**Implementation of TCP Elastic using Network Simulator**

**Abstract**

In modern day networking, Transmission Control Protocol (TCP) congestion control is one of the key mechanisms that ensures the stability of the Internet. It is a popular research topic whenever TCP and its variants are studied. Open source network simulators such as Network Simulator -3 (NS-3) [2] are amazing tools used to gain important insight into existing TCP congestion control algorithms as well as develop new ones. We analyze one of the existing TCP congestion control algorithm CUBIC [3]. After this we have implemented Elastic TCP [5] on the network simulator NS-3. Lastly, we propose a new and efficient congestion control algorithm and implement it on NS-3.

Through our study, we highlight the key features of each algorithm.

i

**Table of Contents**

Page No.

**Abstract** i

**List of Figures** iii

**Abbreviations and Nomenclature** iv

**1: Introduction** 1

1.1 General 1

**2: Background Study** 2

2.1 TCP congestion control algorithms 2

2.1.1 CUBIC 3

2.1.2 Elastic TCP 4

2.1.3 Elmod 5

**3: Requirement Analysis** 7

3.1: Software Requirements 7

3.2: Hardware Requirements 7

3.3: Functional Requirements 7

3.4: Non-Functional Requirements 7

3.5: User Requirements 7

**4: Detailed Design** 8

4.1: Topology 8

4.2: Implementation 9

**5: Results** 10

5.1: Congestion window vs Time Plots 9

5.2: Congestion window, ACK and 13

RTT plots.

5.3: Comparative Analysis 16

**6: Conclusion** 17

**7: Future Scope** 17

**8: References** 18

ii

**List of Figures**

**Figure Title Page**

4.1 Topology 8

5.1(a) Congestion window vs time Plot for C = 0.2 10

5.1(b) Congestion window vs time Plot for C = 0.4 10

5.1(c) Congestion window vs time Plot for C = 0.6 11

5.1(d) Congestion window vs time Plot for C = 0.8 11

5.1(e) Congestion window vs time Plot for C = 1 11

5.1(f) Congestion window vs time Plot for Elastic 12

5.1(g) Average Throughput for Elastic 12

5.2(a) Congestion window (segment), ACK (segment), 13

RTT(s) vs Time(s) plot for Cubic.

5.2(b) Congestion window (segment), ACK (segment), 13

RTT(s) vs Time(s) plot for Elastic.

5.2(c) Congestion window (segment), ACK (segment), 14

RTT(s) vs Time(s) plot for New Reno.

5.2(d) Congestion window (segment), ACK (segment), 14

RTT(s) vs Time(s) plot for Veno.

5.2(e) Congestion window (segment), ACK (segment), 15

RTT(s) vs Time(s) plot for Elmod.

5.2(f) Congestion window (segment), ACK (segment), 15

RTT(s) vs Time(s) plot for Westwood.

5.3(a) Bar Graph for comparative analysis 16

(Cubic, Elastic, Veno, New Reno, Elmod)

iii

**Abbreviations and Nomenclature**

TCP Transmission Control Protocol

UDP User Datagram Protocol

Ns-3 Network Simulator -3

IPv4 IP Addressing version 4

IP Internet Protocol

ACK Acknowledgement

RFC Request for Comments

BIC Binary Increase Congestion control

RTT Round Trip Time

MSS Maximum Segment Size

BDP Bandwidth Delay Product

iv

**1: Introduction**

World of Internet is growing as we speak. Nowadays, 4G networks are becoming famous for which higher bandwidth and lower latency are being used. Applications like video streaming in 4G increase traffic in network very fast. With the modern advancements in networking, comes the emergence of new network environments such as Gigabit Ethernet or satellite links with challenging characteristics: high bit-error rate, long propagation delay, high link capacity, and asymmetric channels. Therefore congestion control is important. Optimization will increase the throughput, efficiency and downloading speed which is required by operators as well as customers.

Network congestion may occur when a sender overflows the network with too many packets. During congestion, the network cannot handle this traffic properly, which results in a poor quality of service. The typical symptoms of congestion are: excessive packet delay, packet loss and retransmissions. Transport Layer’s TCP congestion control techniques prevent congestion or help mitigate the congestion after it occurs. Standard TCP has been enhanced for the Future Internet resulting in many variants such as New Reno, Veno, and CUBIC etc.

The study of existing congestion control algorithms and the development of any new enhancements gain substantial benefits through the use of open-source network simulators such as NS-3. We implement Elastic TCP and study CUBIC in detail. This project presents our implementation details of these algorithms.

**2: Background Study**

This section provides the theoretical background of congestion control algorithms implemented in our project including CUBIC, Elastic TCP and Elmod\*.

**2.1: TCP Congestion Control**

TCP congestion control [1] algorithms may be loss based, delay based or hybrid based. Loss based algorithms treat the occurrence of a packet loss as an indication of congestion. Delay-based algorithms work on the basis of queuing delays. Hybrid algorithms take advantage of both loss and delay-based mechanisms. Most TCP congestion control algorithms are derivatives of the standard defined in RFC 5681[1], known as the Reno algorithm. The given four standard algorithms play a vital role in prevention of congestion: slow start, congestion avoidance, fast retransmit and fast recovery. The implementation of these algorithms requires following variables: cwnd, rwnd, and ssthresh. Congestion window (cwnd) determines the amount of data a sender can send before it receives an ACK to prevent network overflow. The receiver window (rwnd) indicates the amount of data a receiver is able to accept. Slow start threshold (ssthresh) provides the transition point between slow start and congestion avoidance phases.

* **Slow start** comes into action when too much data is sent to a network and the network is not able to process that data thus resulting in congestion. During slow start, cwnd is incremented by 1 for every ACK received, resulting in an exponential increase of the sending rate until a loss occurs as shown in Equation 1.

cwnd = cwnd + 1 (1)

* **Congestion avoidance** algorithm is started when ssthresh is reached and cwnd > ssthresh. The cwnd is then incremented by 1 for every RTT (round trip time), resulting in a linear increase until a loss occurs. Refer Equation 2.

cwnd = cwnd + 1/cwnd (2)

When the segment acknowledgements are not received, ssthresh is set to half of the current cwnd size.

* **Fast retransmit** algorithm is responsible for detecting and recovering lost data by observing the number of duplicate ACKs received, with the arrival of three duplicate ACKs signifying a packet loss. At this moment TCP makes cwnd = 1 and sets ssthresh = cwnd/2.
* **Fast recovery** governs data transmission after fast retransmit until a new ACK arrives informing the recovery of the loss. During a loss, Reno halves its slow-start threshold and set sending rate according to Equations 3 and 4, respectively.

ssthresh = cwnd/2 (3)

cwnd = ssthresh + 3 (4)

**2.1.1: CUBIC**

As the name represents, the window growth function of CUBIC [3] is a cubic function. There are two components of window growth, the first is a concave part where the window size quickly increases before the last congestion event. Then comes the convex part where CUBIC probes for more bandwidth. The congestion window of CUBIC is calculated by🡪 Wcubic = C (t − K)^3 +Wmax. Here C is a scaling factor, t is the elapsed time from the last window reduction, Wmax is the window size just before the last window reduction and K = (Wmax\*β/ C)^1/3, where β is the multiplicative decrease factor, used for reducing cwnd at the time of loss. If cwnd is less than the window size that TCP would reach at time t after the last loss event, then CUBIC is in the TCP. Otherwise, if cwnd is less than Wmax, then CUBIC is in the concave region, and if cwnd is larger than Wmax, CUBIC is in the convex region.

TCP-friendly region: - On receiving an ACK in congestion avoidance phase, we check whether the protocol is in the TCP region or not. We analyze the window size of TCP in terms of the elapsed time t. Using a simple analysis we can find the average window size of additive increase and multiplicative decrease (AIMD) with an additive factor α and a multiplicative factor β to be the following function:

1/ RTT ((α\*2 – β)/2βp)^1/2 (5)

Concave region: - On receiving an ACK in congestion avoidance, if the protocol is not in the TCP mode and cwnd is less than Wmax, then the protocol is in the concave region. In this region, cwnd is incremented by

(W(t+RTT )−cwnd )/cwnd (6)

Convex region: - When the window size of CUBIC is larger than Wmax, it follows the convex profile of the cubic function. Since cwnd is larger than the previous Wmax, this shows that the network conditions might have been changed since the last loss occurred, possibly implying more available bandwidth after some flow departures.

Multiplicative decrease: - When a packet loss occurs, CUBIC reduces its window size by a factor of β. We set β to 0.8.

3

Fast convergence*: -* The protocol remembers the last value of Wmax before it updates Wmax for the current loss event. At a loss event, if the current value of Wmax is less than the last value of it, this indicates that the saturation point experienced by this flow is getting reduced because of the change in available bandwidth. Hence Wmax is reduced further.

**2.1.2: Elastic TCP**

Elastic TCP [5] is ideal for high-Bandwidth Delay Product (BDP) networks (networks having large RTTs and large bandwidths) to improve the utilization of bandwidth. It is an RTT independent and delay based congestion control algorithm.

**Initialization**

RTTmax 🡨 0, RTTcurr 🡨 0, RTTbase 🡨 0x7FFFFFFF, Cwnd 🡨 2;

**On** receiving **ACK**:

**If** Not a Duplicate ACK **then**

**If** Slow start **then**

Cwnd 🡨 Cwnd+1;

**else**

RTTcurr 🡨 (now – sendtime);

**If** RTTcurr < RTTbase **then**

RTTbase 🡨 RTTcurr;

**end**

**If** RTTcurr > RTTmax **then**

RTTmax 🡨 RTTcurr;

**end**

WWF 🡨 ((RTTmax/RTTcurr)\*Cwnd)^1/2;

Cwnd 🡨 Cwnd + WWF/Cwnd;

**end**

**else**

Multiplicative Decrease

**end**

**Algorithm -1**: - The pseudocode of Elastic-TCP [5]

4

Initially it works just like standard slow start algorithm. When a duplicate ACK arrives/Timer timeout, it reduces its congestion window using multiplicative decrease factor *B*(beta) and enters into congestion avoidance stage. Now it increases its congestion window by (WWF/cwnd), where WWF stands for Window-correlated Weighting Function. Here WWF plays a major role as it improves the ability of Elastic to deal with high delays, long BDP networks and big buffers.

Elastic-TCP improves the total performance in terms of delay, average throughput and loss as compared to other TCP variants like NewReno, Veno, Vegas and even CUBIC.

**2.1.3: Elmod (New Algorithm)**

TCP Elmod too is ideal for High Bandwidth Delay Product networks. It is an RTT-independent delay based congestion control algorithm.

It is based on TCP Elastic. It starts with standard slow start algorithm. When a duplicate ACK arrives/Timer timeout, it reduces its congestion window using multiplicative decrease factor B(beta) and enters congestion avoidance phase. Now it increases its congestion window by (WWF/Cwnd\_next).

* But unlike Elastic, it stores the value of the last congestion window when the congestion occurs.
* Current congestion window then increases aggressively (concave region) till its value reaches the value of last congestion window.
* After that, the congestion window is increased in an additive manner as the congestion may occur again.

Due to exponential increase of congestion window in early stages, Elmod performs better then Elastic for high BDP networks.

Refer to the Algorithm below:-

5

**Initialization**

RTTmax 🡨 0, RTTcurr 🡨 0, RTTbase 🡨 0x7FFFFFFF, Cwnd 🡨 0,

LastCwnd 🡨 0;

**On** receiving **ACK**:

**If** Not a Duplicate ACK **then**

**If** Slow start **then**

Cwnd 🡨 Cwnd+1;

**else**

Cwnd\_next = Cwnd;

RTTcurr 🡨 (now – sendtime);

**If** RTTcurr < RTTbase **then**

RTTbase 🡨 RTTcurr;

**end**

**If** RTTcurr > RTTmax **then**

RTTmax 🡨 RTTcurr;

**end**

WWF 🡨 ((RTTmax/RTTcurr)\*Cwnd)^2/3;

Cwnd\_next 🡨 Cwnd\_next + WWF/Cwnd\_next;

**If** Cwnd\_next < LastCwnd **then**

Cwnd 🡨 Cwnd\_next;

**else**

Cwnd 🡨 Cwnd + 1/Cwnd; //Additive increase

**end**

**end**

**else if** //Loss occurs

LastCwnd 🡨 Cwnd;

Multiplicative Decrease.

**end**

**end**

**Algorithm -2: -** The pseudocode of Elmod

6

**3: Requirement Analysis**

Requirement analysis is the process of determining user expectations for a new or modified product/software.

**3.1: Software Requirements**

* Language C++
* Network Simulator NS-3 (version 3.30)
* Linux Ubuntu or any other distribution

**3.2: Hardware Requirements**

* Processor Intel i3 or above
* RAM 4GB or above

**3.3: Functional Requirements**

**3.3.1: Design and Implementation Constraints**

This project is constrained by the number of TCP variants available for implementation on NS-3. Also NS-3 simulation is limited to LINUX kernel only.

**3.3.2: External Interface Requirements**

**User Interface**

* Front-end software – Network simulator (NS-3)
* NetAnim

**Hardware Interface**

* Waf compilation tool

**Software Interface**

* This project requires network simulator NS3 version 3.30.

**3.4: Non-Functional Requirements**

**3.4.1: Performance Requirements**

Depending on the Linux distribution and RAM of the system, the simulation will take no more than 10 sec to compile. Once compiled, the program will not take more than 2sec to run and show simulation results.

**3.4.2: Availability**

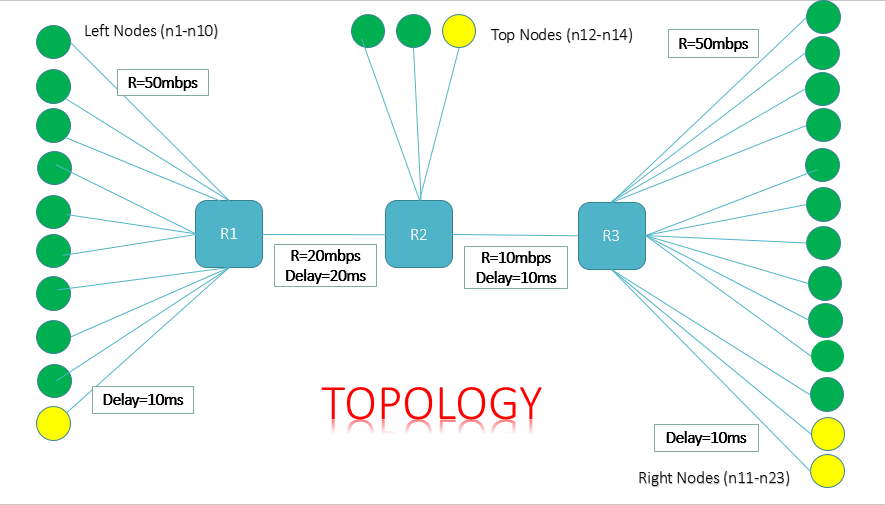
Users can use the program in Linux kernel with ns3 being properly installed.

**3.4.3: Security**

No security limitations since no user personal information is required.

**4: Detailed Design**

**4.1: Topology**



**Fig 4.1** – Figure depicts the Network Topology used for performance evaluation in this work. Green nodes refer to TCP sources and sinkers while the yellow ones are UDP sources and sinkers. All the nodes are connected via peer-to-peer architecture.

**Initial Simulation Parameters Setting**

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Value(s) | Parameter | Value(s) |
| No. of packets sent | 100000 | Bottleneck1 bandwidth | 20Mbps |
| Data packet size | 1040 bytes | Bottleneck1 delay | 20ms |
| Simulation Time | 200ms | Bottleneck2 bandwidth | 10Mbps |
| Nodal bandwidth | 50Mbps | Bottleneck2 delay | 10ms |
| Nodal delay | 10ms | Queue management | FqCoDel |

**4.2: Implementation**

**Step1**- 10 Left nodes (9 TCP and 1 UDP) start sending data in packets to router R1 at time t=1ms.

**Step2**- The router R1 push these packets to R2. At router R2, 3 Top nodes (2 TCP and 1 UDP) add data and send these packets to router R3.

**Step3**- Finally R3 send these data packets to destination sinkers (Right Nodes). We have 13 Right nodes with 11 TCP sinkers and 2 UDP sinkers. Sinkers start at time t=0.

Since bandwidth of R1-R2 and R2-R3 are less than that of the source nodes, congestion takes place. Different algorithms implemented here tackle this congestion in different ways.

9

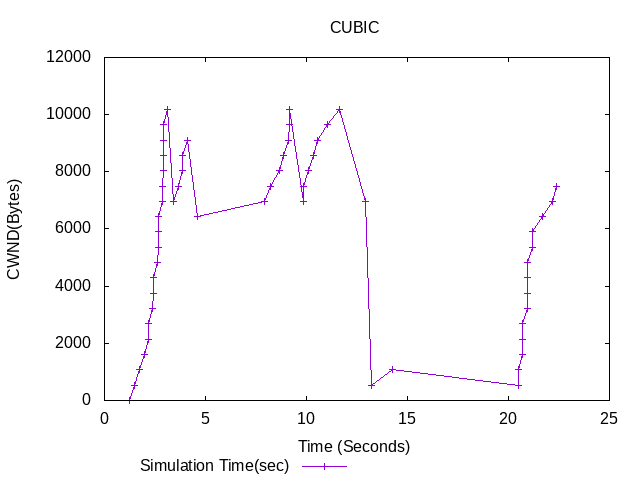
**5: Results**

We successfully implemented Elastic TCP on network simulator NS-3.

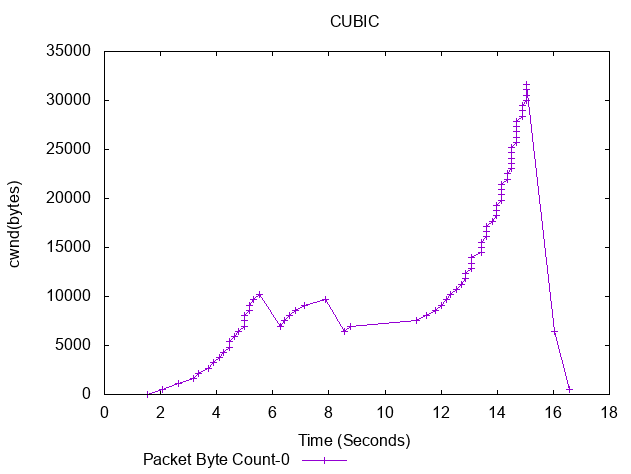
**5.1: Congestion window vs Time Plots**

Successful implementation of CUBIC.

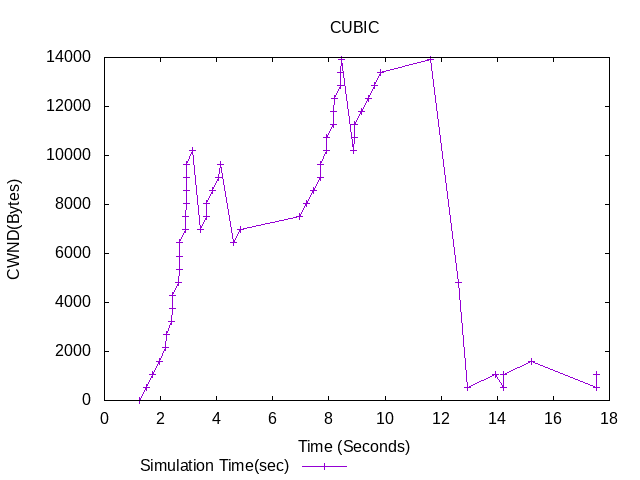
Range of CUBIC scaling factor C [4] (0.2 to 1).



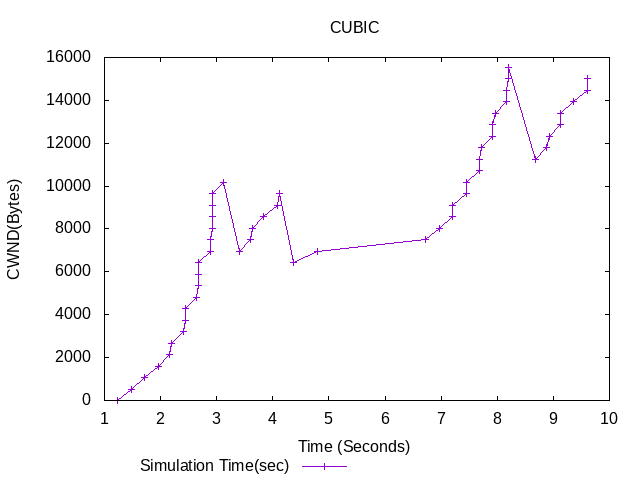
**Fig 5.1(a) –** Cwnd(Bytes) vs Time(s) Plot for C = 0.2 (CUBIC)



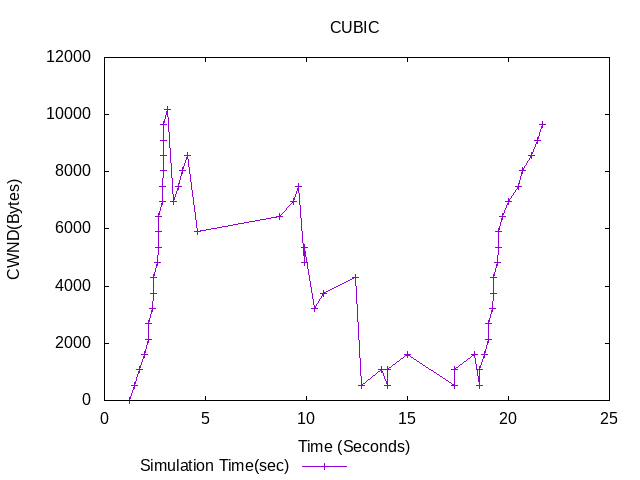
**Fig 5.1(b) –** Cwnd(Bytes) vs Time(s) Plot for C = 0.4 (CUBIC)



**Fig 5.1(c) –** Cwnd(Bytes) vs Time(s) Plot for C = 0.6 (CUBIC)

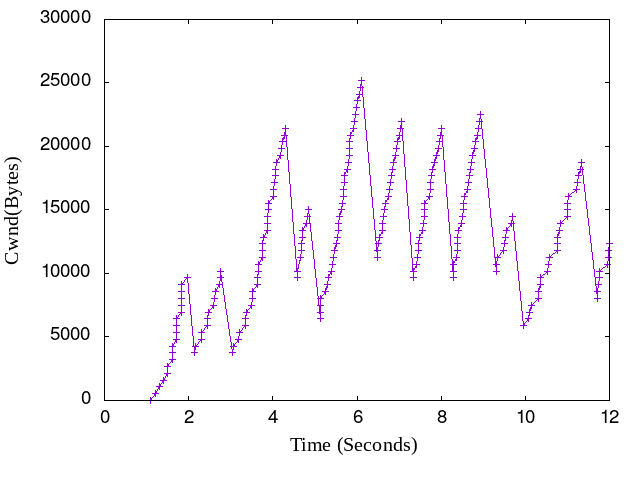


**Fig 5.1(d) –** Cwnd(Bytes) vs Time(s) Plot for C = 0.8 (CUBIC)

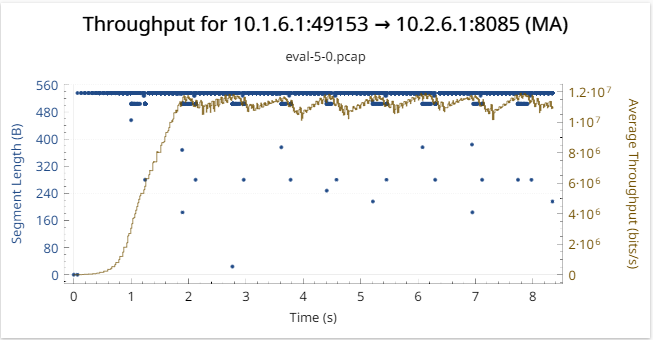


**Fig 5.1(e) –** Cwnd(Bytes) vs Time(s) Plot for C = 1 (CUBIC)

11



**Fig 5.1(f) –** Cwnd(Bytes) vs Time(s) Plot for Elastic-TCP

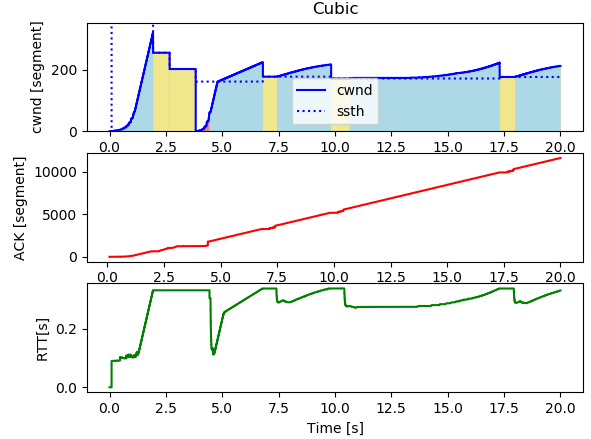


**Fig 5.1(g)** – Average Throughput for Elastic. X-axis 🡪Segment Length (Bytes),

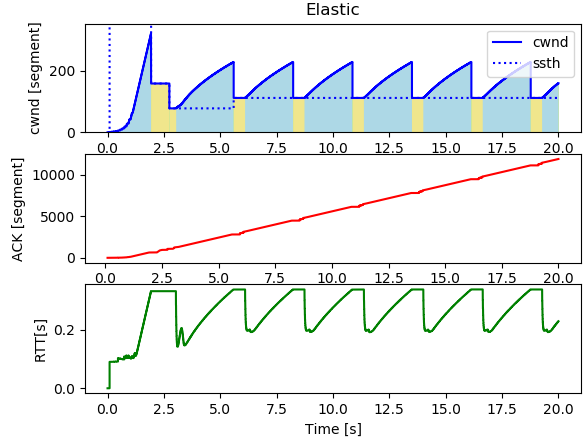
Y-axis🡪 Time(s).

12

**5.2: Congestion window, ACK and RTT plots.**

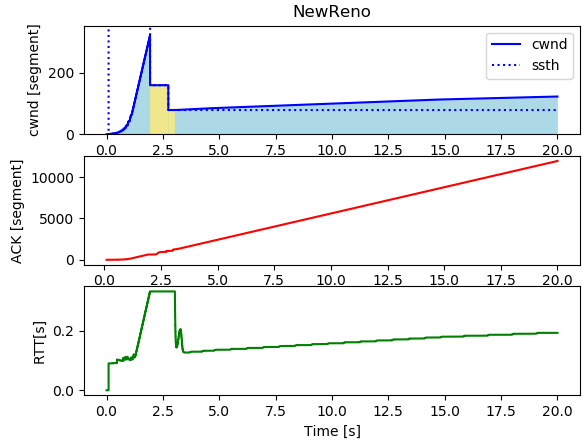


**Fig 5.2(a)**- Congestion window(segment) vs time(s) , ACK(segment) vs time(s) and RTT(s) vs time(s) plot for Cubic.



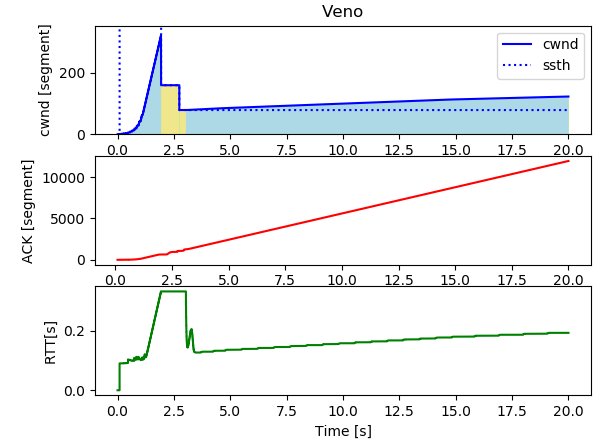
**Fig 5.2(b)**- Congestion window(segment) vs time(s) , ACK(segment) vs time(s) and RTT(s) vs time(s) plot for Elastic.

13



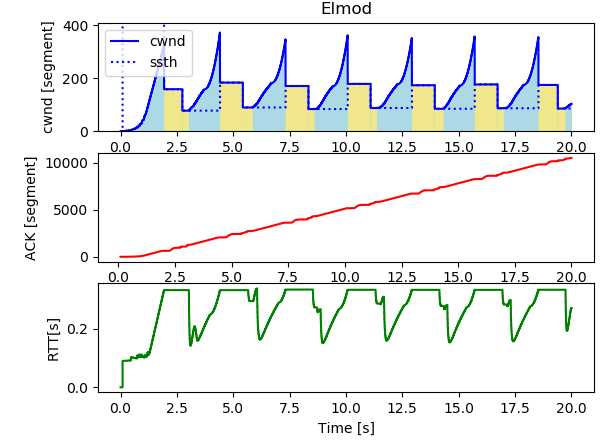
**Fig 5.2(c)**- Congestion window(segment) vs time(s) , ACK(segment) vs time(s)

and RTT(s) vs time(s) plot for New Reno.



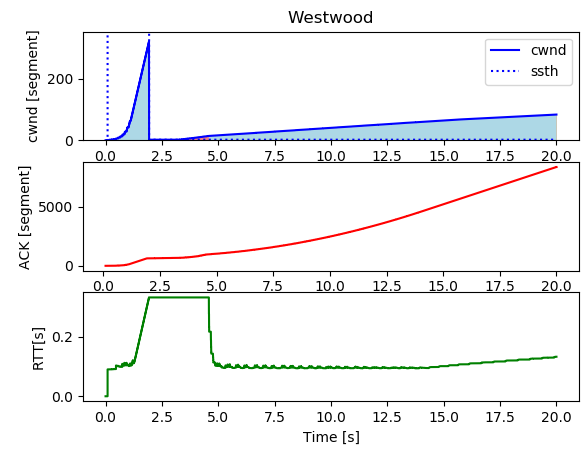
**Fig 5.2(d)-** Congestion window(segment) vs time(s) , ACK(segment) vs time(s) and RTT(s) vs time(s) plot for Veno.

14



**Fig 5.2(e)**- Congestion window(segment) vs time(s) , ACK(segment) vs time(s)

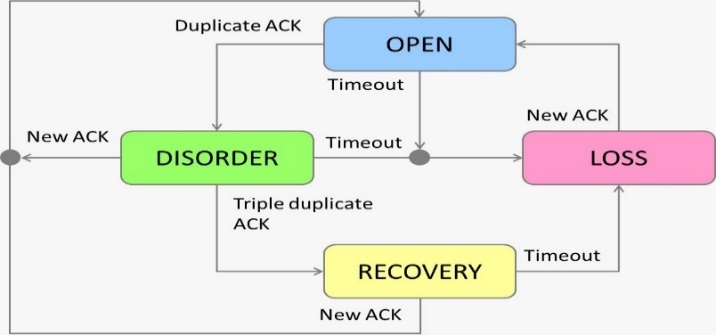
and RTT(s) vs time(s) plot for Elmod.



**Fig 5.2(f)**- Congestion window(segment) vs time(s) , ACK(segment) vs time(s)

and RTT(s) vs time(s) plot for Westwood.

15



OPEN – Normal State, no dubious events. [6]

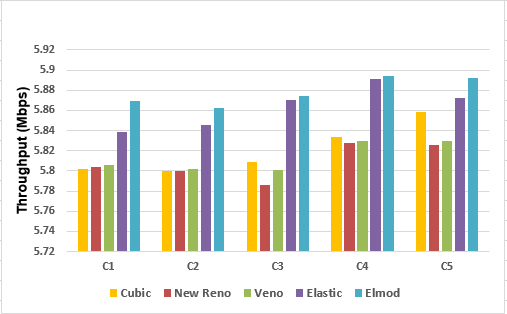
DISORDER – Some SACKs or duplicate ACKs received.

RECOVERY – Triple duplicate ACK, Cwnd was reduced.

LOSS – When timeout or ACK reneging.

**5.3:** **Comparative Analysis:-**

Throughput is the aggregate data that is successfully received to all the network terminals. It is an important performance indicator and for maximum throughput, both the connection ends (sender and receiver) should utilize maximum link capacity.Throughput = Sum of all bits received / connection time



**Fig 5.3(a)** - Graph depicts the value of maximum throughput for TCP Cubic, New Reno, Elastic, Elmod and Veno based on different values of nodal and bottleneck bandwidths. Delay 10ms for all cases.

16

**C1** - For Rate = 200Mbps, bt1 1000 Mbps and bt2 800 Mbps, Elmod gave best average throughput. It performed 0.511% better than Elastic, 1.142% better than Cubic, 1.073% better than Veno and 1.107% better than New Reno.

**C2** - For Rate = 500Mbps, bt1 1200 Mbps and bt2 1000 Mbps, Elmod gave best average throughput. It performed 0.273% better than Elastic, 1.058% better than Cubic, 1.024% better than Veno and 1.058% better than New Reno.

**C3** - For Rate = 1500Mbps, bt1 1800 Mbps and bt2 1200 Mbps, Elmod gave best average throughput. It performed 0.068% better than Elastic, 1.107% better than Cubic, 1.242% better than Veno and 1.498% better than New Reno.

**C4** - For Rate = 3000Mbps, bt1 3600 Mbps and bt2 2400 Mbps, Elmod gave best average throughput. It performed 0.0508% better than Elastic, 1.017% better than Cubic, 1.085% better than Veno and 1.119% better than New Reno.

**C5** - For Rate = 4000Mbps, bt1 8000 Mbps and bt2 6000 Mbps, Elmod gave best average throughput. It performed 0.339% better than Elastic, 1.577% better than Cubic, 1.052% better than Veno and 1.120% better than New Reno.

**6: Conclusion**

In this project, we have successfully implemented TCP Elastic on network simulator NS-3.Furthermore we were able to device a new efficient congestion control algorithm Elmod.

We observed that Elastic performs better than other TCP variants for high BDP networks. Elmod performs slightly better than Elastic. Also, Cubic performs like standard TCP under lower bandwidth networks.

**7: Future Scope**

* Enhancement of Elmod by increasing/decreasing cwnd by a variable factor using various buffer management techniques.
* Start working with Wireless networks.

17

**8: References**

[1] M.Allman, V.Paxson, ICSI, E.Blanton, Purdue University, September 2009.

[2] G. Riley and T. Henderson, “The ns-3 network simulator” in Modeling and Tools for Network Simulation (K. Wehrle, M. Gunes, and J. Gross, eds.), pp. 15--34, Springer Berlin Heidelberg, 2010.

[3] J. Wang, J. Wen, Y. Han, J. Zhang, C. Li and Z. Xiong, "CUBIC-FIT: A High Performance and TCP CUBIC Friendly Congestion Control Algorithm," in IEEE Communications Letters, vol. 17, no. 8, pp. 1664-1667, August 2013.

[4] Amin, Gulshan & Shahdad, Syed. (2014). TCP CUBIC-Congestion Control Transport Protocol. 3.

[5] Alrshah, Mohamed & Al-Moqri, Mohamed & Othman, Mohamed. (2019). Elastic-TCP: Flexible Congestion Control Algorithm to Adapt for High-BDP Networks. IEEE Systems Journal. PP. 1-11. 10.1109/JSYST.2019.2896195.

[6] https://haltaro.github.io/comparing-tcp-algorithms/