound by using NS2. We shall compare the performance of the CUBIC in various scenarios including wired and wireless to study its behavior towards congestion control.

6.1. Wired Scenarios

First scenario is very simple in which two TCP nodes are connected through a dumbbell topology. The source node is represented by S1, destination node D1, while R1 and R2 are routers connecting S1 and D1 respectively. The link between the routers is bottleneck link of 2 Mbps. Source and destination nodes are connected to the routers with link speed of 1Gbps. The network diagram is shown in Figure 6-a.



a) Network Topology with forward traffic only.

b) Network topology with forward and reverse traffic.

Figure 6. Network traffic orientation.

The goodput of network is evaluated for CUBIC, NewReno and Compound by using the forward traffic. Further, the reverse traffic is also injected by 10 TCP sources on the same channel as used by the single TCP source as shown in Figure 6(b). Comparative analysis of goodput of these protocols is presented in Figure 7.

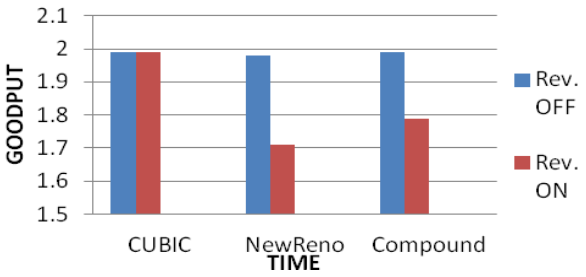


Figure 7. Goodput analysis of CUBIC, NewReno and Compound.

From Figure 7, it is clear that the goodput of all the three TCP protocols is almost same and reaches approximately up to the bottleneck capacity in the case of forward traffic only. But when the reverse traffic is on, the behavior of the protocols is totally changed. With the presence of reverse traffic, the goodput of CUBIC is best, while the NewReno shows the worst goodput. However, TCP Compound has comparatively better goodput than the NewReno.

In the second wired scenario, we increase the number of connecting devices up to 'n' nodes ranging from S_0 to S_n where 'n' denotes the total number of nodes. The same number of nodes are on the destination side i.e. D_0 to D_n. The participating nodes are sharing a link of 10 Gbps. The main objective of this scenario is to calculate the intra-protocol fairness. The value of 'n' ranges from 20 to 200 in order to evaluate fairness with respect to RTT. The simulation time is set to 1000s. The reverse traffic is injected as 10 TCP sources on the same link to induce ACK compression. The network topology is shown in Figure 8 and the results are shown in Figure 9.

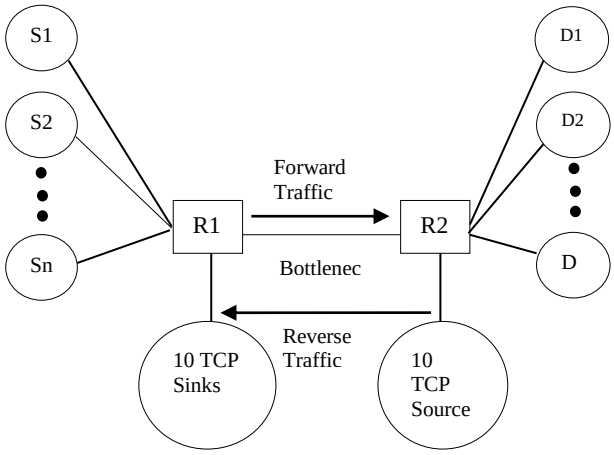


Figure 8. Multiple connections with single bottleneck scenario

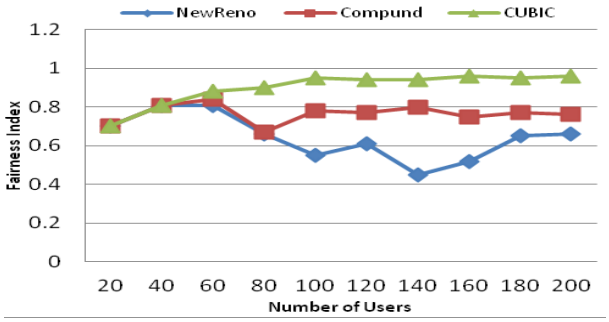


Figure 9. TCP intra-protocol fairness (wired).

As shown in Figure 9, it was observed that TCP CUBIC improves overall more fairness in bandwidth sharing as compared to Compound and NewReno. Both NewReno and Compound show almost the same fairness index at the start (where n < 70). But as number of nodes increase, Compound shows better performance than NewReno due to the fact that Compound reduces its window size in response to delay. The CUBIC achieves best fairness due to its ideal window increase function. It does not depend upon the receipt of ACKs rather than it increases its window only at the occurrence of last congestion.

6.2. Wireless Scenario:

In this scenario, we modify the network topology as discussed in Figure 8. We simply add wireless segment on the receiving side to introduce high delays as shown in Figure 10.

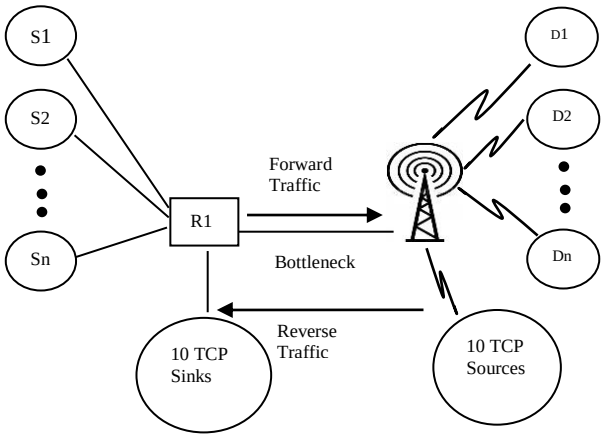
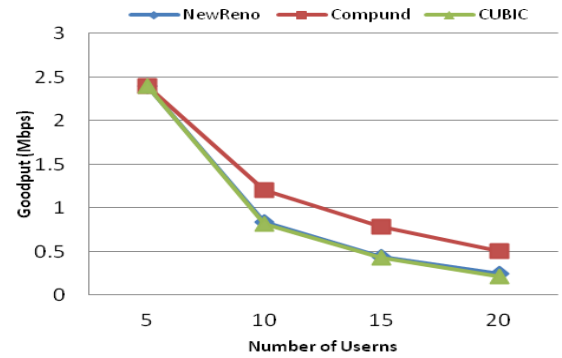
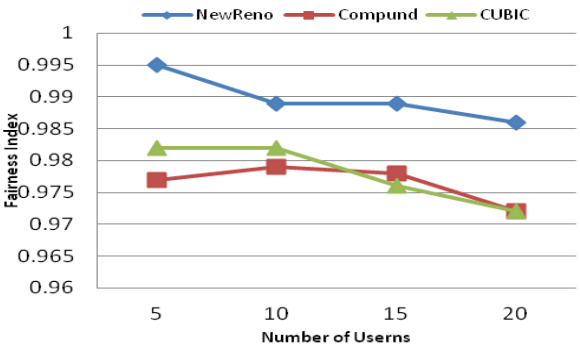


Figure 10. Multiple connections wireless scenario.

The ‘n’ numbers of nodes are connected at wired portion of the network as the source nodes. Similarly, on the destination side ‘n’ number of nodes are connected but the link is wireless on destination side. In order to induce reverse traffic, ten TCP sources are connected at the wireless side and ten TCP sinks are connected at the wired side of the network. The wired link between router and wired devices is 1Gbps while the link between base-station and mobile nodes is 54 Mbps. The mobile devices operate at IEEE 802.11g protocol. The simulation time is set to 1000s. The simulation results regarding the goodput and intra-protocol fairness are shown in Figure 11-a and 11-b respectively.



a) TCP goodput (wireless).



b) TCP intra-protocol fairness (wireless).

Figure 11. The simulation results.

The goodput and intra-protocol fairness has been evaluated for the three protocols. It is observed that the Compound has the better goodput as compared to CUBIC and NewReno which achieves almost the similar goodput. The goodput of the protocols significantly decreases with the increase in number of users. It is very clear that all the three variants attain very poor goodput although the wireless link capacity is 54 Mbps. The reason is that reverse traffic is causing delays in ACKs which affects the performance of these protocols. In case of intra-protocol fairness, NewReno performs better than CUBIC and Compound. CUBIC has overall worst performance in wireless scenario.

7. Conclusions

In this paper we have discussed TCP CUBIC in detail in various aspects including its design architecture, CUBIC support in Linux kernel, CUBIC in network simulators (NS2 & NS3). At the end CUBIC has been evaluated for goodput and inter-protocol fairness both in wired and wireless environments along with two other variants TCP Compound and NewReno. Simulation results prove that CUBIC is excellent in high speed wired environment and achieves better level of goodput and fairness as compared to other two variants. The goodput of CUBIC is not affected in wired network in the presence of reverse traffic. On the other hand, in wireless network CUBIC has

performance issues. It does not achieve goodput and better level of fairness.

This gives us an open issue regarding the poor performance of CUBIC in wireless or low speed networks. Further study may include improving the performance of CUBIC in wireless networks as it is very suitable for high speed and scalable networks.

References

[1] A Discrete-Event Network Simulator, ns-3 simulator Tutorial, https://www.nsnam.org/docs/tutorial/html, last Visited, 2016.

[2] Ahmad M., Ngadi A., Nawaz A., Ahmad U., Mustafa T., and Raza A., "A Survey on TCP CUBIC Variant Regarding Performance," *in Proceedings of 15th International Multitopic Conference*, Islamabad, pp. 409-412, 2012.

[3] Ahmad N., Shoaib U., and Prinetto P., "Usability of Online Assistance from Semiliterate Users' Perspective," *International Journal of Human-Computer Interaction*, vol. 31, no. 1, pp. 55-64, 2015.

[4] Arianfar S., "TCP's Congestion Control Implementation in Linux Kernel," *in Proceedings of Seminar on Network Protocols in Operating Systems*, pp. 1-6. 2012.

[5] Bisen D. and Sharma D., "Improve Performance of Tcp New Reno over Mobile Ad-Hoc Network Using Abra," *International Journal of Wireless and Mobile Networks*, vol. 3, no. 2, pp. 102-111, 2011.

[6] Brakmo L. and Peterson L., "TCP Vegas: End to End Congestion Avoidance on A Global Internet," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, pp. 1465-1480, 1995.

[7] Caini C. and Firrincieli R., "TCP Hybla: A TCP Enhancement for Heterogeneous Networks," *International Journal of Satellite Communications and Networking*, vol. 22, no. 5, pp. 547-566, 2004.

[8] Casetti C., Gerla M., Mascolo S., Sanadidi M., and Wang R., "TCP Westwood: End-to-End Congestion Control for Wired/Wireless Networks," *Wireless Networks*, vol. 8, no. 5, pp. 467-479, 2002.

[9] Floyd S., "High Speed TCP for Large Congestion Windows," Network Working Group, 2003.

[10] Gharge S. and Valanjoo A., "Simulation Based Performance Evaluation of TCP Variants and Routing Protocols in Mobile Ad-Hoc Networks," *IEEE International Conference on Advances in Engineering and Technology Research*, Unnao, pp. 1-8, 2014.

[11] Ha S., Rhee I., and Xu L., "CUBIC: A New TCP-Friendly High-Speed TCP Variant," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 64-74, 2008.

[12] Henderson T., Floyd S., Gurtov A., and Nishida Y., "The Newreno Modification to TCP's Fast Recovery Algorithm" Network Working Group, 2012.

[13] Irfan M., Oriat C., and Groz R., "Model Inference and Testing," *Advances in Computers*, vol. 89, pp. 89-139, 2013.

[14] Jacobson V., "Berkeley TCP Evolution From 4.3-Tahoe to 4.3-Reno," *in Proceedings of the 18th Internet Engineering Task Force*, Vancouver, 1990.

[15] Kozu T., Akiyama Y., and Yamaguchi S., "Improving Rtt Fairness on Cubic Tcp," *in Proceedings of 1st International Symposium on Computing and Networking*, Matsuyama, pp. 162-167, 2013.

[16] Kumari D., Tahiliani M., and Shenoy U., "Experimental Analysis of CUBIC TCP in Error Prone Manets," *in Proceedings of 5th International Conference on the Applications of Digital Information and Web Technologies*, Bangalore, pp. 256-261, 2014.

[17] Liaqat M., Chang V., Gani A., Hamid H., Toseef M., Shoaib U., and Ali R., "Federated Cloud Resource Management: Review and Discussion," *Journal of Network and Computer Applications*, vol. 77, pp. 87-105, 2017.

[18] Liu S., Başar T., and Srikant R., "TCP-Illinois: A Loss-And Delay-Based Congestion Control Algorithm for High-Speed Networks," *Performance Evaluation*, vol. 65, no. 6, pp. 417-440, 2008.

[19] Monowar M., Rahman O., Pathan A., and Hong C., "Prioritized Heterogeneous Traffic-Oriented Congestion Control Protocol for Wsns," *The International Arab Journal of Information Technology*, vol. 9, no. 1, pp. 39-48, 2012.

[20] Prinetto P., Shoaib U., and Tiotto G., "The Italian Sign Language Sign Bank: Using Wordnet for Sign Language Corpus Creation," *in Proceedings of International Conference on Communications and Information Technology*, Aqaba, pp. 134-137, 2011.

[21] Rahman A., Sarfraz S., Shoaib U., Abbas G., and Sattar M., "Cloud based E-Learning, Security Threats and Security Measures," *Indian Journal of Science and Technology*, vol. 9, no. 48, 2016.

[22] Riley G., and Henderson T., "The Ns-3 Network Simulator," *in Proceedings of Modeling and Tools for Network Simulation*, Berlin, pp. 15-34, 2010.

[23] Šošić M. and Stojanović V., "Resolving Poor TCP Performance on High-Speed Long Distance Links-Overview and Comparison of BIC, CUBIC and Hybla," *in Proceedings of IEEE 11th*

International Symposium on Intelligent Systems and Informatics, Subotica, pp. 26-28, 2013.

[24] Tan K., Song J., Zhang Q., and Sridharan M., "A Compound TCP Approach For High-Speed and Long Distance Networks," *in Proceedings of IEEE INFOCOM 25th IEEE International Conference on Computer Communications*, Barcelona, 2006.

[25] Wang Z. and Crowcroft J., "Eliminating Periodic Packet Losses in the 4.3-Tahoe BSD TCP Congestion Control Algorithm," *ACM SIGCOMM Computer Communication Review*, vol. 22, no. 2, pp. 9-16, 1992.

[26] Wei D., Jin C., Low S., and Hegde S., "FAST TCP: Motivation, Architecture, Algorithms, Performance," *IEEE/ACM Transactions on Networking*, vol. 14, no. 6, pp. 1246-1259, 2006.

[27] Xu L., Harfoush K., and Rhee I., "Binary Increase Congestion Control For Fast Long-Distance Networks," *in Proceedings of IEEE INFOCOM 23rd Annual Joint Conference on Computer and Communications Societies*, Hong Kong, pp. 2514-2524, 2004.

Abrar Khan received his master degree in Information Technology from University of the Punjab, Gujranwala Campus and M.Phil (IT) from University of Gujrat, Pakistan. His research interests include TCP variants, cloud computing and big data analysis.

Umar Shoaib did his PhD in Department of Computer and Control Engineering Politecnico di Torino, Italy. His current research interests include Machine Learning, Robotics, Artificial Intelligence, Scalable Networks, Cloud Computing, Natural Language Processing, Text mining and Internet of Things.

Muhammad Sarfraz received his Ph.D in Remote Sensing & GIS from Asian Institute of Technology, Thailand. He is Associate Professor at University of Gujrat, Pakistan. He is members of numerous international societies like Telecoms Sans Frontiers for emergency response in Asia-Pacific, ISPRS Health and Geological Society of America. His research interests include Remote sensing, Geospatial analysis, Scalable networks and Digital image processing.